***Báo cáo tuần 2+3***

***Bài 5: K-means Clustering: Simple Applications***

Note : Không dám nói code của tác giả bài viết sai nhưng code đó đã cũ , 1 số cái không còn phù hợp với các bản cập nhật hiện giờ , ngoài ra cũng có 1 số lỗi nhỏ nhưng lại khiến chương trình không chạy được , em đã sửa lại , trích dẫn và lưu ý lại hết , mong rằng bài báo cáo của em có thể giúp 1 phần nào cho thầy và các học sinh khóa sau

1. ***Phân nhóm chữ số viết tay***

*Bộ cơ sở dữ liệu MNIST :*

[Bộ cơ sở dữ liệu MNIST](http://yann.lecun.com/exdb/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 đên 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.



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. (*Vì mục đích hiển thị ma trận pixel ở bên phải, tôi đã resize bức ảnh về 14x14*)

Ảnh có chứa bàn

Mô tả được tạo tự động

***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.

Trước khi áp dụng thuật toán [K-means clustering](https://machinelearningcoban.com/2017/01/01/kmeans/), 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.

***Chú ý****: Cách làm này chỉ là cách đơn giản nhất để mô tả dữ liệu ảnh bằng 1 vector. Trên thực tế, người ta áp dụng 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.*

### **Làm việc trên Python**

Download cơ sở dữ liệu MNIST từ web : https://web.archive.org/web/20220331130319/https://yann.lecun.com/exdb/mnist/ (hai file t10k-images-idx3-ubyte.gz và t10k-labels-idx1-ubyte.gz vì thư viện python-mnist cần cả hai file này để load dữ liệu từ tập test).

(nhiều khi bị lỗi , phải đổi dấu . ở 2 file trên thành dấu – mới chạy được)

**Trước tiên chúng ta cần khai báo một số thư viện:**

# *%reset*

*import* numpy *as* np

*from* mnist.loader *import* MNIST

*import* matplotlib

matplotlib.use('TkAgg')

*import* matplotlib.pyplot *as* plt

*from* sklearn.cluster *import* KMeans

*from* sklearn.neighbors *import* NearestNeighbors

*from* sklearn.preprocessing *import* normalize

*from* display\_network *import* \*

**Để hiện thị nhiều bức ảnh các chữ số cùng một lúc, tôi có dùng thêm hàm số**[**display\_network.py**](https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/kmeans/display_network.py). ( để cùng 1 thư mục với code )

**Thực hiện thuật toán K-means clustering trên toàn bộ 10k chữ số.**

**from** display\_network **import** **\***

mndata = MNIST('/home/anhvietnx1/project/Project1/MNIST') # *path to your MNIST folder*

mndata.load\_testing()  #*loads the testing images and labels from the MNIST dataset.*

X = mndata.test\_images

X0 = np.asarray(X)[:1000,:]/256.0 #*lấy 1000 mẫu đầu tiên từ X , chia 256 để chuẩn hóa về 0 or 1*

#*kích thước 1000,784 do mỗi mẫu có kích thước 1,784*

X = X0

K = 10

kmeans = KMeans(n\_clusters=K, n\_init=10).fit(X)

pred\_label = kmeans.predict(X)

print(type(kmeans.cluster\_centers\_.T))

print(kmeans.cluster\_centers\_.T.shape)

A = display\_network(kmeans.cluster\_centers\_.T, K, 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()

# *plt.savefig('a1.png', bbox\_inches='tight')*

# *a colormap and a normalization instance*

cmap = plt.cm.jet

norm = plt.Normalize(vmin=A.min(), vmax=A.max())

# *map the normalized data to colors*

# *image is now RGBA (512x512x4)*

image = cmap(norm(A))

*import* imageio

image = cmap(norm(A))

image = (image \* 255).astype(np.uint8) # *convert to uint8*

imageio.imwrite('aa.png', image)

(source code có thể được tìm thấy [tại đây](https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/kmeans/Kmeans2.ipynb))

Đây là kết quả của đoạn code trên

<class 'numpy.ndarray'>

(784, 10)

Đến đây, sau khi đã tìm được các center và phân nhóm dữ liệu vào từng cluster, tôi muốn hiển thị xem center trông như thế nào và các bức ảnh được phân vào mỗi cluster có giống nhau hay không. Dưới đây là kết quả khi tôi chọn ngẫu nhiên 20 bức ảnh từ mỗi cluster.

#*chon vai anh tu cluster*

print(type(pred\_label))

print(pred\_label.shape)

print(type(X0))

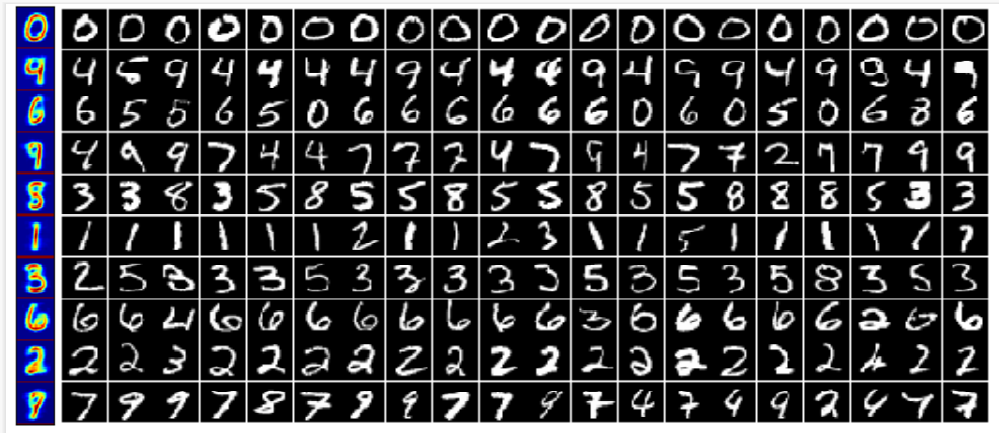
<class 'numpy.ndarray'>

(784, 10)

<class 'numpy.ndarray'>

(1000,)

<class 'numpy.ndarray'>



Mỗi hàng tương ứng với một cluster, cột đầu tiên có nền xanh bên trái là centers tìm được của các clusters (màu đỏ hơn là các pixel có giá trị cao hơn). Chúng ta thấy rằng các center đều hoặc là giống với một chữ số nào đó, hoặc là kết hợp của hai/ba chữ số nào đó. Ví dụ: center của nhóm thứ 4 là sự kết hợp của các số 4, 7, 9; của hàng thứ 7 là kết hợp của chữ số 7, 8 và 9.

Tuy nhiên, các bức ảnh lấy ra ngẫu nhiên từ mỗi nhóm trông không thực sự giống nhau. Lý do có thể là những bức ảnh này ở xa các center của mỗi nhóm (mặc dù center đó đã là gần nhất).

Tổng kết : Code trên thực hiện việc áp dụng thuật toán KMeans để phân loại dữ liệu hình ảnh số viết tay trong tập MNIST. Sau đó, nó hiển thị các trung tâm cụm (cluster centers) trên một hình ảnh. Tiếp theo, code chọn ra 20 ảnh từ mỗi cụm và hiển thị chúng trên một hình ảnh.

Cụ thể, đoạn code thực hiện các công việc sau:

* Tải tập dữ liệu MNIST và giới hạn số lượng mẫu ở đầu tiên thành 1000 mẫu.
* Sử dụng KMeans để phân loại dữ liệu thành K cụm, ở đây K=10. Sau đó, lưu trữ nhãn dự đoán của mỗi mẫu trong biến pred\_label.
* Hiển thị các trung tâm cụm trên một hình ảnh sử dụng hàm display\_network.
* Với mỗi cụm, chọn ra 20 mẫu gần nhất đến trung tâm cụm và lưu chúng vào biến X1. Tiếp theo, chọn ra 20 mẫu đầu tiên trong cụm và lưu chúng vào biến X2.
* Hiển thị các mẫu được chọn từ X2 trên một hình ảnh.

1. ***Object Segmentation (tách vật thể trong ảnh)***

***Đặt vấn đề***

Chúng ta cùng thử áp dụng thuật toán K-means clustering vào một bài toán xử lý ảnh khác: tách vật thể.

Giả sử chúng ta có bức ảnh dưới đây và muốn một thuật toán tự động nhận ra vùng khuôn mặt và tách nó ra.

Ảnh có chứa người, trang phục, ăn mặc, trẻ

Mô tả được tạo tự động

Credit ảnh: [Trọng Vũ](https://www.facebook.com/photo.php?fbid=1219980151402370&set=a.113129725420757.13101.100001711890571&type=3&theater)

### **Lên ý tưởng**

Có 3 màu chủ đạo trong bức ảnh => 3 cluster

Bức ảnh có ba màu chủ đạo: hồng ở khăn và môi; đen ở mắt, tóc, và hậu cảnh; màu da ở vùng còn lại của khuôn mặt. Vậy chúng ta có thể áp dụng thuật toán K-means clustering để phân các pixel ảnh thành 3 clusters, sau đó chọn cluster chứa phần khuôn mặt (phần này do con người làm).

Đây là một bức ảnh màu, mỗi điểm ảnh sẽ được biểu diễn bới 3 giá trị tương ứng với màu Red, Green, và Blue (mỗi giá trị này cũng là một số tự nhiên không vượt quá 255). Nếu ta coi mỗi điểm dữ liệu là một vector 3 chiều chứa các giá trị này, sau đó áp dụng thuật toán K-means clustering, chúng ta có thể có kết quả mong muốn. Hãy thử xem

***Làm việc trên Python***

**Khai báo thư viện và load bức ảnh:**

**Link download ảnh :** <https://drive.google.com/file/d/1aSLjmDCjnASsnCysy8riIUR4eCKcSckC/view?usp=share_link>

*import* matplotlib.image *as* mpimg

*import* matplotlib.pyplot *as* plt

*import* numpy *as* np

*from* sklearn.cluster *import* KMeans

img = mpimg.imread('girl3.jpg')

plt.imshow(img)

imgplot = plt.imshow(img)

plt.axis('off')

plt.show()

#*chuyển đổi kích thước của ảnh img từ (m, n, c) thành một ma trận 2 chiều X có kích thước (m \* n, c).*

# *Trong đó, m và n là chiều rộng và chiều cao của ảnh và c là số kênh màu*

X = img.reshape((img.shape[0]\*img.shape[1], img.shape[2]))

*for* K *in* [2, 5, 10, 15, 20,100]:

    kmeans = KMeans(n\_clusters=K, n\_init='auto').fit(X)

    label = kmeans.predict(X)

    # *array img4 which has the same shape as the input array X but all*

    # *its elements are initialized to zero*

    img4 = np.zeros\_like(X)

    # *replace each pixel by its center*

*for* k *in* range(K):

        img4[label == k] = kmeans.cluster\_centers\_[k]

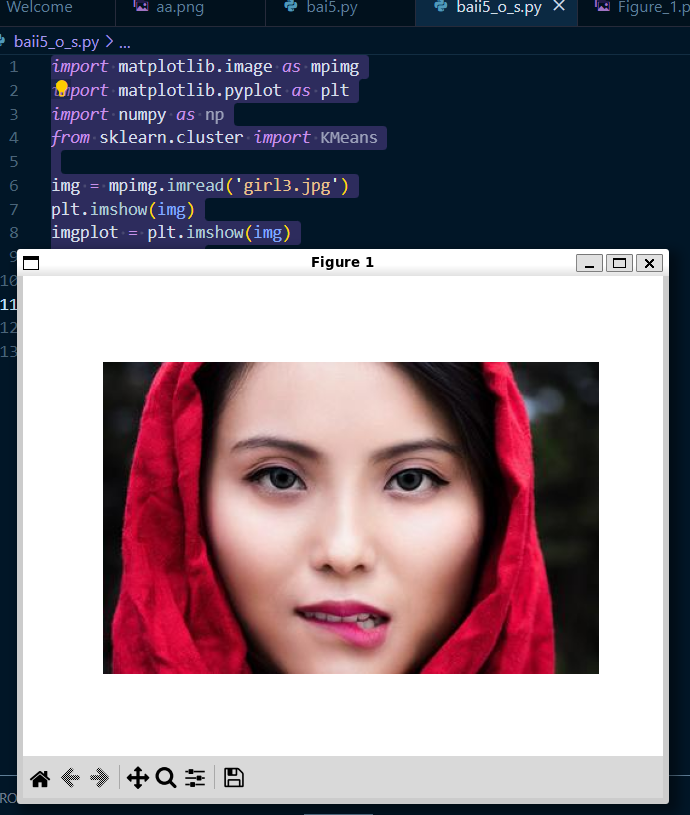
    # *reshape and display output image*

    img5 = img4.reshape((img.shape[0], img.shape[1], img.shape[2]))

    plt.imshow(img5, interpolation='nearest')

    plt.axis('off')

    plt.show()



***Tóm tắt :*** Sử dụng thư viện matplotlib để đọc ảnh và hiển thị ảnh gốc. Tiếp theo, nó chuyển đổi ảnh thành một ma trận 2 chiều X với mỗi hàng là một vector biểu diễn một điểm ảnh trong không gian RGB. Sau đó, code áp dụng thuật toán KMeans để phân cụm các điểm ảnh thành K cụm với K lần lượt là 2, 5, 10, 15, 20. Đối với mỗi giá trị K, code thay thế mỗi điểm ảnh bằng tâm của cụm tương ứng và hiển thị ảnh kết quả.

Ảnh có chứa văn bản

Mô tả được tạo tự động

***Kết quả :***

Ảnh có chứa Website

Mô tả được tạo tự độngThậm chí với k =100 , có thể thấy gần giống ảnh thật

Ảnh có chứa người, tóc, tư thế, nhìn chằm chằm

Mô tả được tạo tự động

***Kết luận :*** khi số lượng clusters tăng lên, chất lượng bức ảnh đã được cải thiện. Đồng thời, chúng ta chỉ cần lưu các centers và label của mỗi điểm ảnh là đã có được một bức ảnh nén (có mất dữ liệu).

Bài học rút ra : Có lẽ bài toán K-Clustering về phần ứng dụng này là 1 bài toán cực hay và quan trọng trong trong đời sống