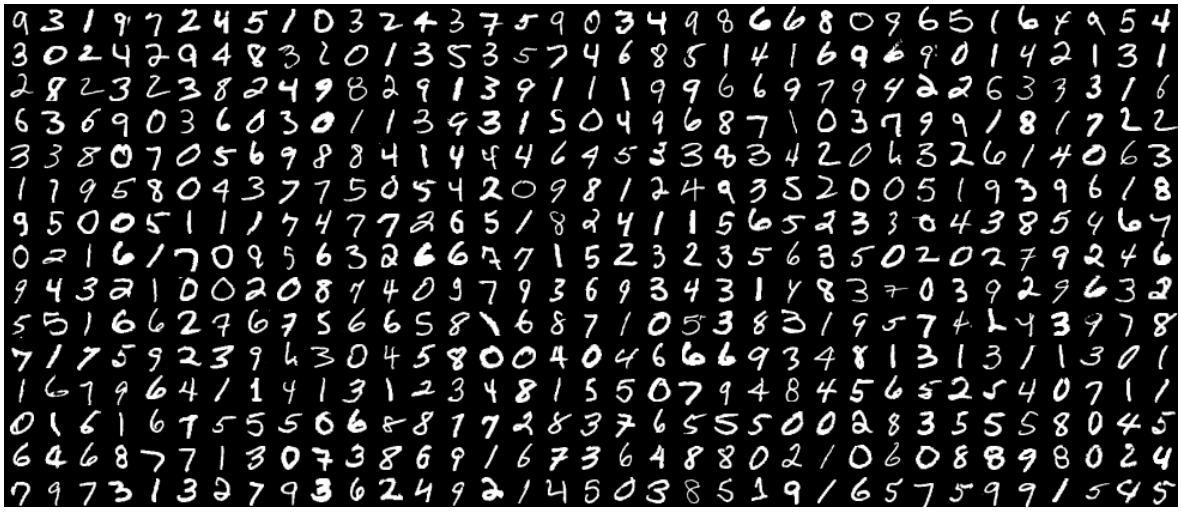
KMC: Là học không giám sát(Unsupervised learning***)***

Bộ cơ sở dữ liệu MNIST

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).

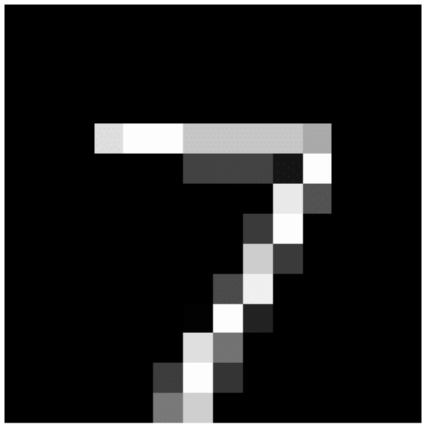
MNIST bao gồm hai tập con:

- 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.



MNIST: bộ cơ sở dữ liệu của chữ số viết tay.
(Nguồn: Simple Neural Network implementation in Ruby)

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*)



=

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0 222 254 254 198 198 198 198 170   0   0   0]
 [  0   0   0   0   0   0  66  67  67  21 254   0   0   0]
 [  0   0   0   0   0   0   0   0   0 233  83   0   0   0]
 [  0   0   0   0   0   0   0   0  59 254   0   0   0   0]
 [  0   0   0   0   0   0   0   0 205  58   0   0   0   0]
 [  0   0   0   0   0   0   0  75 240   0   0   0   0   0]
 [  0   0   0   0   0   0   3 254  35   0   0   0   0   0]
 [  0   0   0   0   0   0 224 115   0   0   0   0   0   0]
 [  0   0   0   0   0  61 254  52   0   0   0   0   0   0]
 [  0   0   0   0   0 121 207   0   0   0   0   0   0   0]]
```

Làm việc trên Python

Trước tiên các bạn vào trang chủ của MNIST để download bộ cơ sở dữ liệu này. Mặc dù trong bài này chúng ta chỉ dùng bộ dữ liệu test với 10k ảnh và không cần label, các bạn vẫn cần download cả 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.

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

numpy cho các phép toán liên quan đến ma trận. mnist để đọc dữ liệu từ MNIST. matplotlib để hiển thị hình vẽ. sklearn chính là scikit-learn mà chúng ta đã làm quen trong các bài trước. (*Về việc cài đặt các thư viện này, tôi hy vọng bạn đọc có thể Google thêm đôi chút. Nếu có khó khăn trong việc cài đặt, hãy để lại comment ở dưới bài. Lưu ý, làm việc trên Windows sẽ khó khăn hơn một chút so với Linux*)

```
# %reset

import numpy as np

from mnist import MNIST # require `pip install python-mnist`

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
```

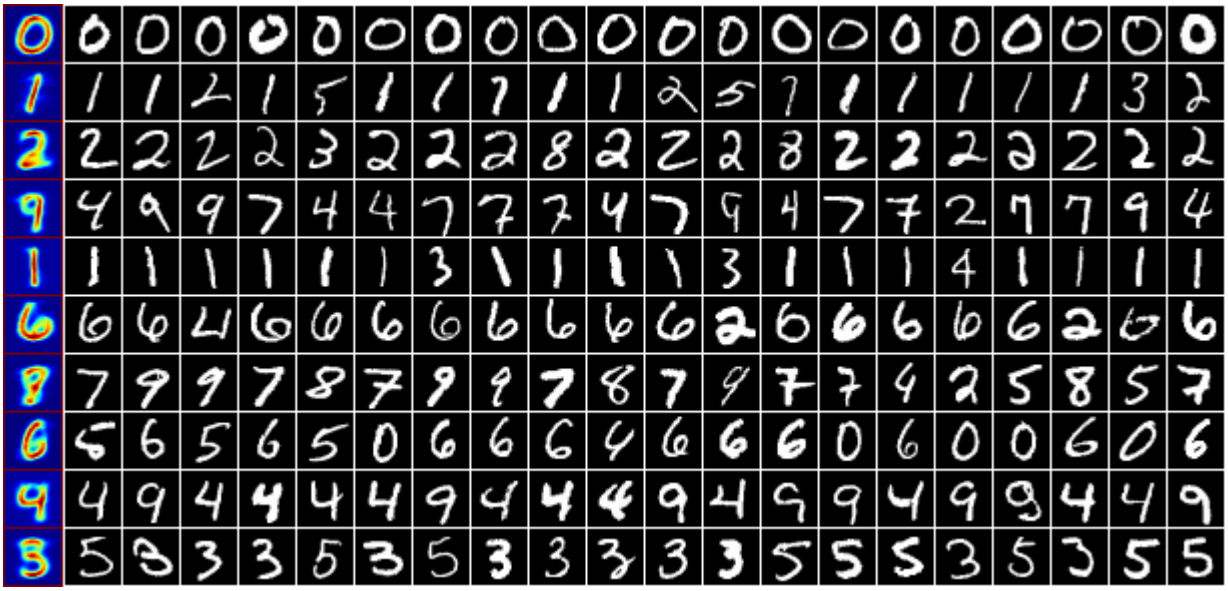
Để hiện thị nhiều bức ảnh các chữ số cùng một lúc, tôi có dùng thêm hàm số display_network.py.

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

```
from display_network import *


mndata = MNIST('../MNIST/') # path to your MNIST folder

mndata.load_testing()

X = mndata.test_images


kmeans = KMeans(n_clusters=K).fit(X)

pred_label = kmeans.predict(X)
```

(*Phần còn lại của source code có thể được tìm thấy tại đây*)

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.

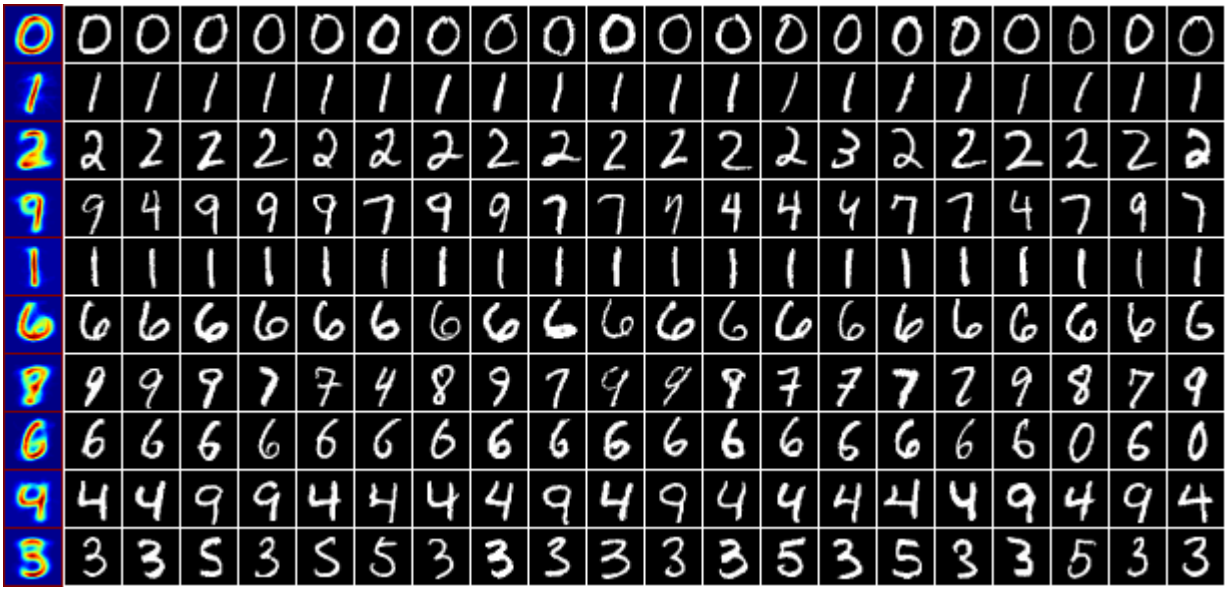


Áp dụng K-means clustering vào tập test set của bộ cơ sở dữ liệu MNIST với K = 10 cluster. Cột 1: centers của các cluster. Các cột còn lại: Mỗi hàng là 20 điểm dữ liệu ngẫu nhiên được chọn ra từ mỗi cluster.

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). Như vậy thuật toán K-means clustering làm việc không thực sự tốt trong trường hợp này. (*Thật may là vì thế nên chúng ta vẫn còn nhiều thứ để học nữa*).

Chúng ta vẫn có thể khai thác một số thông tin hữu ích sau khi thực hiện thuật toán này. Bây giờ, thay vì chọn ngẫu nhiên các bức ảnh trong mỗi cluster, tôi chọn 20 bức ảnh gần center của mỗi cluster nhất, vì càng gần center thì độ tin cậy càng cao. Hãy xem hình dưới đây:



Áp dụng K-means clustering vào tập test set của bộ cơ sở dữ liệu MNIST với K = 10 cluster. Cột 1: centers của các cluster. Các cột còn lại: Mỗi hàng là 20 điểm dữ liệu gần center nhất của mỗi cluster.

Bạn đọc có thể thấy dữ liệu trong mỗi hàng khá giống nhau và giống với center ở cột đầu tiên bên trái. Có một vài quan sát thú vị có thể rút ra từ đây:

1. Có hai kiểu viết chữ số 1, một thẳng, một chéo. Và K-means clustering nghĩ rằng đó là hai chữ số khác nhau. Điều này là dễ hiểu vì K-means clustering là thuật toán Unsupervised learning. Nếu có sự can thiệp của con người, chúng ta có thể nhóm hai clusters này vào làm một.
2. Hàng số 9, chữ số 4 và 9 được phân vào cùng 1 cluster. Sự thật là hai chữ số này cũng khá giống nhau. Điều tương tự xảy ra đối với hàng số 7 với các chữ số 7, 8, 9 được xếp vào 1 cluster. Với các cluster này, chúng ta có thể tiếp tục áp dụng K-means clustering để phân nhỏ cluster đó ra.
3. Trong clustering có một kỹ thuật thường được sử dụng là Hierarchical clustering (clustering phân tầng). Có hai loại Hierachical clustering:
    - **Agglomerative** tức "đi từ dưới lên". Ban đầu coi mỗi điểm dữ liệu thuộc 1 cluster khác nhau, sau đó các cặp cluster gần giống nhau được gộp lại làm một cluster lớn hơn. Lặp lại quá trình này đến khi nhận được kết quả chấp nhận được.
    - **Divisive** tức "đi từ trên xuống". Ban đầu coi tất cả các điểm dữ liệu thuộc cùng một cluster, sau đó chia nhỏ mỗi cluster bằng một thuật toán clustering nào đó.

2. 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.

Credit ảnh: Trọng Vũ

Lên ý tưởng

(*Lại giả sử rằng chúng ta chưa biết gì khác ngoài K-means clustering, các bạn hãy dừng vài giây để nghĩ xem chúng ta có thể xử lý thế nào. Gợi ý: Có ba màu chủ đạo trong bức ảnh.*)

Ok, có ba màu, ba clusters!

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 bởi 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:

```
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()
```

Biến đổi bức ảnh thành 1 ma trận mà mỗi hàng là 1 pixel với 3 giá trị màu

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

(*Phần còn lại của source code có thể xem tại đây*).

Sau khi tìm được các cluster, tôi thay giá trị của mỗi pixel bằng center của cluster chứa nó, và được kết quả như sau:

Ba màu: Hồng, đen, và màu da đã được phân nhóm. Và khuôn mặt có thể được tách ra từ phần có màu da (và vùng bên trong nó). Vậy là K-means clustering tạo ra một kết quả chấp nhận được.

3. Image Compression (nén ảnh và nén dữ liệu nói chung)

Để ý thấy rằng mỗi một pixel có thể nhận một trong số 2563=16,777,2162563=16,777,216 (16 triệu màu mà chúng ta vẫn nghe khi quảng cáo màn hình). Đây là một số rất lớn (tương đương với 24 bit cho một điểm ảnh). Nếu ta muốn lưu mỗi điểm ảnh với một số bit nhỏ hơn và chấp nhận mất dữ liệu ở một mức nào đó, có cách nào không nếu ta chỉ biết K-means clustering?

Câu trả lời là có. Trong bài toán Segmentation phía trên, chúng ta có 3 clusters, và mỗi một điểm ảnh sau khi xử lý sẽ được biểu diễn bởi 1 số tương ứng với 1 cluster. Tuy nhiên, chất lượng bức ảnh rõ ràng đã giảm đi nhiều. Tôi làm một thí nghiệm nhỏ với số lượng clusters được tăng lên là 5, 10, 15, 20. Sau khi tìm được centers cho mỗi cluster, tôi thay giá trị của một điểm ảnh bằng giá trị của center tương ứng:

```
for K in [5, 10, 15, 20]:
    kmeans = KMeans(n_clusters=K).fit(X)
```

```
    label = kmeans.predict(X)


    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()
```

Kết quả:

k =5

K = 10

K = 15

K = 20

Bạn đọc có thể quan sát là 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).

4. Thảo luận

1. Với một thuật toán K-means clustering đơn giản, chúng ta đã có thể áp dụng nó vào các bài toán thực tế. Mặc dù kết quả chưa thực sự như ý, chúng ta vẫn thấy được tiềm năng của thuật toán đơn giản này.
2. Cách thay một điểm dữ liệu bằng center tương ứng là một trong số các kỹ thuật có tên chung là Vector Quantization (VQ). Không chỉ trong nén dữ liệu, VQ còn được áp dụng rộng rãi trong các thuật toán tạo Feature Vector cho các bài toán Phân loại (classification).
3. Các thuật toán trong bài toán này được áp dụng cho xử lý ảnh vì việc này giúp tôi dễ dàng minh họa kết quả hơn. Các kỹ thuật này hoàn toàn có thể áp dụng cho các loại cơ sở dữ liệu khác.

6. Tài liệu tham khảo

Pattern Recognition and Machine Learning - Christopher Bishop

Source: web: https://machinelearningcoban.com/2017/01/04/kmeans2/