REPORT PROJECT 1
COLOR COMPRESSION

Toán ứng dụng và thống kê trong tin học

Lê Vũ Ngân Trúc

MSSV: 21127709

Lớp: 21CLC01

MỤC LỤC

1. TỔNG QUAN VỀ K-MEANS

Ý tưởng

K-Means clustering là một trong những thuật toán học tập không giám sát thường được sử dụng được sử dụng để phân vùng một tập dữ liệu thành một tập *k* cụm, trong đó *k* phải được người dùng chỉ định trước. Mỗi điểm dữ liệu có *n* tính chất (properties). Mục tiêu của thuật toán này là phân loại các điểm dữ liệu hiện có thành các cụm sao cho:

1. Tính chất của các điểm dữ liệu trong cùng một cụm càng giống nhau càng tốt
2. Tính chất của các điểm dữ liệu từ các cụm khác nhau càng khác nhau càng tốt

Các bước cơ bản trong thuật toán k-means gồm:

1. Người dùng chỉ định số lượng cụm *k* và số lần lặp tối đa *max_iterations*
2. Chọn ngẫu nhiên *k* điểm khác nhau từ tập dữ liệu làm centroid (tâm cụm) ban đầu
3. Gán mỗi điểm dữ liệu cho tâm gần nhất, thường theo khoảng cách Euclide
4. Tính toán các trọng tâm mới bằng cách lấy giá trị trung bình của tất cả các điểm dữ liệu thuộc về cụm
5. Lặp lại bước 3 và 4 cho đến khi nó hội tụ, tức là không thay đổi vị trí tâm hoặc vượt quá số *max_iterations* đã được chỉ định ở bước 1

Như đã đề cập bên trên, thuật toán k-means được ứng dụng trong rất nhiều chương trình, một số ví dụ phổ biến nhất phải kể đến:

- Nén màu ảnh
- Chia cụm người dùng có sở thích giống nhau (phân đoạn khách hàng)
- Thống kê số liệu theo cụm
- ...

Trong đồ án này, chúng ta sẽ ứng dụng ý tưởng thuật toán k-means để cài đặt chương trình nén màu ảnh

Tính dừng và tính hội tụ

Trong các tài liệu tham khảo, sự hội tụ của thuật toán là có tồn tại khi các tâm cụm sau 1 lần lặp không có sự thay đổi. Tuy nhiên không phải bao giờ sự hội tụ này cũng nhanh chóng, đôi khi nó tiến đến vô hạn, chính vì vậy ta cần xác định con số max iterations tức số lần lặp tối đa hoặc một con số delta chấp nhận được giữa 2 lần lặp với nhau

2. PHÂN TÍCH Ý TƯỞNG K-MEANS TRONG NÉN MÀU ẢNH

Cấu tạo của một bức ảnh

Để ứng dụng thuật toán k-means trong đồ án nén màu ảnh, chúng ta cần phân tích các yếu tố đầu vào của bài toán.

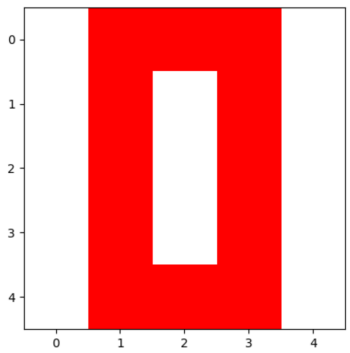
Một bức ảnh được cấu tạo bởi các ô vuông điểm ảnh pixel với kích cỡ 1x1 ghép lại tạo thành một bức ảnh lớn với kích cỡ *width x height* hay còn gọi là một ma trận điểm ảnh. Một điểm ảnh có thể được biểu diễn dưới dạng nhiều hệ số khác nhau như HEX, RGB, HSL hay HSV,...

Trong bài toán này ta sẽ sử dụng hệ RGB. RGB là hệ gồm ba số nguyên không dấu (dương) 8 bit [0...255] hoặc được chia tỷ lệ thành ba số float không dấu (dương) [0...1]. Ba giá trị này nhằm xác định thông số của mỗi một màu đỏ (red), xanh lục (green), xanh lam (blue) trong ảnh. Ngoài ra ta còn có hệ RGBA với A (alpha) biểu diễn cho hiệu ứng trong suốt.

Như vậy, đối chiếu từ 1 điểm ảnh, ta rút ra được:

- Một điểm ảnh là một điểm dữ liệu
- Một điểm ảnh có 3 channel (RGB) ứng với 3 tính chất (properties) của 1 điểm dữ liệu

Như vậy, từ một bức ảnh ta có thể hiểu dưới dạng số là một ma trận 3 chiều
width x height x channels



[[[255, 255, 255], [255, 0, 0], [255, 0, 0], [255, 0, 0], [255, 255, 255]],

[[255, 255, 255], [255, 0, 0], [255, 255, 255], [255, 0, 0], [255, 255, 255]],

[[255, 255, 255], [255, 0, 0], [255, 255, 255], [255, 0, 0], [255, 255, 255]],

[[255, 255, 255], [255, 0, 0], [255, 255, 255], [255, 0, 0], [255, 255, 255]],

[[255, 255, 255], [255, 0, 0], [255, 0, 0], [255, 0, 0], [255, 255, 255]]]

K-means và thuật toán nén ảnh

Ý tưởng chính của thuật toán nén ảnh gồm các bước:

1. Reshape tập tin ảnh thành ma trận thông số
2. Khởi tạo đối tượng k-means với dataset là ma trận thông số
3. Chạy thuật toán k-means với init centroids là các màu ngẫu nhiên hoặc chọn từ ma trận thông số
4. Tạo ma trận thông số cho ảnh nén với các màu mới sau khi chạy thuật toán k-means
5. Reshape từ ma trận thông số thành tập tin ảnh

Kết quả của chương trình là một ảnh nén với *k* màu tương ứng với giá trị *k cluster* được chỉ định

3. CÀI ĐẶT CHƯƠNG TRÌNH:

Thư viện cần dùng:

```
import matplotlib.pyplot as plt      # Hiển thị hình ảnh
import numpy as np                   # Tính toán
from PIL import Image                # Đọc, lưu file ảnh
from io import BytesIO               # Tính kích cỡ ảnh
import time                          # Tính toán thời gian chạy của thuật toán
```

Chương trình sẽ gồm các phần chính:

- *Class KMeans* với các method và attribute để chạy thuật toán
- Hàm *compressColor()* chạy thuật toán nén ảnh
- Hàm *main()* đọc thông tin của ảnh, reshape và lưu ảnh đã được nén

Input	- Tên tập tin ảnh (nếu cùng 1 thư mục) hoặc đường dẫn tập tin - Định dạng đầu ra của tập tin ảnh (png hoặc pdf)
Output	- Ảnh đã nén theo định dạng đầu ra người dùng chọn, có các mức độ nén ứng với giá trị của các k màu (3, 5, 7)

Class KMeans:

```
class KMeans():
    # attributes: các thuộc tính cần thiết của thuật toán kmeans
    int k_cluster
    int max_iterations
    array all_centroids
    array all_labels
    array init_centroids

    # constructor khởi tạo các giá trị ban đầu của thuật toán
    def __init__

    # methods
    def fit
    def get_random_centroids
    def get_labels
    def should_stop
    def get_centroids
```

Attributes và hàm constructor	
k_cluster : int	Số lượng cụm người dùng chỉ định, ở đây ứng với số lượng màu ảnh còn lại sau khi nén ảnh
max_iterations : int	Số lượng lần lặp tối đa của thuật toán mà người dùng mong muốn, default = 10
all_centroids = []	Một dãy với mỗi phần tử *centroids[i][j] = [](3)* trong đó: + *i (< max_iterations)*: số lần lặp thứ i+1, i=0 ứng với khởi tạo random init_centroids + *j (< k_cluster)*: tâm cụm thứ j trong số k tâm cụm + *[](3)*: dãy các channels tính chất của 1 điểm dữ liệu (với đồ án này là dãy gồm 3 phần tử)
all_labels = []	Dãy tất cả các nhãn dữ liệu là các index của tâm cụm mà điểm dữ liệu đó thuộc về. *all_labels[i][j] = k* với: + *i (< max_iterations):* số lần lặp thứ *i+1, i=0* ứng với random init centroids + *j (< size of dataSet):* điểm dữ liệu thứ j + *k (< k_cluster):* index của điểm dữ liệu thứ j

init_centroids: str	Loại khởi tạo init centroids: + *'random'*: các tâm cụm có thông số màu ngẫu nhiên, không phụ thuộc dataSet + *'in_pixels'*: các tâm cụm được chọn ngẫu nhiên trong dataSet
Methods	
def **fit(self, dataSet):** *numFeature* ← số tính chất của các điểm dữ liệu *centroids* ← get_random_centroids *all_centroids*.append() while not *should_stop()*: *oldCentroids* ← centroids *iteration* ← iterations + 1 *labels* ← get_labels() *all_labels*.append() *centroids* ← get_centroids() *all_centroids*.append() return centroids	Xác định số lượng numFeatures (số tính chất hay channel của bộ dữ liệu) bằng *dataSet.shape[1]* Chạy vòng lặp cho đến khi hàm *should_stop* trả về true. Mỗi lần lặp: + Lưu lại các centroids cũ để kiểm tra shoud_stop + Tăng biến đếm + Lấy nhãn mới cho mỗi điểm dữ liệu + Đo lại các tâm cụm mới

def **get_random_centroids(self, k_cluster, numFeatures, dataSet):**
if *init_centroids* = 'random:
centroids ← [random(0...255)]
if *init_centroids* = 'in_pixels':
centroids ← [random trong bộ dữ liệu]
return centroids
	Nếu init_centroids = 'random', mỗi phần tử sẽ là giá trị random từ 0 → 255
	Nếu init_centroids = 'in_pixels', ta lấy k điểm bất kỳ trong bộ dữ liệu

def **get_labels(self, dataSet, centroids):** *labels* ← empty for x in dataSet: *distances* ← các khoảng cách từ điểm đến centroids tương ứng *label* ← index của min(distances) *labels*.append(label) return labels	Với mỗi điểm dữ liệu, ta tính khoảng cách từ điểm dữ liệu đó đến các tâm cụm centroids bằng khoảng cách Euclid. Sau đó chọn khoảng cách nhỏ nhất và lưu lại index của tâm cụm đó
def **should_stop(oldCentroids, centroids, iterations):** if *iterations* > max_iterations: return True return *oldCentroids == centroids*	Nếu tâm cụm không đổi hoặc đã vượt quá số lần lặp tối đa thì trả về True, thuật toán nên dừng
def **get_centroids(dataSet, labels, current_centroids, k_cluster):** *centroids* ← empty	

for j ← 0 to k_cluster: *idx_j* ← dãy các điểm dữ liệu có nhãn j *centroids[j]* ← trung bình các điểm dữ liệu idx_j return centroids	Với mỗi tâm cụm, ta lấy vị trí của các điểm dữ liệu thuộc tâm cụm đó, tính trung bình và gán lại giá trị trung bình đó cho tâm cụm

Hàm compressColor():

```
def compressColor(k_cluster, ori_pixels, option):
    # Khởi tạo instance kmeans
    kmeans ← KMeans(k_cluster, max_iterations, init_centroid = option)

    # Lấy k cụm cuối cùng sau khi thuật toán hoàn thành
    centroids ← kmeans.fit(ori_pixels)
    # Lấy k nhãn điểm dữ liệu
    labels ← kmeans.get_lables(ori_pixels, centroids)

    # Chuyển đổi từ nhãn điểm dữ liệu thành giá trị của tâm cụm mà
    # điểm đó thuộc về
    for x in labels
        result.append(centroids[x])
    return result
```

Hàm main():

```
def main():
    img ←  file ảnh muốn nén (dùng hàm input)
    ori_img ← img convert từ RGBA sang hệ RGB
    file_type ← thể loại ảnh muốn nén là png hay pdf  (dùng hàm input)
    # Các hàm tính toán debug khác như in dung lượng, phương sai,...

    # Số lượng cụm k thử nén
    k_cluster ← [3, 5, 7, 20]
    for k in k_cluster:
        new_img ← compressColor(ori_img)
        # Các hàm tính toán và debug khác như in dung lượng, phương sai,...
        # ...
        new_img.save(<tên ảnh>)
```

Các hàm khác:

def **imageByteSize(img):**	Nhận 1 file ảnh và trả về kích thước file ảnh đó theo KB để so sánh kích cỡ trước và sau khi nén

4. BỘ TEST KẾT QUẢ:

Test 1: Ảnh gồm 2 màu chủ đạo, độ phân giải thấp

Ảnh gốc
Dung lượng: ~52.52KB
Số màu: 17 399
Kích cỡ ảnh: 200 x 233 px

Ảnh nén với **k = 3**
Dung lượng: ~3.346KB
Giảm 93,63% dung lượng
Thời gian chạy: 38,91s

Ảnh nén với **k = 5**
Dung lượng: ~4,93KB
Giảm 90,6% dung lượng
Thời gian chạy: 31,62s

Ảnh nén với **k = 7**
Dung lượng: ~7,65KB
Giảm 85,43% dung lượng
Thời gian chạy: 42,93s

Ảnh nén với **k = 20**
Dung lượng: ~14,30KB
Giảm 72,78% dung lượng
Thời gian chạy: 85,47s

Test 2: Ảnh gồm nhiều màu rực rỡ, đổ gradient, kích cỡ ảnh vẫn thấp

Ảnh gốc
Dung lượng: ~51,72KB
Số màu: 19 318
Kích cỡ ảnh: 200 x 196 px

Ảnh nén với **k = 3**
Dung lượng: ~3,60KB
Giảm 93,03% dung lượng
Thời gian chạy: 113,6s

Ảnh nén với **k = 5**
Dung lượng: ~5,66KB
Giảm 89,0% dung lượng
Thời gian chạy: 76,1s

Ảnh nén với **k = 7**
Dung lượng: ~6,46KB
Giảm 87,52% dung lượng
Thời gian chạy: 119,83s

Ảnh nén với **k = 20**
Dung lượng: ~13,00KB
Giảm 74,85% dung lượng
Thời gian chạy: 141,26s

Test 3: Ảnh chụp nhiều màu, chất lượng cao

Ảnh gốc
Dung lượng: ~470.34KB
Số màu: 119 127
Kích cỡ ảnh: 600 x 400 px

Ảnh nén với **k = 3**
Dung lượng: ~27,67KB
Giảm 94,12% dung lượng
Thời gian chạy: 140,6s

Ảnh nén với **k = 5**
Dung lượng: ~50,78KB
Giảm 89,20% dung lượng
Thời gian chạy: 133,4s

Ảnh nén với **k = 7**
Dung lượng: ~67,73KB
Giảm 85,6% dung lượng
Thời gian chạy: 173,56s

Ảnh nén với **k = 20**
Dung lượng: ~124,81KB
Giảm 73,46% dung lượng
Thời gian chạy: 278,16s

5. NHẬN XÉT VÀ MỞ RỘNG:

So sánh các kết quả thu được từ bộ thử nghiệm

Kết quả đầu tiên dễ dàng rút ra đó chính là các hàm dung lượng theo số cụm, theo số cụm đều **biến thiên bậc nhất** (tăng hoặc giảm tương ứng khi k thay đổi)

Riêng thời gian chạy không phải là hàm biến thiên bậc nhất, không phải bao giờ k lớn cũng có thời gian chạy lâu hơn (VD ở bộ test 3 giữa k=3 và k=5 cho ra 140s và 133s)

Các tính chất của ảnh ảnh hưởng đến thời gian chạy là:

- Số lượng màu ảnh
- Tính đa dạng trong màu ảnh (bao nhiêu tông màu)
- Kích cỡ và dung lượng
- Mức độ nén (số k cluster)

Có thể thấy khi chọn *k* thấp, kết quả ảnh cho ra chỉ nhìn thấy được bố cục cơ bản và màu chủ đạo của 1 bức ảnh, tuy thời gian chạy nhanh và giảm được nhiều dung lượng hơn nhưng cũng mất đi rất nhiều thông tin ảnh.

Thời gian

Ở đồ án này, ta thử nghiệm với các k nhỏ (3, 5, 7, 20) nhưng vẫn cho ra các thời gian chạy khá lâu dù ảnh có kích cỡ thấp và ít tông màu chủ đạo (ở bộ test 1 với k=3 vẫn cho ra thời gian chạy trung bình 38s!!!)

Chính vì vậy trong các mô hình học máy, người ta thường dùng ý tưởng k-means như một cơ sở và thêm kèm với các thuật toán khác, heuristic để xác định delta chấp nhận được, số lần lặp, gom cụm và thay nhãn dữ liệu nhanh chóng hơn,...

Ngoài ra thuật toán cũng khởi tạo bằng cách chọn centroids random trong bộ dữ liệu hoặc random theo màu sắc, ảnh hưởng đến việc tốn một vài lần lặp để định vị lại tâm cụm chính xác hơn. Có thể cải tiến bằng một hàm chọn vị trí tâm cụm hiệu quả từ đầu bằng cách xem xét và đánh giá dữ liệu đầu vào.

Một ý tưởng hay ho khác trong nén ảnh

Đề cập ở mục 6 (số thứ tự 1) đã đề xuất một phương án chạy thuật toán k-means trong nén ảnh khá hay ho. Thay vì cách so 3 kênh màu theo số thông thường, thuật toán chuyển đổi từ 3 kênh màu đó thành tên 1 màu sắc la tinh. Như vậy 1 cụm gọi là chấp nhận được khi màu của tên tâm cụm giống với màu của tất cả các điểm dữ liệu thuộc tâm cụm đó.

6. THAM KHẢO:

1. Giải thích ứng dụng k-means trong nén hình ảnh
2. Giải thích thuật toán k-means
3. Hướng dẫn cài đặt k-means

7. SOURCE CODE:

Tham khảo tại NganTrucLe/color-compression (github.com)