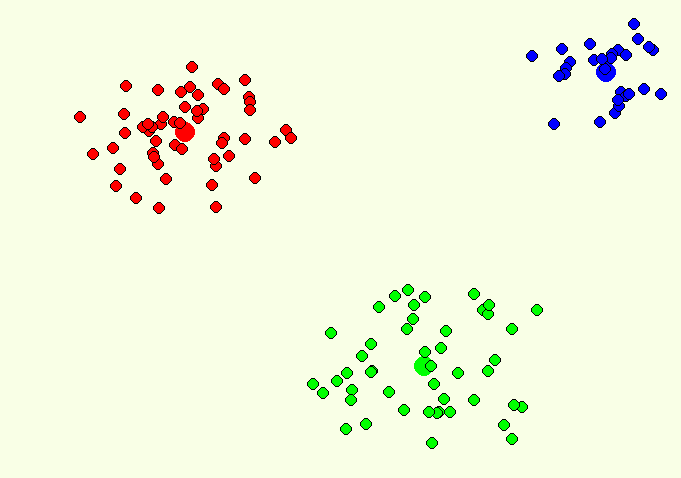
Báo cáo

Học phần: Trí tuệ nhân tạo

Giáo viên hướng dẫn: Nguyễn Xuân Phúc

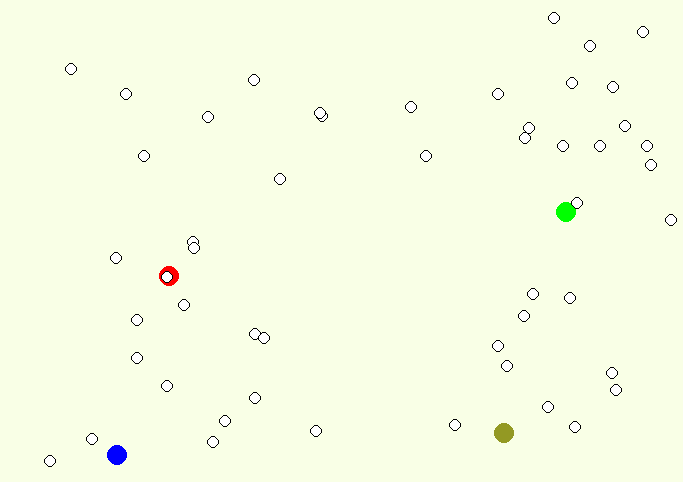
1. ***Giới thiệu về thuật toán phân cụm K-means ( K-means clustering)***

* Thuật toán được James MacQueen sử dụng lần đầu vào năm 1967.
* Là 1 thuật toán cơ bản và thông dụng trong *unsupervised learning.*
* Trong thuật toán K-means, chúng ta không biết nhãn (label) của từng điểm dữ liệu. Mục đích là làm thế nào để phân các dữ liệu thành các cụm (cluster) khác nhau sao cho dữ liệu trong cùng 1 cụm có tính chất tương tự nhau.
* Ví dụ đơn giản về cụm (cluster) là tập hợp các điểm gần nhau trong một không gian nào đó. Hình dưới đây biểu diễn ví dụ về 3 cluster:

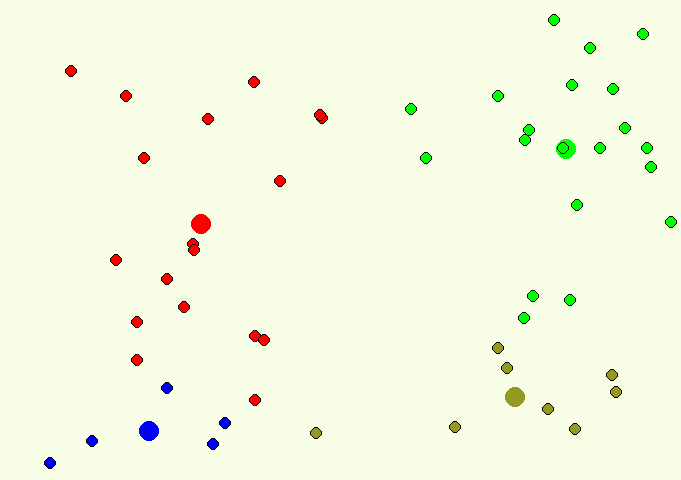


1. ***Tóm tắt thuật toán.***

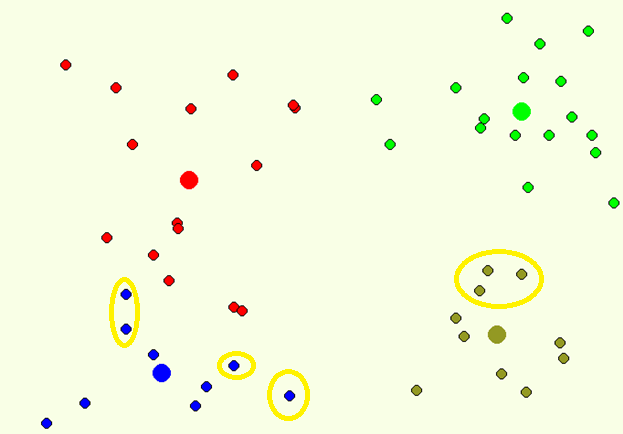
* Input : Dữ liệu X và số lượng cluster cần tìm K
* Output: Các center M và label cho từng điểm dữ liệu Y
* Các bước thực hiện:
  + Chọn K điểm bất kì làm các center ban đầu
  + Phân mỗi điểm dữ liệu vào cluster có center gần nó nhất và gán nhãn cho các điểm dữ liệu trên tương ứng với các center
  + Nếu việc gán nhãn vào từng cluster không thay đổi so với vòng lặp trước thì ta dừng thuật toán
  + Cập nhật center cho từng cluster bằng cách lấy trung bình cộng của tất cả các điểm dữ liệu đã được gán vào cluster đó sau bước 2
  + Quay lại bước 2
* Thuật toán sẽ dừng lại sau một số vòng lặp hữu hạn. Thật vậy , vì hàm mất mát là 1 số dương và sau bước 2 và bước 3, giá trị của hàm mất mát sẽ bị giảm đi nên sau hữu hạn bước thuật toán sẽ hội tụ. Hơn nữa, số lượng cách phân nhóm cũng là hữu hạn nên đến một lúc nào đó, hàm mất mát sẽ không thay đổi và thuật toán dừng lại.
* Sau đây là ví dụ với K =4
  + Bước 1: Khởi tạo dữ liệu và khởi tạo ngẫu nhiên 4 center



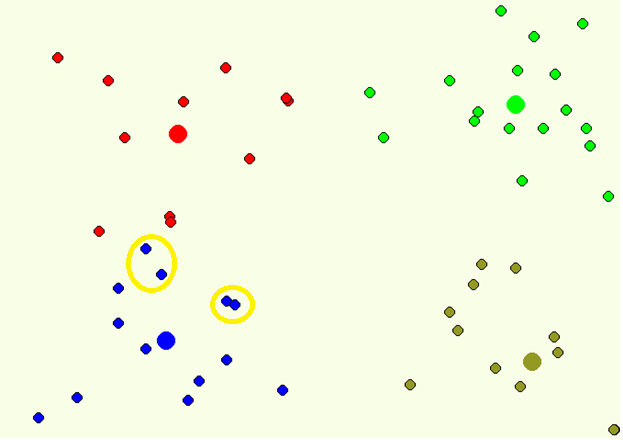
* + Bước 2: phân các điểm vào cluster và gán nhãn cho chúng



* + Bước 3: điều chỉnh lại các center và cập nhật lại nhãn cho các điểm dữ liệu (chạy lại bước 2)

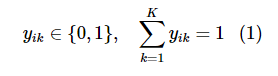


* + Bước 4: Làm tương tự như bước 3 và sau đó đây cũng là bài toán khi hội tụ.



1. ***Phân tích toán học.***

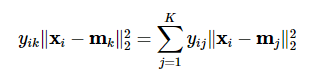
* **Một số kí hiệu toán học:**
  + Giả sử có N điểm dữ liệu X = [x1, x2,…,xN] và K < N là số cluster chúng ta muốn phân chia. Chúng ta cần tìm các center m1 , m2 , …, mK và label của mỗi điểm dữ liệu
  + Với mỗi điểm dữ liệu xi đặt yi = [yi1, yi2,…, yiK] là label vector của nó, giả sử xi được phân vào cluster k thì yik = 1 và các yij = 0 với j != k
    - Ví dụ: nếu 1 điểm có label vector là [1,0,0,…,0] thì nó thuộc cluster 1
  + Ràng buộc của yi có thể được viết lại như sau:



* **Hàm mất mát và tối ưu:**
  + Nếu ta coi center mk là center của một cluster và ước lượng tất cả các điểm được phân vào bởi mk, thì một điểm dữ liệu xi được phân vào cluster k sẽ bị sai số là (xi – mk). Sau số này phải có giá trị tuyệt đối nhỏ nhất khi và chỉ khi đại lượng sau nhỏ nhất:



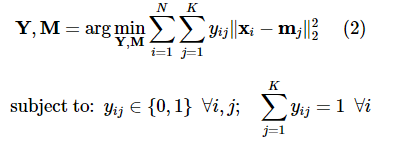
* + Hơn nữa, vì xi được phân vào cluster k nên yik =1, yij = 0 với j != k. Khi đó biểu thức trên sẽ được viết lại là:



* + Khi đó sai số cho toàn bộ dữ liệu là:

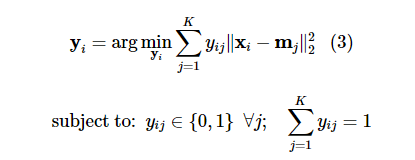


* + Bài toán cần tối là:

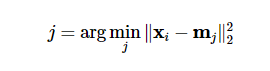


(giải thích về argmin: Nếu f(x) = x2 – 2x + 1 = (x – 2)2 có giá trị nhỏ nhất của hàm số là 0, để thỏa mãn điều kiện f(x) = 0 thì x =1. Trong ví dụ này minx f(x) = 0 và argminx xi =1)

* **Thuật toán tối ưu hàm mất mát:**
  + Cố định M, tìm Y (cố định các điểm centers và tìm các label vector để hàm mất mát đạt giá trị nhỏ nhất)
    - Khi các centers đã cố định, bài toán label vector cho toàn bộ dữ liệu có thể chia nhỏ thành bài toán tìm label vector cho từng điểm dữ liệu xi như sau:

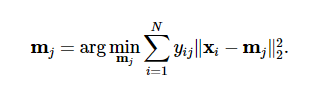


* + - Vì chỉ có 1 phần tử của label vector yi bằng 1 nên bài toán (3) ở trên có thể viết dưới dạng đơn giản hơn như sau:

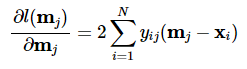


Vì là bình phương khoảng cách từ điểm xi tới center mj, ta có thể kết luận rằng mỗi điểm xi thuộc vào cluster có center gần nó nhất. Từ đó ta có label vector của từng điểm dữ liệu

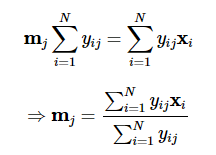
* + Cố định Y, tìm M (giả sử đã tìm được cluster cho từng điểm, tìm center mới cho mỗi cluster để hàm mất mát có giá trị nhỏ nhất)
    - Khi chúng ta đã xác định được label vector cho từng điểm dữ liệu, bài toán tìm center cho mỗi cluster được rút gọn thành:



* + - Ta có thể tìm nghiệm bằng cách tìm đạo hàm bằng 0, vì hàm cần tối ưu là hàm liên tục và có đạo hàm tại mọi điểm.
    - Đặt l(mj) là hàm bên trong dấu argmin ( ), ta có đạo hàm:



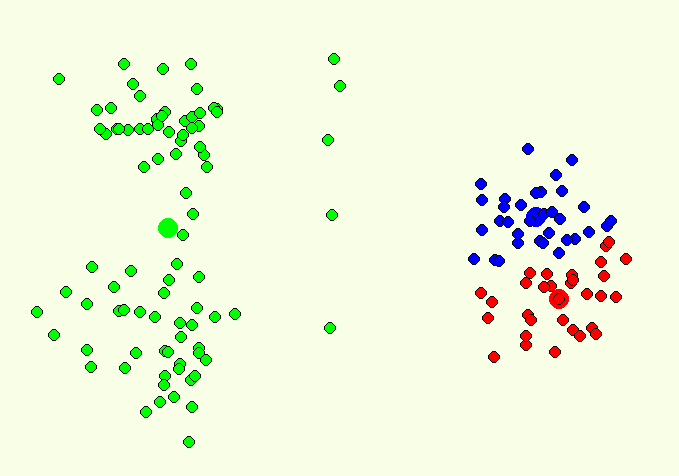
* + - Giải phương trình đạo hàm = 0, ta có:



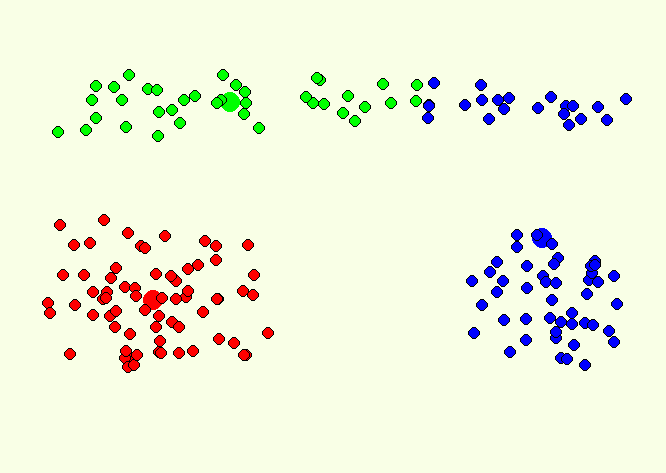
* + - Khi đó, mj là trung bình cộng của các điểm trong cluster j.

1. ***Một số hạn chế.***

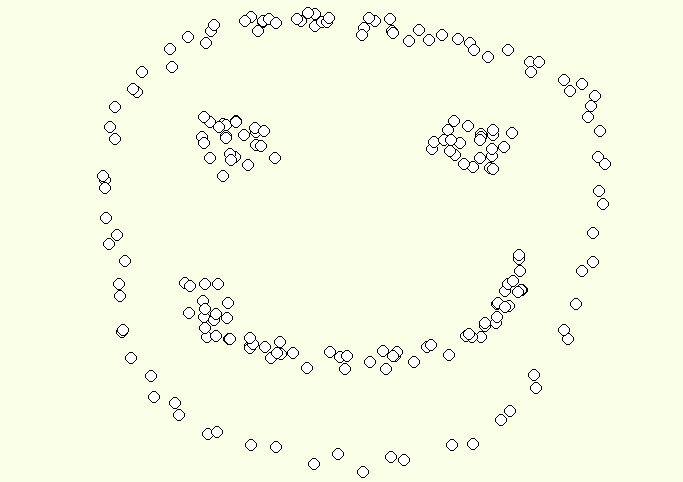
* **Cần biết trước số lượng cluster:**
  + Khi chạy thuật toán trên, chúng ta cần biết đại lượng K là số lượng cluster. Trong thực tế, nhiều trường hợp không xác định được giá trị này.
  + Có 1 số phương pháp giúp xác định cluster như Elbow method,…..
* **Nghiệm cuối cùng phụ thuộc vào centers được khởi tạo ban đầu:**
  + Tùy vào các center ban đầu mà thuật toán có thể có tốc độ hội tụ chậm
  + Cho nghiệm không chính xác ( chỉ là local minimum – điểm cực tiểu chứ không phải giá trị nhỏ nhất ), ví dụ như:



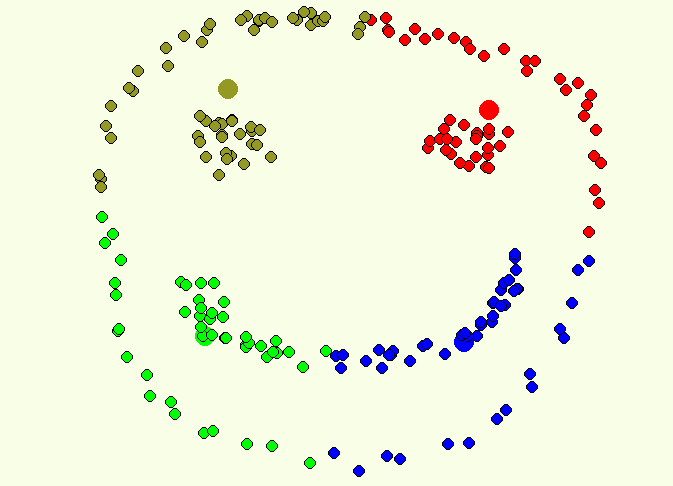
* + *Một vài các khắc phục đó là:*
    - Chạy K-means clustering nhiều lần với các center ban đầu khác nhau rồi tìm ra hàm mất mát có giá trị nhỏ nhất
    - K-means ++
* **Các cluster cần có dạng hình tròn**
  + Do các khoảng cách từ các điểm tới các center là khoảng cách Euclid nên khi có 1 vài điểm nằm trong cùng 1 cụm nhưng lại gần center khác hơn gây ra mất mát dữ liệu
  + Ví dụ 1 cluster có dạng hình dẹt:



* **Khi một cluster nằm trong 1 cluster khác**
  + *Ví dụ:*



* + Ta thấy có 1 khuôn mặt, thông thường sẽ được phân thành 4 cụm là 2 mắt, miệng và đường bao quanh khuôn mặt, nhưng vì mắt miệng nằm trong khuôn mặt nên:



1. ***Bài toán phân vùng màu ảnh và bài toán nén ảnh (image compression)***

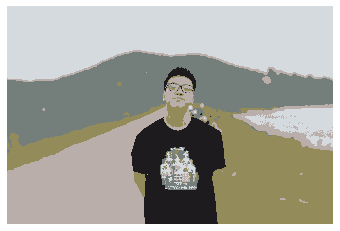
* **Giới thiệu về pixel (điểm ảnh):** 
  + Được biểu diễn bởi 3 giá trị tương ứng với màu Red, Green, Blue (mỗi giá trị này sẽ nằm trong khoảng từ 0-255). Vì vậy trong bài toán này ta coi mỗi pixel là một vector 3 chiều.
  + Mỗi pixel có thể nhận 2563 = 16,777,216 ( 16 triệu màu khi được nghe giới thiệu về các loại màn hình, TV,..). Đây là một số rất lớn khi xử lý dữ liệu gây tốn thời gian và tài nguyên. Vì vậy, đôi khi có thể nén bức ảnh lại, tuy độ chính xác không cao như ảnh gốc nhưng bù lại sẽ cho chúng ta tốc độ xử lý cao hơn.
* Đối với bài toán nén ảnh, ta có thể xử dụng thuật toán K-means, với mỗi điểm ảnh sau khi xử lý sẽ được biếu diễn bởi 1 số tương ứng với 1 cluster. Tuy nhiên chất lượng bức ảnh đó sẽ bị giảm đi.
* Sau đây là 1 ví dụ về bức ảnh với số lượng cluster thay đổi từ 2,5,10,15,20. Sau khi tìm được center cho mỗi cluster, giá trị của mỗi điểm ảnh sẽ được thay bằng giá trị của center tương ứng:
  + Ảnh gốc:



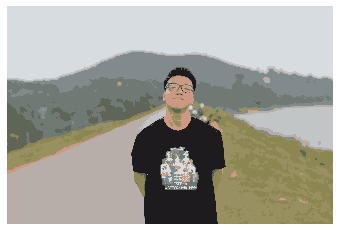
* + K = 2: Ảnh chỉ hiển thì 2 màu đen trắng



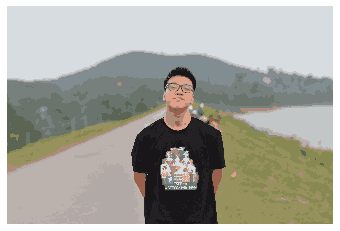
* + K=5:



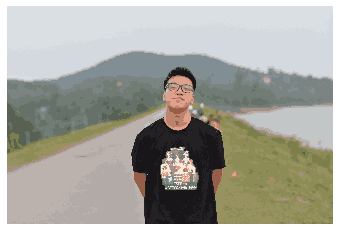
* + K=10:



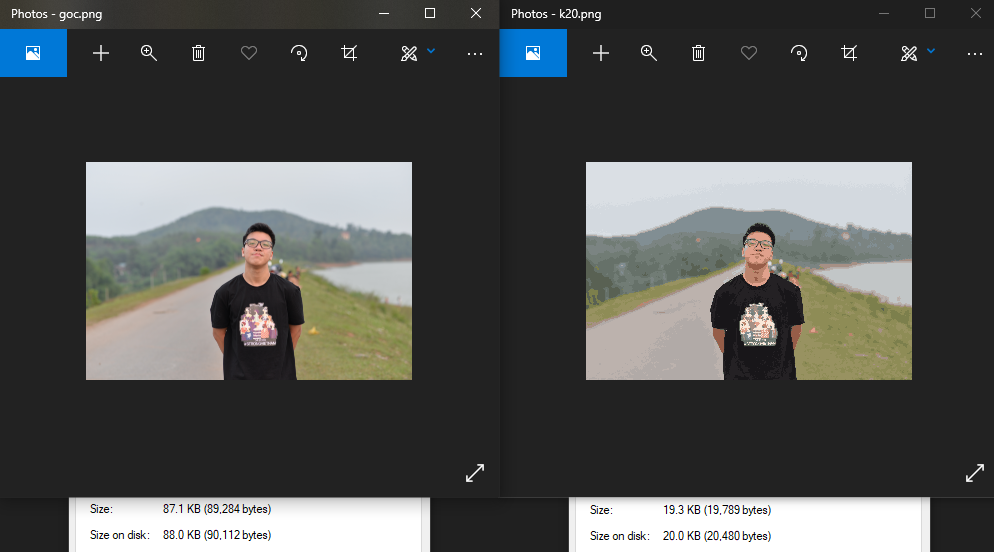
* + K = 15:



* + K = 20:



* + Như trên, ta thấy khi K = 20 bức ảnh đã có chất lượng được cải thiện khá tốt. Nếu nhìn bằng mắt thường, màu sắc của bức ảnh khá là đầy đủ so với ảnh gốc nhưng kích thước cũng được giảm đáng kể (giảm 4.5 lần):



1. ***Kết luận***

* Với thuật toán K-means clustering đơn giản, ta có thể áp dụng vào các bài toán thực tế. Mặc dù kết quả chưa đạt cao, nhưng tiềm năng thuật toán đem lại vẫn là rất lớn nhờ sự đơn giản và nhanh chóng.
* Hiện tại có rất nhiều bài toán trong lĩnh vực Computer Vision sử dụng ví dụ như:
  + Bài toán phân loại đối tượng (Bag-of-word)
  + Bài toán phân vùng ảnh (image segmentation)
  + Bài toán theo bám đối tượng (object tracking)