Bài 4: K-means Clustering

Clustering (/tags#Clustering) Kmeans (/tags#Kmeans) Unsupervised-learning (/tags#Unsupervised-learning)

Jan 1, 2017

Trong trang này:

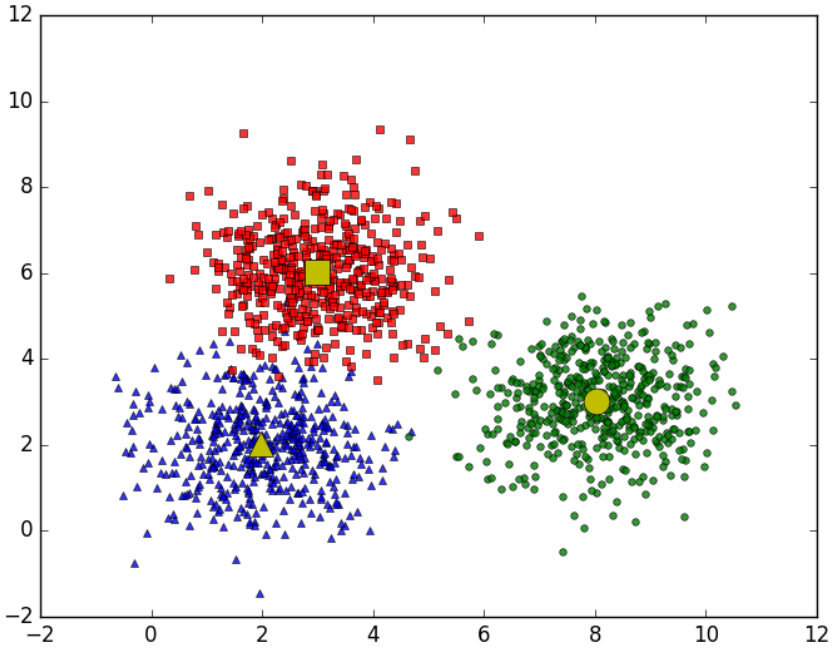
1. Giới thiệu

Trong bài trước (/2016/12/28/linearregression/), chúng ta đã làm quen với thuật toán Linear Regression - là thuật toán đơn giản nhất trong Supervised learning (/2016/12/27/categories/#supervised-learning-hoc-co-giam-sat). Bài này tôi sẽ giới thiệu một trong những thuật toán cơ bản nhất trong Unsupervised learning (/2016/12/27/categories/#unsupervised-learning-hoc-khong-giam-sat) - thuật toán K-means clustering (phân cụm K-means).

Trong thuật toán K-means clustering, chúng ta không biết nhãn (label) của từng điểm dữ liệu. Mục đích là làm thể 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 anh (chị) ta nghĩ đến sẽ là K-means Clustering. Vì nó là một trong những thuật toán đầu tiên mà anh ấy tìm được trong các cuốn sách, khóa học về Machine Learning. Và tôi cũng chắc rằng anh ấy đã đọc blog Machine Learning cơ bản (https://tiepvupsu.github.io). 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.

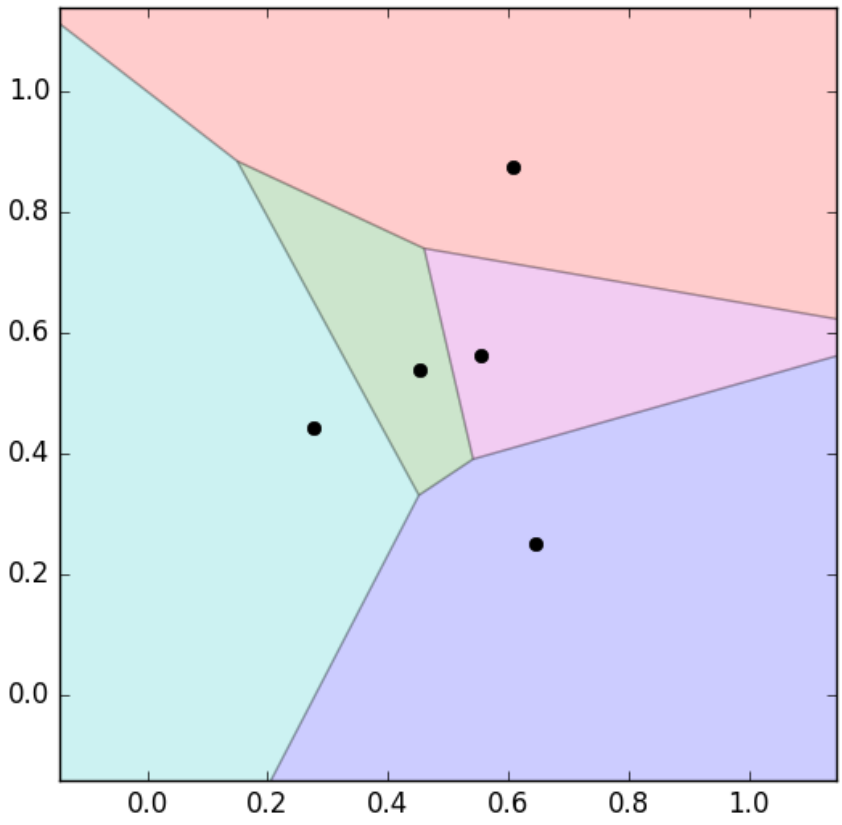
Ý tưởng đơn giản nhất về cluster (cụm) là tập hợp các điểm *ở gần nhau trong một không gian nào đó* (không gian này có thể có rất nhiều chiều trong trường hợp thông tin về một điểm dữ liệu là rất lớn). Hình bên dưới là một ví dụ về 3 cụm dữ liệu (từ giờ tôi sẽ viết gọn là *cluster*).



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, chúng ta 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:



Phân vùng lãnh hải của mỗi đảo. Các vùng khác nhau có màu sắc khác nhau.

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 (https://en.wikipedia.org/wiki/Voronoi_diagram).

Trong không gian ba chiều, lấy ví dụ là các hành tinh, thì (tạm gọi là) *lãnh không* của mỗi hành tinh sẽ là một đa diện. Trong không gian nhiều chiều hơn, chúng ta sẽ có những thứ (mà tôi gọi là) *siêu đa diện* (hyperpolygon).

Quay lại với bài toán phân nhóm và cụ thể là thuật toán K-means clustering, chúng ta cần một chút phân tích toán học trước khi đi tới phần tóm tắt thuật toán ở phần dưới. Nếu bạn không muốn đọc quá nhiều về toán, bạn có thể bỏ qua phần này. (*Tốt nhất là đừng bỏ qua, bạn sẽ tiếc đấy*).

2. Phân tích toán học

Mục đích cuối cùng của thuật toán phân nhóm này là: từ dữ liệu đầu vào và số lượng nhóm chúng ta muốn tìm, hãy chỉ ra center của mỗi nhóm và phân các điểm dữ liệu vào các nhóm tương ứng. Giả sử thêm rằng mỗi điểm dữ liệu chỉ thuộc vào đúng một nhóm.

Một số ký hiệu toán học

Giả sử có N điểm dữ liệu là $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N] \in \mathbb{R}^{d \times N}$ và $K < N$ là số cluster chúng ta muốn phân chia. Chúng ta cần tìm các center $\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_K \in \mathbb{R}^{d \times 1}$ và label của mỗi điểm dữ liệu.

Lưu ý về ký hiệu toán học: *trong các bài viết của tôi, các số vô hướng được biểu diễn bởi các chữ cái viết ở dạng không in đậm, có thể viết hoa, ví dụ x_1, N, y, k. Các vector được biểu diễn bằng các chữ cái thường in đậm, ví dụ $\mathbf{m}, \mathbf{x}_1$. Các ma trận được biểu diễn bởi các chữ viết hoa in đậm, ví dụ $\mathbf{X}, \mathbf{M}, \mathbf{Y}$. Lưu ý này đã được nêu ở bài Linear Regression (/2016/12/28/linearregression/). Tôi xin được không nhắc lại trong các bài tiếp theo.*

Với mỗi điểm dữ liệu $\mathbf{x}_i$ đặt $\mathbf{y}_i = [y_{i1}, y_{i2}, \ldots, y_{iK}]$ là label vector của nó, trong đó nếu $\mathbf{x}_i$ được phân vào cluster k thì $y_{ik} = 1$ và $y_{ij} = 0, \forall j \neq k$. Điều này có nghĩa là có đúng một phần tử của vector $\mathbf{y}_i$ là bằng 1 (tương ứng với cluster của $\mathbf{x}_i$), các phần tử còn lại bằng 0. Ví dụ: nếu một điểm dữ liệu có label vector là $[1, 0, 0, \ldots, 0]$ thì nó thuộc vào cluster 1, là $[0, 1, 0, \ldots, 0]$ thì nó thuộc vào cluster 2, ... Cách mã hóa label của dữ liệu như thế này được gọi là biểu diễn *one-hot* (https://en.wikipedia.org/wiki/One-hot). Chúng ta sẽ thấy cách biểu diễn one-hot này rất phổ biến trong Machine Learning ở các bài tiếp theo.

Ràng buộc của $\mathbf{y}_i$ có thể viết dưới dạng toán học như sau:

$$y_{ik} \in \{0, 1\}, \quad \sum_{k=1}^{K} y_{ik} = 1 \quad (1)$$

Hàm mất mát và bài toán tối ưu

Nếu ta coi center $\mathbf{m}_k$ là center (hoặc representative) của mỗi cluster và *ước lượng* tất cả các điểm được phân vào cluster này bởi $\mathbf{m}_k$, thì một điểm dữ liệu $\mathbf{x}_i$ được phân vào cluster k sẽ bị sai số là $(\mathbf{x}_i - \mathbf{m}_k)$. Chúng ta mong muốn sai số này có trị tuyệt đối nhỏ nhất nên (giống như trong bài Linear Regression (/2016/12/28/linearregression/#sai-so-du-doan)) ta sẽ tìm cách để đại lượng sau đây đạt giá trị nhỏ nhất:

$$\|\mathbf{x}_i - \mathbf{m}_k\|_2^2$$

Hơn nữa, vì $\mathbf{x}_i$ được phân vào cluster k nên $y_{ik} = 1, y_{ij} = 0, \ \forall j \neq k$. Khi đó, biểu thức bên trên sẽ được viết lại là:

$$y_{ik}\|\mathbf{x}_i - \mathbf{m}_k\|_2^2 = \sum_{j=1}^{K} y_{ij}\|\mathbf{x}_i - \mathbf{m}_j\|_2^2$$

(*Hy vọng chỗ này không quá khó hiểu*)

Sai số cho toàn bộ dữ liệu sẽ là:

$$\mathcal{L}(\mathbf{Y}, \mathbf{M}) = \sum_{i=1}^{N} \sum_{j=1}^{K} y_{ij} \|\mathbf{x}_i - \mathbf{m}_j\|_2^2$$

Trong đó $\mathbf{Y} = [\mathbf{y}_1; \mathbf{y}_2; \ldots; \mathbf{y}_N]$, $\mathbf{M} = [\mathbf{m}_1, \mathbf{m}_2, \ldots \mathbf{m}_K]$ lần lượt là các ma trận được tạo bởi label vector của mỗi điểm dữ liệu và center của mỗi cluster. Hàm số mất mát trong bài toán K-means clustering của chúng ta là hàm $\mathcal{L}(\mathbf{Y}, \mathbf{M})$ với ràng buộc như được nêu trong phương trình (1).

Tóm lại, chúng ta cần tối ưu bài toán sau:

$$\mathbf{Y}, \mathbf{M} = \arg\min_{\mathbf{Y}, \mathbf{M}} \sum_{i=1}^{N} \sum_{j=1}^{K} y_{ij} \|\mathbf{x}_i - \mathbf{m}_j\|_2^2 \quad (2)$$

$$\text{subject to: } y_{ij} \in \{0, 1\} \ \forall i, j; \quad \sum_{j=1}^{K} y_{ij} = 1 \ \forall i$$

(*subject to* nghĩa là *thỏa mãn điều kiện*).

Nhắc lại khái niệm $\arg\min$: Chúng ta biết ký hiệu $\min$ là *giá trị nhỏ nhất của hàm số*, $\arg\min$ chính là *giá trị của biến số để hàm số đó đạt giá trị nhỏ nhất đó*. Nếu $f(x) = x^2 - 2x + 1 = (x - 1)^2$ thì giá trị nhỏ nhất của hàm số này bằng 0, đạt được khi $x = 1$. Trong ví dụ này $\min_x f(x) = 0$ và $\arg\min_x f(x) = 1$. Thêm ví dụ khác, nếu $x_1 = 0, x_2 = 10, x_3 = 5$ thì ta nói $\arg\min_i x_i = 1$ vì 1 là chỉ số để x_i đạt giá trị nhỏ nhất (bằng 0). Biến số viết bên dưới $\min$ là biến số cúng ta cần tối ưu. Trong các bài toán tối ưu, ta thường quan tâm tới $\arg\min$ hơn là $\min$.

Thuật toán tối ưu hàm mất mát

Bài toán (2) là một bài toán khó tìm *điểm tối ưu* vì nó có thêm các điều kiện ràng buộc. *Bài toán này thuộc loại mix-integer programming (điều kiện biến là số nguyên) - là loại rất khó tìm nghiệm tối ưu toàn cục (global optimal point, tức nghiệm làm cho hàm mất mát đạt giá trị nhỏ nhất có thể).* Tuy nhiên, trong một số trường hợp chúng ta vẫn có thể tìm được phương pháp để tìm được nghiệm gần đúng hoặc điểm cực tiểu. (*Nếu chúng ta vẫn nhớ chương trình toán ôn thi đại học thì điểm cực tiểu chưa chắc đã phải là điểm làm cho hàm số đạt giá trị nhỏ nhất*).

Một cách đơn giản để giải bài toán (2) là xen kẽ giải $\mathbf{Y}$ và $\mathbf{M}$ khi biến còn lại được cố định. Đây là một thuật toán lặp, cũng là kỹ thuật phổ biến khi giải bài toán tối ưu. Chúng ta sẽ lần lượt giải quyết hai bài toán sau đây:

Cố định $\mathbf{M}$, tìm $\mathbf{Y}$

Giả sử đã tìm được các centers, hãy tìm các label vector để hàm mất mát đạt giá trị nhỏ nhất. Điều này tương đương với việc tìm cluster cho mỗi điểm dữ liệu.

Khi các centers là cố định, bài toán tìm label vector cho toàn bộ dữ liệu có thể được chia nhỏ thành bài toán tìm label vector cho từng điểm dữ liệu $\mathbf{x}_i$ như sau:

$$\mathbf{y}_i = \arg\min_{\mathbf{y}_i} \sum_{j=1}^{K} y_{ij} \|\mathbf{x}_i - \mathbf{m}_j\|_2^2 \quad (3)$$

$$\text{subject to: } y_{ij} \in \{0, 1\} \ \forall j; \quad \sum_{j=1}^{K} y_{ij} = 1$$

Vì chỉ có một phần tử của label vector $\mathbf{y}_i$ bằng 1 nên bài toán (3) có thể tiếp tục được viết dưới dạng đơn giản hơn:

$$j = \arg\min_{j} \|\mathbf{x}_i - \mathbf{m}_j\|_2^2$$

Vì $\|\mathbf{x}_i - \mathbf{m}_j\|_2^2$ chính là bình phương khoảng cách tính từ điểm $\mathbf{x}_i$ tới center $\mathbf{m}_j$, ta có thể kết luận rằng **mỗi điểm $\mathbf{x}_i$ thuộc vào cluster có center gần nó nhất**! Từ đó ta có thể dễ dàng suy ra label vector của từng điểm dữ liệu.

Cố định $\mathbf{Y}$, tìm $\mathbf{M}$

Giả sử đã tìm được cluster cho từng điểm, hãy tìm center mới cho mỗi cluster để hàm mất mát đạt giá trị nhỏ nhất.

Một 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:

$$\mathbf{m}_j = \arg\min_{\mathbf{m}_j} \sum_{i=1}^{N} y_{ij} \|\mathbf{x}_i - \mathbf{m}_j\|_2^2.$$

Tới đây, ta có thể tìm nghiệm bằng phương pháp giải đạo hàm bằng 0, vì hàm cần tối ưu là một hàm liên tục và có đạo hàm xác định tại mọi điểm. *Và quan trọng hơn, hàm này là hàm convex (lồi) theo $\mathbf{m}_j$ nên chúng ta sẽ tìm được giá trị nhỏ nhất và điểm tối ưu tương ứng. Sau này nếu có dịp, tôi sẽ nói thêm về tối ưu lồi (convex optimization) - một mảng cực kỳ quan trọng trong toán tối ưu*.

Đặt $l(\mathbf{m}_j)$ là hàm bên trong dấu $\arg\min$, ta có đạo hàm:

$$\frac{\partial l(\mathbf{m}_j)}{\partial \mathbf{m}_j} = 2 \sum_{i=1}^{N} y_{ij} (\mathbf{m}_j - \mathbf{x}_i)$$

Giải phương trình đạo hàm bằng 0 ta có:

$$\mathbf{m}_j \sum_{i=1}^{N} y_{ij} = \sum_{i=1}^{N} y_{ij} \mathbf{x}_i$$

$$\Rightarrow \mathbf{m}_j = \frac{\sum_{i=1}^{N} y_{ij} \mathbf{x}_i}{\sum_{i=1}^{N} y_{ij}}$$

Nếu để ý một chút, chúng ta sẽ thấy rằng mẫu số chính là phép đếm *số lượng các điểm dữ liệu* trong cluster j (*Bạn có nhận ra không?*). Còn tử số chính là *tổng các điểm dữ liệu* trong cluster j. (*Nếu bạn đọc vẫn nhớ điều kiện ràng buộc của các y_{ij} thì sẽ có thể nhanh chóng nhìn ra điều này*).

Hay nói một cách đơn giản hơn nhiều: **$\mathbf{m}_j$ là trung bình cộng của các điểm trong cluster j**.

Tên gọi *K-means clustering* cũng xuất phát từ đây.

Tóm tắt thuật toán

Tới đây tôi xin được tóm tắt lại thuật toán (*đặc biệt quan trọng với các bạn bỏ qua phần toán học bên trên*) như sau:

Đầu vào: Dữ liệu $\mathbf{X}$ và số lượng cluster cần tìm K.

Đầu ra: Các center $\mathbf{M}$ và label vector cho từng điểm dữ liệu $\mathbf{Y}$.

1. Chọn K điểm bất kỳ làm các center ban đầu.
2. Phân mỗi điểm dữ liệu vào cluster có center gần nó nhất.
3. Nếu việc gán dữ liệu vào từng cluster ở bước 2 không thay đổi so với vòng lặp trước nó thì ta dừng thuật toán.
4. Cập nhật center cho từng cluster bằng cách lấy trung bình cộng của tất các các điểm dữ liệu đã được gán vào cluster đó sau bước 2.
5. Quay lại bước 2.

Chúng ta có thể đảm bảo rằng thuật toán sẽ dừng lại sau một số hữu hạn vòng lặp. Thật vậy, vì hàm mất mát là một số dương và sau mỗi bước 2 hoặc 3, giá trị của hàm mất mát bị giảm đi. Theo kiến thức về dãy số trong chương trình cấp 3: *nếu một dãy số giảm và bị chặn dưới thì nó hội tụ!* Hơn nữa, số lượng cách phân nhóm cho toàn bộ dữ liệu là hữu hạn nên đến một lúc nào đó, hàm mất mát sẽ không thể thay đổi, và chúng ta có thể dừng thuật toán tại đây.

Chúng ta sẽ có một vài thảo luận về thuật toán này, về những hạn chế và một số phương pháp khắc phục. Nhưng trước hết, hãy xem nó thể hiện như thế nào trong một ví dụ cụ thể dưới đây.

3. Ví dụ trên Python

Giới thiệu bài toán

Để kiểm tra mức độ hiểu quả của một thuật toán, chúng ta sẽ làm một ví dụ đơn giản (thường được gọi là *toy example*). Trước hết, chúng ta chọn center cho từng cluster và tạo dữ liệu cho từng cluster bằng cách lấy mẫu theo phân phối chuẩn có kỳ vọng là center của cluster đó và ma trận hiệp phương sai (covariance matrix) là ma trận đơn vị.

Trước tiên, chúng ta cần khai báo các thư viện cần dùng. Chúng ta cần `numpy` và `matplotlib` như trong bài Linear Regression (/2016/12/28/linearregression/) cho việc tính toán ma trận và hiển thị dữ liệu. Chúng ta cũng cần thêm thư viện `scipy.spatial.distance` để tính khoảng cách giữa các cặp điểm trong hai tập hợp một cách hiệu quả.

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
np.random.seed(11)
```

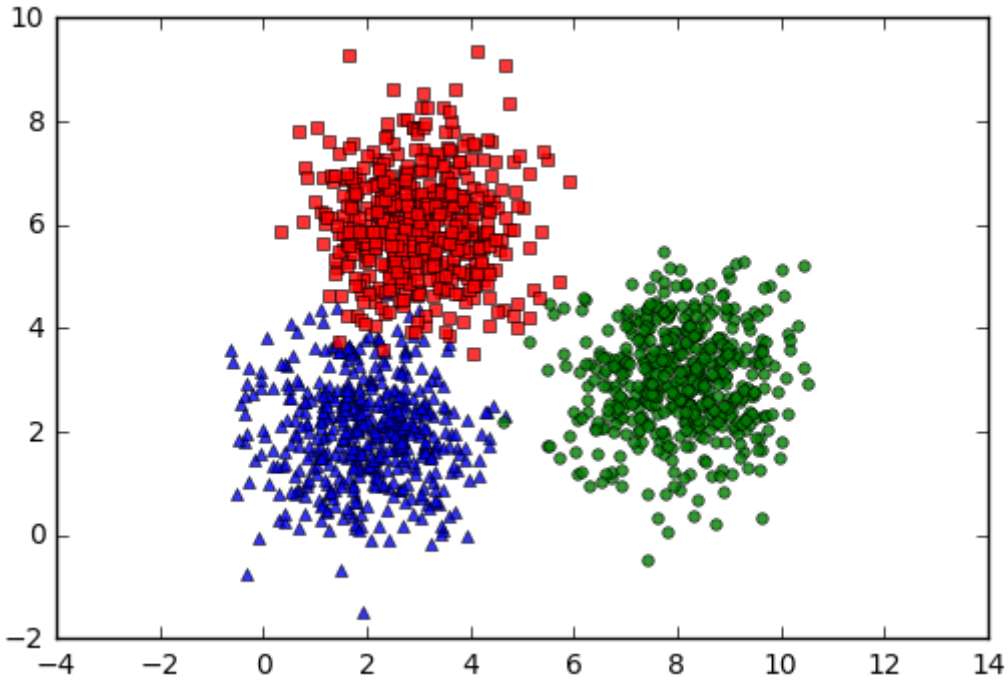
Tiếp theo, ta tạo dữ liệu bằng cách lấy các điểm theo phân phối chuẩn có kỳ vọng tại các điểm có tọa độ `(2, 2)`, `(8, 3)` và `(3, 6)`, ma trận hiệp phương sai giống nhau và là ma trận đơn vị. Mỗi cluster có 500 điểm. (*Chú ý rằng mỗi điểm dữ liệu là một hàng của ma trận dữ liệu.*

```
means = [[2, 2], [8, 3], [3, 6]]
cov = [[1, 0], [0, 1]]
N = 500
X0 = np.random.multivariate_normal(means[0], cov, N)
X1 = np.random.multivariate_normal(means[1], cov, N)
X2 = np.random.multivariate_normal(means[2], cov, N)

X = np.concatenate((X0, X1, X2), axis = 0)
K = 3

original_label = np.asarray([0]*N + [1]*N + [2]*N).T
```

Hiển thị dữ liệu trên đồ thị

Chúng ta cần một hàm `kmeans_display` để hiển thị dữ liệu. Sau đó hiển thị dữ liệu theo nhãn ban đầu.

```
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.axis('equal')
    plt.plot()
    plt.show()

kmeans_display(X, original_label)
```



Trong đồ thị trên, mỗi cluster tương ứng với một màu. Có thể nhận thấy rằng có một vài điểm màu đỏ bị lẫn sang phần cluster màu xanh.

Các hàm số cần thiết cho K-means clustering

Viết các hàm:

1. `kmeans_init_centers` để khởi tạo các centers ban đầu.
2. `kmeans_asign_labels` để gán nhán mới cho các điểm khi biết các centers.
3. `kmeans_update_centers` để cập nhật các centers mới dữa trên dữ liệu vừa được gán nhãn.
4. `has_converged` để kiểm tra điều kiện dừng của thuật toán.

```
def kmeans_init_centers(X, k):
    # randomly pick k rows of X as initial centers
    return X[np.random.choice(X.shape[0], k, replace=False)]

def kmeans_assign_labels(X, centers):
    # calculate pairwise distances btw data and centers
    D = cdist(X, centers)
    # return index of the closest center
    return np.argmin(D, axis = 1)

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]))
```

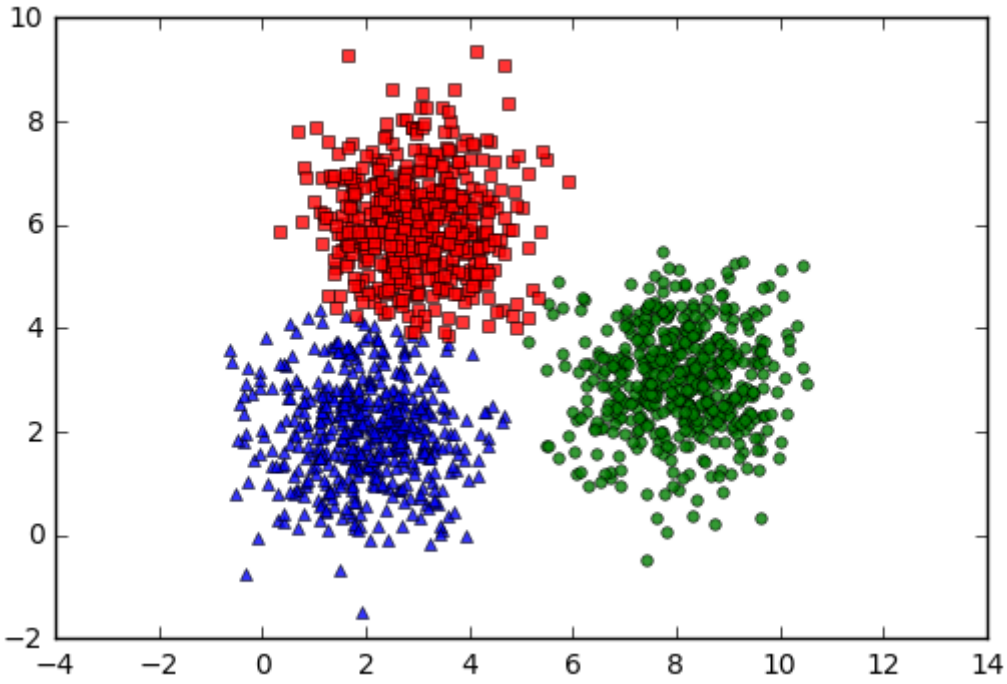
Phần chính của K-means clustering:

```
def kmeans(X, K):
    centers = [kmeans_init_centers(X, K)]
    labels = []
    it = 0
    while True:
        labels.append(kmeans_assign_labels(X, centers[-1]))
        new_centers = kmeans_update_centers(X, labels[-1], K)
        if has_converged(centers[-1], new_centers):
            break
        centers.append(new_centers)
        it += 1
    return (centers, labels, it)
```

Áp dụng thuật toán vừa viết vào dữ liệu ban đầu, hiển thị kết quả cuối cùng.

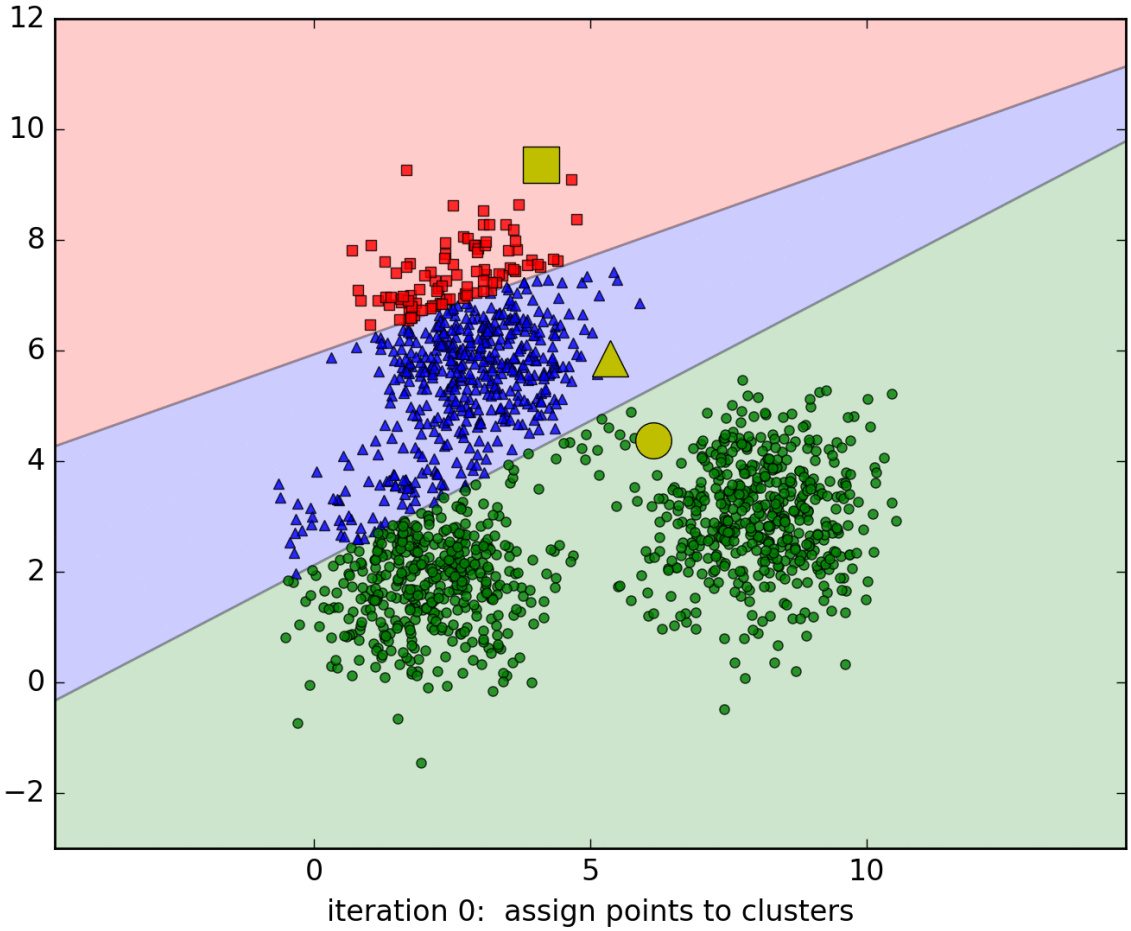
```
(centers, labels, it) = kmeans(X, K)
print('Centers found by our algorithm:')
print(centers[-1])

kmeans_display(X, labels[-1])
```

```
Centers found by our algorithm:
[[ 1.97563391  2.01568065]
 [ 8.03643517  3.02468432]
 [ 2.99084705  6.04196062]]
```



Từ kết quả này chúng ta thấy rằng thuật toán K-means clustering làm việc khá thành công, các centers tìm được khá gần với kỳ vọng ban đầu. Các điểm thuộc cùng một cluster hầu như được phân vào cùng một cluster (trừ một số điểm màu đỏ ban đầu đã bị phân nhầm vào cluster màu xanh da trời, nhưng tỉ lệ là nhỏ và có thể chấp nhận được).

Dưới đây là hình ảnh động minh họa thuật toán qua từng vòng lặp, chúng ta thấy rằng thuật toán trên hội tụ rất nhanh, chỉ cần 6 vòng lặp để có được kết quả cuối cùng:



Các bạn có thể xem thêm các trang web minh họa thuật toán K-means cluster tại:

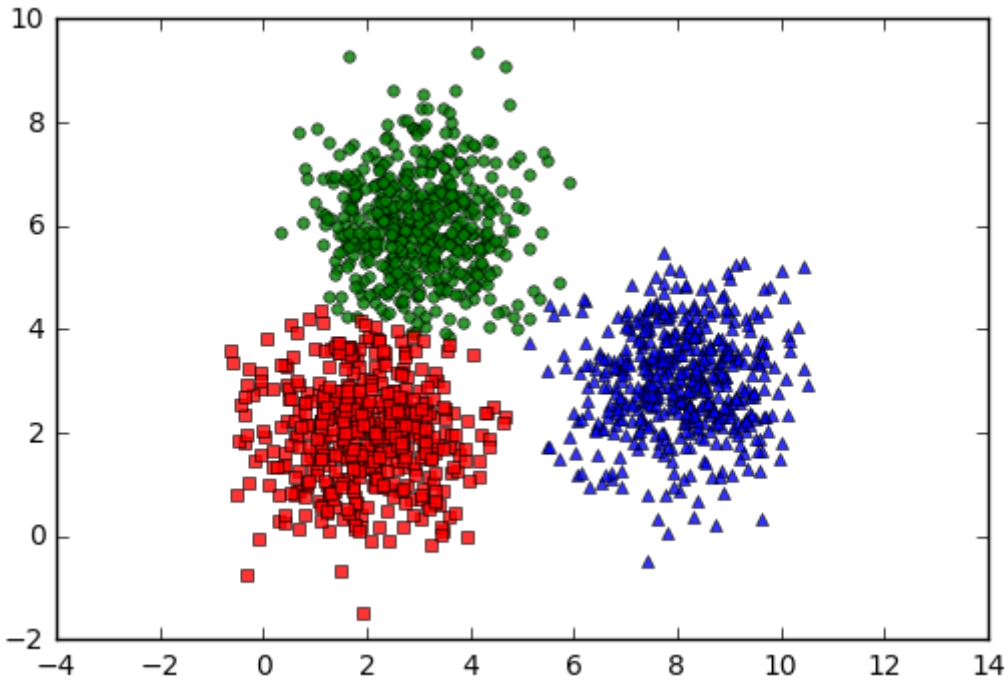
1. Visualizing K-Means Clustering (https://www.naftaliharris.com/blog/visualizing-k-means-clustering/)
2. Visualizing K-Means Clustering - Standford (http://stanford.edu/class/ee103/visualizations/kmeans/kmeans.html)

Kết quả tìm được bằng thư viện scikit-learn

Để kiểm tra thêm, chúng ta hãy so sánh kết quả trên với kết quả thu được bằng cách sử dụng thư viện `scikit-learn` (http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html).

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
print('Centers found by scikit-learn:')
print(kmeans.cluster_centers_)
pred_label = kmeans.predict(X)
kmeans_display(X, pred_label)
```

```
Centers found by scikit-learn:
[[ 8.0410628   3.02094748]
 [ 2.99357611  6.03605255]
 [ 1.97634981  2.01123694]]
```



Thật may mắn (*cho tôi*), hai thuật toán cho cùng một đáp số! Với cách thứ nhất, tôi mong muốn các bạn hiểu rõ được thuật toán K-means clustering làm việc như thế nào. Với cách thứ hai, tôi hy vọng các bạn biết áp dụng thư viện sẵn có như thế nào.

4. Thảo luận

Hạn chế

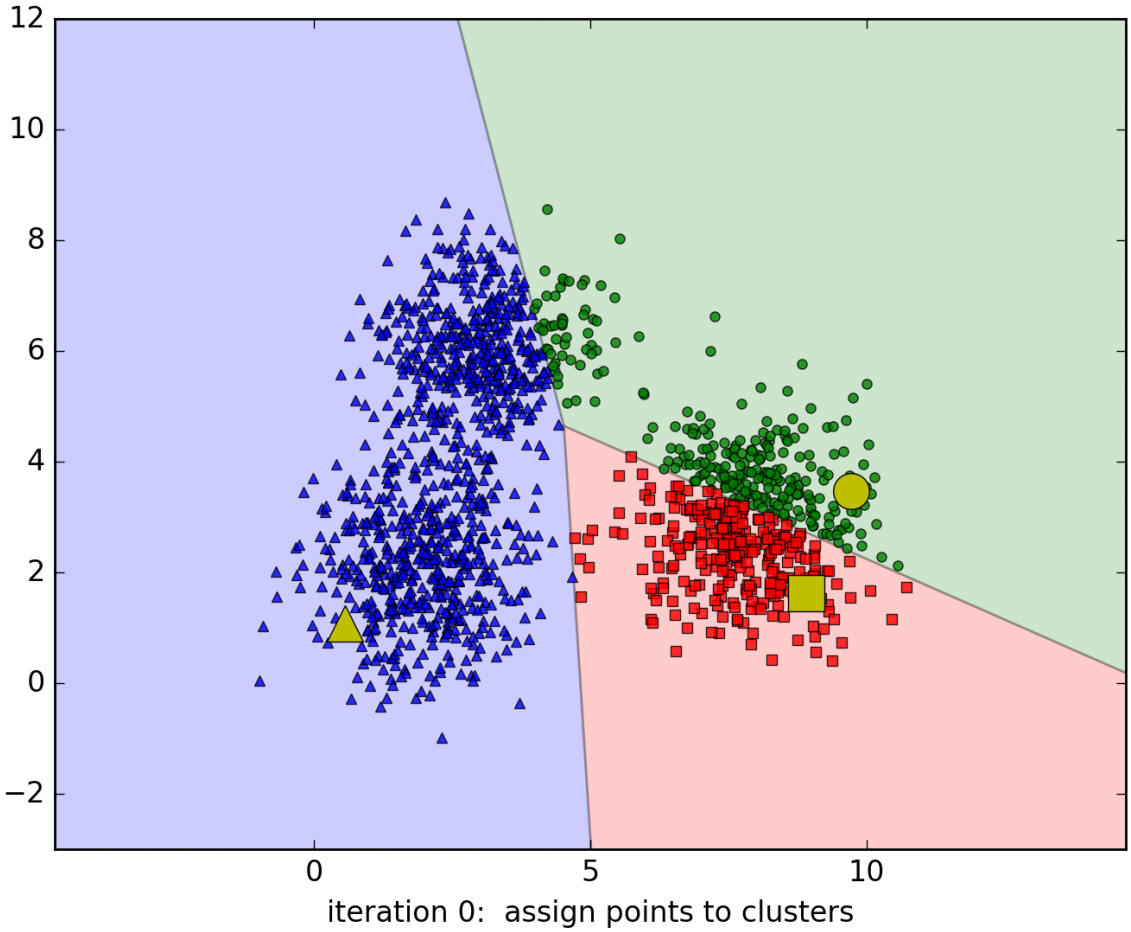
Có một vài hạn chế của thuật toán K-means clustering:

Chúng ta cần biết số lượng cluster cần clustering

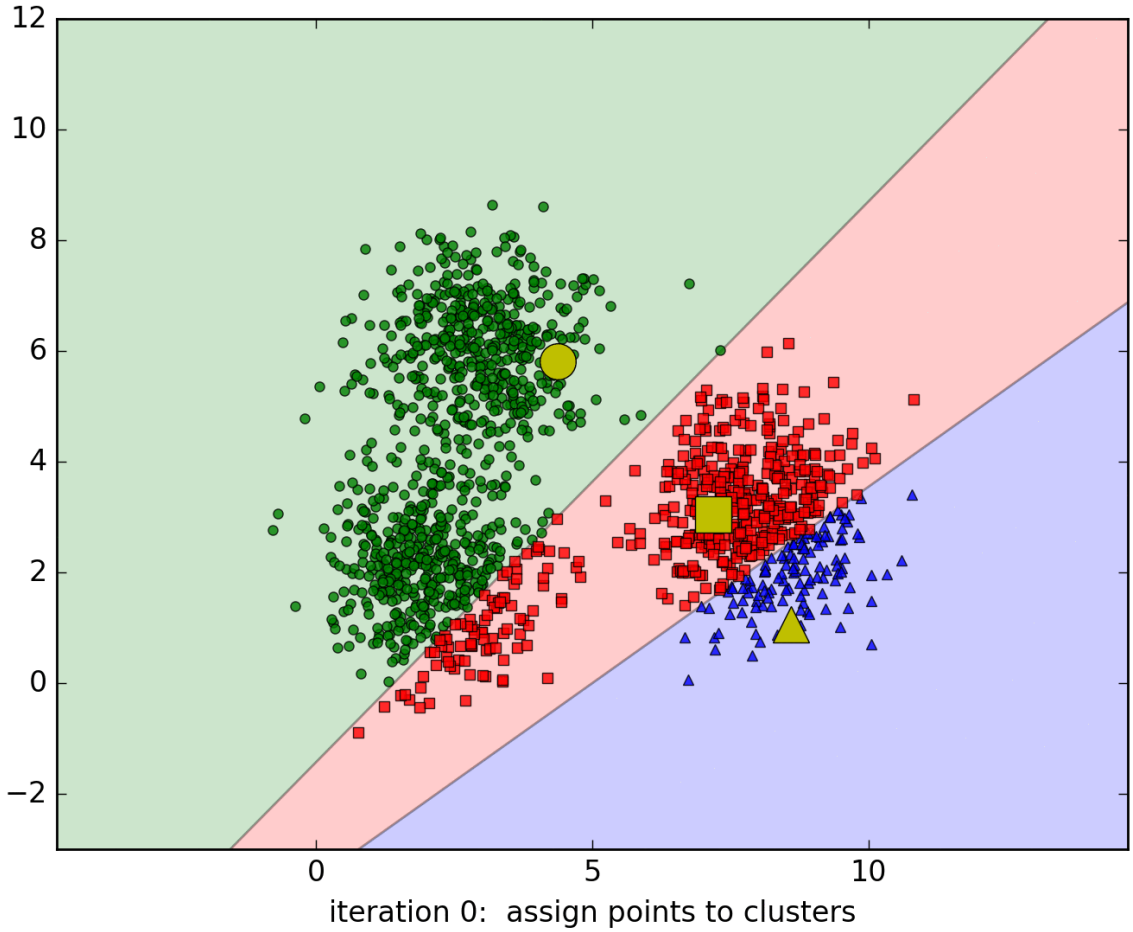
Để ý thấy rằng trong thuật toán nêu trên, chúng ta cần biết đại lượng K là số lượng clusters. Trong thực tế, nhiều trường hợp chúng ta không xác định được giá trị này. Có một số phương pháp giúp xác định số lượng clusters, tôi sẽ dành thời gian nói về chúng sau nếu có dịp. Bạn đọc có thể tham khảo Elbow method - Determining the number of clusters in a data set (https://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set).

Nghiệm cuối cùng phụ thuộc vào các centers được khởi tạo ban đầu

Tùy vào các center ban đầu mà thuật toán có thể có tốc độ hội tụ rất chậm, ví dụ:



hoặc thậm chí cho chúng ta nghiệm không chính xác (chỉ là local minimum - điểm cực tiểu - mà không phải giá trị nhỏ nhất):

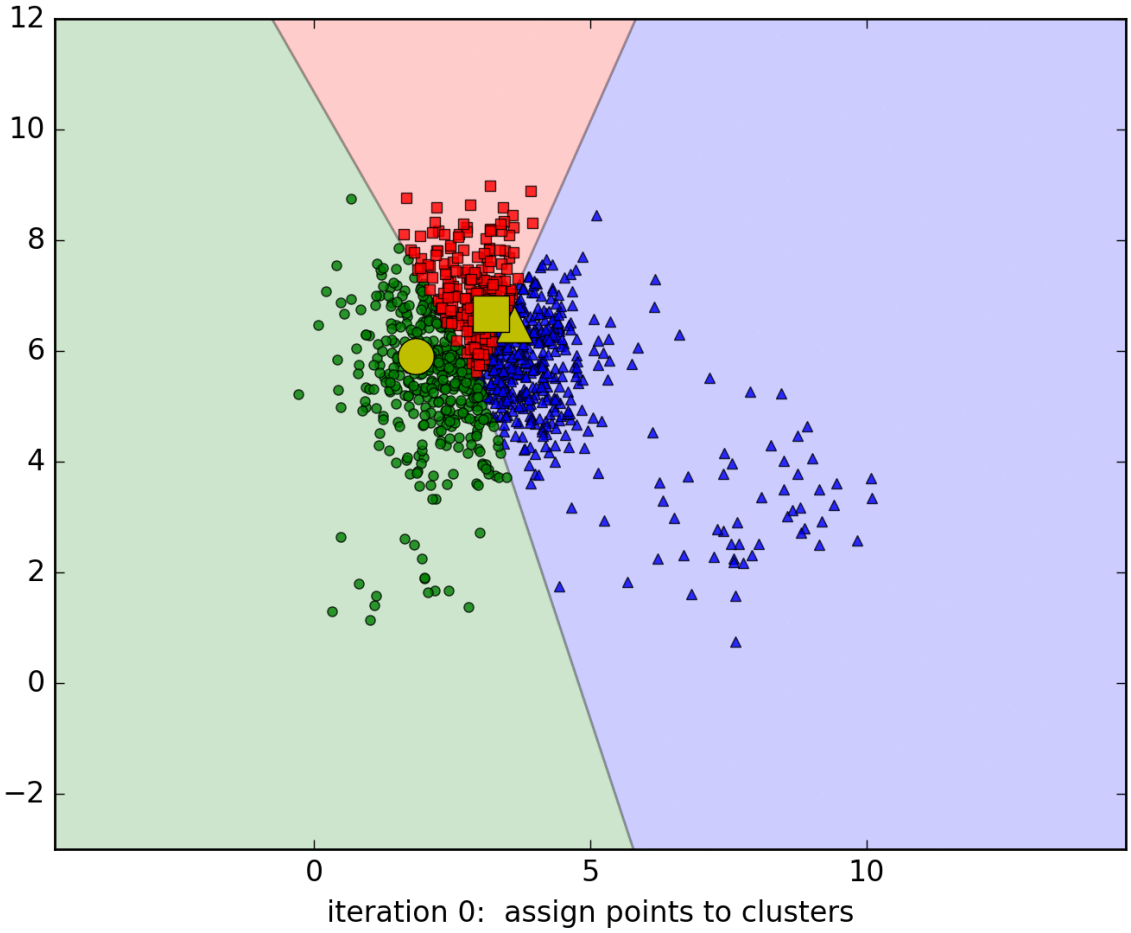


Có một vài cách khắc phục đó là:

- Chạy K-means clustering nhiều lần với các center ban đầu khác nhau rồi chọn cách có hàm mất mát cuối cùng đạt giá trị nhỏ nhất.
- K-means++ -Improve initialization algorithm - wiki (https://en.wikipedia.org/wiki/K-means%2B%2B#Improved_initialization_algorithm).
- Bạn nào muốn tìm hiểu sâu hơn có thể xem bài báo khoa học Cluster center initialization algorithm for K-means clustering (http://www.sciencedirect.com/science/article/pii/S0167865504000996).

Các cluster cần có só lượng điểm gần bằng nhau

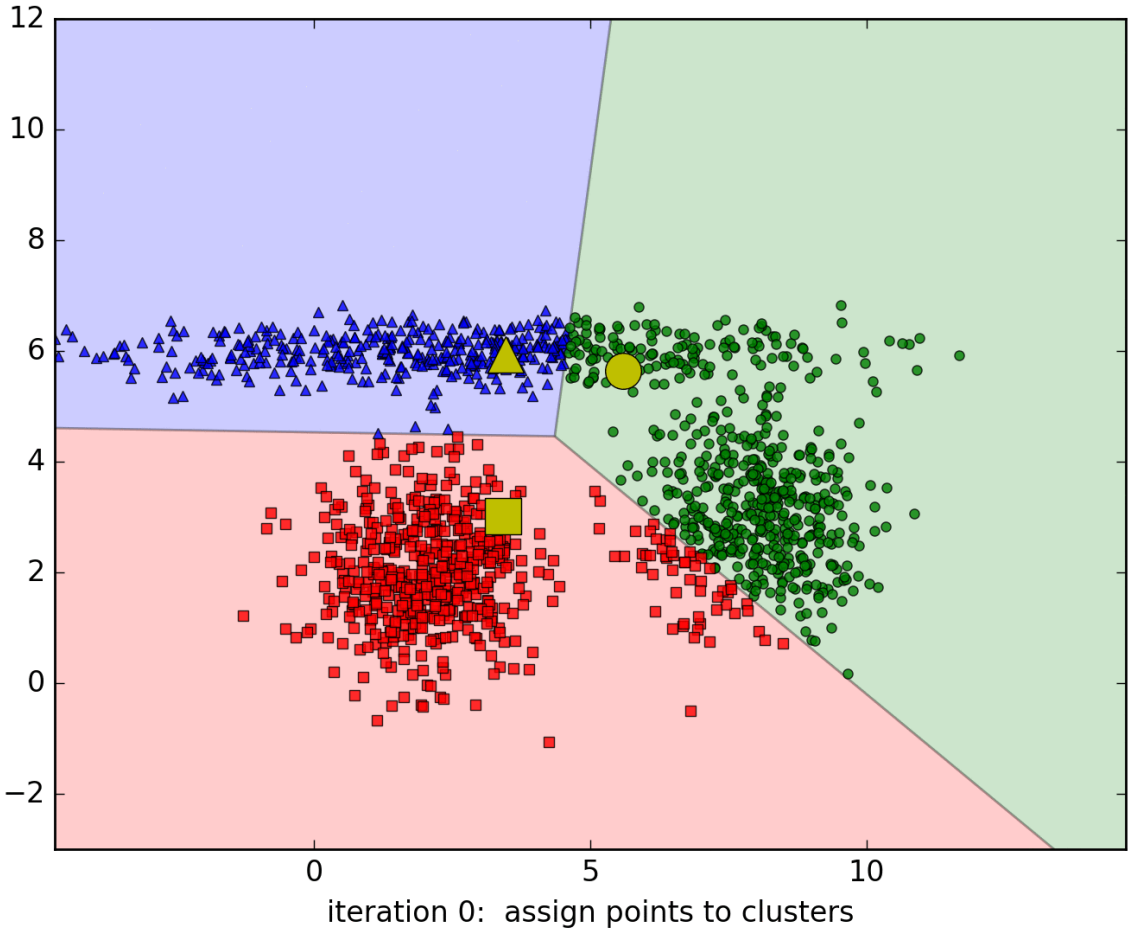
Dưới đây là một ví dụ với 3 cluster với 20, 50, và 1000 điểm. Kết quả cuối cùng không chính xác.



Các cluster cần có dạng hình tròn

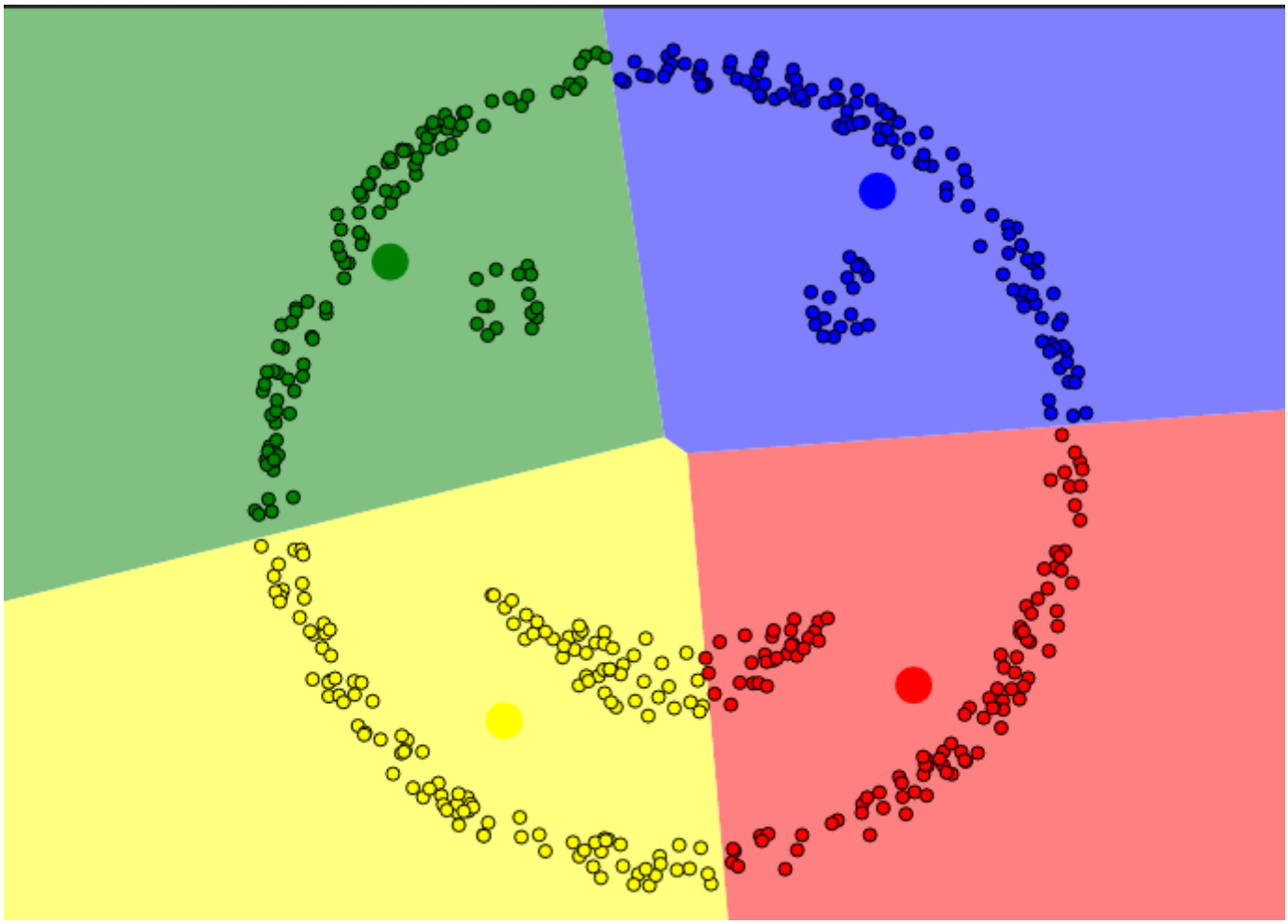
Tức các cluster tuân theo phân phối chuẩn và ma trận hiệp phương sai là ma trận đường chéo có các điểm trên đường chéo giống nhau.

Dưới đây là 1 ví dụ khi 1 cluster có dạng hình dẹt.



Khi một cluster nằm phía trong 1 cluster khác

Đây là ví dụ kinh điển về việc K-means clustering không thể phân cụm dữ liệu. Một cách tự nhiên, chúng ta sẽ phân ra thành 4 cụm: mắt trái, mắt phải, miệng, xung quanh mặt. Nhưng vì mắt và miệng nằm trong khuôn mặt nên K-means clustering không thực hiện được:



Mặc dù có những hạn chế, K-means clustering vẫn cực kỳ quan trọng trong Machine Learning và là nền tảng cho nhiều thuật toán phức tạp khác sau này. Chúng ta cần bắt đầu từ những thứ đơn giản. *Simple is best!*

5. Tài liệu tham khảo

Clustering documents using k-means (http://scikit-learn.org/stable/auto_examples/text/document_clustering.html)

Voronoi Diagram - Wikipedia (https://en.wikipedia.org/wiki/Voronoi_diagram)

Cluster center initialization algorithm for K-means clustering (http://www.sciencedirect.com/science/article/pii/S0167865504000996)

Visualizing K-Means Clustering (https://www.naftaliharris.com/blog/visualizing-k-means-clustering/)

Visualizing K-Means Clustering - Standford (http://stanford.edu/class/ee103/visualizations/kmeans/kmeans.html)

Nếu có câu hỏi, Bạn có thể để lại comment bên dưới hoặc trên Forum (https://www.facebook.com/groups/257768141347267/) để nhận được câu trả lời sớm hơn.
Bạn đọc có thể ủng hộ blog qua 'Buy me a cofee' (/buymeacoffee/) ở góc trên bên trái của blog.