**Ba thuật toán cơ bản trong phân loại hình ảnh mch tại thời điểm hiện tại không dùng được nữa**

**Chỉ đọc để hiểu vì sai số đưa ra quá lớn khó optimize**

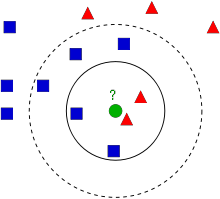
**Hiểu k-gần Neighbor**

**mục tiêu**

Trong chương này, chúng ta sẽ hiểu các khái niệm về thuật toán k-Recent Neighbor (kNN).

**Lý thuyết**

kNN là một trong những thuật toán phân loại đơn giản nhất có sẵn cho việc học có giám sát. Ý tưởng là tìm kiếm sự trùng khớp gần nhất của dữ liệu thử nghiệm trong không gian tính năng. Chúng tôi sẽ xem xét nó với hình ảnh dưới đây.



Trong ảnh, có hai gia đình, *Blue Squares và Red Tam giác* . Chúng tôi gọi mỗi gia đình là **Lớp học** . Nhà của họ được hiển thị trong bản đồ thị trấn của họ mà chúng tôi gọi là *không gian đặc trưng* . *(Bạn có thể coi không gian tính năng là không gian nơi tất cả dữ liệu được chiếu. Ví dụ: hãy xem xét không gian tọa độ 2D. Mỗi dữ liệu có hai tính năng, tọa độ x và y. Bạn có thể biểu thị dữ liệu này trong không gian tọa độ 2D của mình, phải không? Hãy tưởng tượng nếu có ba tính năng, bạn cần không gian 3D. Bây giờ hãy xem xét các tính năng N, nơi bạn cần không gian N chiều, phải không? Không gian N chiều này là không gian tính năng của nó. Trong hình ảnh của chúng tôi, bạn có thể coi đó là trường hợp 2D với hai tính năng)* .

Bây giờ một thành viên mới đi vào thị trấn và tạo ra một ngôi nhà mới, được hiển thị dưới dạng vòng tròn màu xanh lá cây. Anh ta nên được thêm vào một trong những gia đình Xanh / Đỏ này. Chúng tôi gọi quá trình đó, **Phân loại** . Chúng ta làm gì? Vì chúng ta đang làm việc với kNN, chúng ta hãy áp dụng thuật toán này.

Một phương pháp là kiểm tra ai là hàng xóm gần nhất của anh ta. Từ hình ảnh, rõ ràng đó là gia đình Tam giác đỏ. Vì vậy, anh cũng được thêm vào Tam giác đỏ. Phương pháp này được gọi đơn giản là **Hàng xóm gần nhất** , bởi vì phân loại chỉ phụ thuộc vào hàng xóm gần nhất.

Nhưng có một vấn đề với điều đó. Tam giác đỏ có thể là gần nhất. Nhưng nếu có nhiều Blue Squares ở gần anh ta thì sao? Sau đó, Blue Squares có nhiều sức mạnh ở địa phương đó hơn Tam giác đỏ. Vì vậy, chỉ cần kiểm tra gần nhất là không đủ. Thay vào đó chúng tôi kiểm tra một số gia đình *k* gần nhất. Sau đó, bất cứ ai chiếm đa số trong họ, anh chàng mới thuộc về gia đình đó. Trong hình ảnh của chúng tôi, hãy lấy *k = 3* , tức là 3 gia đình gần nhất. Anh ta có hai màu Đỏ và một Màu xanh (có hai màu tương đương với Blues, nhưng vì k = 3, chúng tôi chỉ lấy một trong số chúng), vì vậy một lần nữa anh ta nên được thêm vào gia đình Đỏ. Nhưng nếu chúng ta lấy *k = 7* thì sao? Sau đó, anh ta có 5 gia đình Xanh và 2 gia đình Đỏ. Tuyệt quá!! Bây giờ anh ta nên được thêm vào gia đình Blue. Vì vậy, tất cả thay đổi với giá trị của k. Điều thú vị hơn là, nếu *k = 4*? Ông có 2 người hàng xóm Đỏ và 2 Xanh. Đó là một cái cà vạt !!! Vì vậy, tốt hơn nên lấy k là một số lẻ. Vì vậy, phương pháp này được gọi là **Hàng xóm gần nhất** vì phân loại phụ thuộc vào k hàng xóm gần nhất.

Một lần nữa, trong kNN, đúng là chúng ta đang xem xét k hàng xóm, nhưng chúng ta đang cho tầm quan trọng như nhau đối với tất cả, phải không? Có công bằng không? Ví dụ: lấy trường hợp *k = 4* . Chúng tôi bảo đó là cà vạt. Nhưng hãy xem, 2 gia đình Đỏ gần gũi với anh hơn 2 gia đình Xanh kia. Vì vậy, anh ta có đủ điều kiện để được thêm vào Red. Vậy làm thế nào để chúng ta giải thích một cách toán học? Chúng tôi cung cấp một số trọng lượng cho mỗi gia đình tùy thuộc vào khoảng cách của họ với người mới đến. Đối với những người ở gần anh ta có trọng lượng cao hơn trong khi những người ở xa có trọng lượng thấp hơn. Sau đó, chúng tôi thêm tổng trọng lượng của từng gia đình riêng biệt. Bất cứ ai có tổng trọng lượng cao nhất, người mới đến gia đình đó. Điều này được gọi là **kNN sửa đổi** .

Vì vậy, một số điều quan trọng bạn nhìn thấy ở đây là gì?

* Bạn cần phải có thông tin về tất cả các ngôi nhà trong thị trấn, phải không? Bởi vì, chúng tôi phải kiểm tra khoảng cách từ người mới đến tất cả các ngôi nhà hiện có để tìm người hàng xóm gần nhất. Nếu có nhiều nhà và gia đình, sẽ mất rất nhiều bộ nhớ và cũng cần nhiều thời gian hơn để tính toán.
* Gần như không có thời gian cho bất kỳ loại đào tạo hoặc chuẩn bị.

Bây giờ hãy xem nó trong OpenCV.

**KNN trong OpenCV**

Chúng tôi sẽ làm một ví dụ đơn giản ở đây, với hai gia đình (lớp học), giống như ở trên. Sau đó, trong chương tiếp theo, chúng ta sẽ làm ví dụ tốt hơn nhiều.

Vì vậy, ở đây, chúng tôi gắn nhãn họ Đỏ là **Class-0** (được ký hiệu là 0) và Blue họ là **Class-1** (ký hiệu là 1). Chúng tôi tạo 25 gia đình hoặc 25 dữ liệu đào tạo và gắn nhãn cho họ là Class-0 hoặc Class-1. Chúng tôi làm tất cả những điều này với sự trợ giúp của Trình tạo số ngẫu nhiên trong Numpy.

Sau đó, chúng tôi vẽ nó với sự giúp đỡ của Matplotlib. Các gia đình màu đỏ được hiển thị dưới dạng hình tam giác màu đỏ và gia đình màu xanh được hiển thị dưới dạng hình vuông màu xanh.

nhập cv2

nhập numpy như np

nhập matplotlib.pyplot dưới dạng plt

# Bộ tính năng chứa các giá trị (x, y) của 25 dữ liệu đã biết / đào tạo

trainData = np . ngẫu nhiên . randint ( 0 , 100 , ( 25 , 2 )) . astype ( np . float32 )

# Nhãn mỗi màu Đỏ hoặc Xanh lam với các số 0 và 1

phản hồi = np . ngẫu nhiên . randint ( 0 , 2 , ( 25 , 1 )) . astype ( np . float32 )

# Lấy gia đình Đỏ và vẽ chúng

màu đỏ = trainData [ phản hồi . ravel () == 0 ]

plt . phân tán ( đỏ [:, 0 ], đỏ [:, 1 ], 80 , 'r' , '^' )

# Lấy gia đình Blue và vẽ chúng

màu xanh = trainData [ phản hồi . ravel () == 1 ]

plt . phân tán ( màu xanh [:, 0 ], màu xanh [:, 1 ], 80 , 'b' , 's' )

plt . hiển thị ()

Bạn sẽ nhận được một cái gì đó tương tự như hình ảnh đầu tiên của chúng tôi. Vì bạn đang sử dụng trình tạo số ngẫu nhiên, bạn sẽ nhận được dữ liệu khác nhau mỗi lần bạn chạy mã.

Tiếp theo khởi tạo thuật toán *kNN* và *truyền trainData* và *trả lời* để huấn luyện *kNN* (Nó xây dựng một cây tìm kiếm).

Sau đó, chúng tôi sẽ đưa một người mới đến và phân loại anh ta đến một gia đình với sự giúp đỡ của kNN trong OpenCV. Trước khi đến kNN, chúng tôi cần biết một vài điều về dữ liệu thử nghiệm của chúng tôi (dữ liệu của những người mới đến). Dữ liệu của chúng tôi phải là một mảng dấu phẩy động với kích thước con số \;  của \;  testdata \ lần số \;  của \;  Tính năng, đặc điểm. Sau đó, chúng tôi tìm thấy những người hàng xóm gần nhất của người mới đến. Chúng tôi có thể chỉ định có bao nhiêu hàng xóm chúng tôi muốn. Nó trở lại:

1. Nhãn được trao cho người mới đến tùy thuộc vào lý thuyết kNN mà chúng ta đã thấy trước đó. Nếu bạn muốn thuật toán Vùng lân cận gần nhất, chỉ cần xác định *k = 1* trong đó k là số lượng lân cận.
2. Các nhãn của hàng xóm k-gần nhất.
3. Khoảng cách tương ứng từ người mới đến mỗi người hàng xóm gần nhất.

Vì vậy, hãy xem cách nó hoạt động. Comer mới được đánh dấu màu xanh lá cây.

người mới đến = np . ngẫu nhiên . randint ( 0 , 100 , ( 1 , 2 )) . astype ( np . float32 )

plt . phân tán ( người mới [:, 0 ], người mới [:, 1 ], 80 , 'g' , 'o' )

knn = cv2 . KNearest ()

knn . train ( trainData , hồi đáp )

ret , results , xóm , dist = knn . find\_nearest ( người mới , 3 )

in "kết quả:" , kết quả , " \ n "

in "hàng xóm:" , hàng xóm , " \ n "

in "khoảng cách:" , dist

plt . hiển thị ()

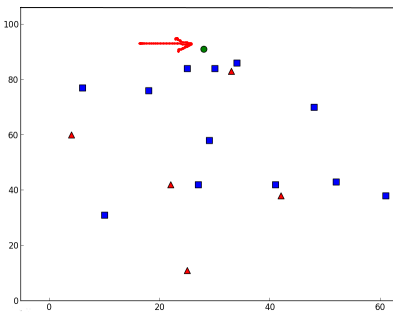
Tôi nhận được kết quả như sau:

Kết quả : [[ 1. ]]

hàng xóm : [[ 1. 1. 1. ]]

khoảng cách : [[ 53. 58. 61. ]]

Nó nói người mới của chúng tôi có 3 người hàng xóm, tất cả đều từ gia đình Blue. Do đó, anh được gắn mác là gia đình Blue. Rõ ràng từ cốt truyện dưới đây:



Nếu bạn có số lượng dữ liệu lớn, bạn có thể chuyển nó dưới dạng mảng. Kết quả tương ứng cũng thu được dưới dạng mảng.

# 10 người mới đến người

mới = np . ngẫu nhiên . randint ( 0 , 100 , ( 10 , 2 )) . astype ( np . float32 )

ret , kết quả , hàng xóm , dist = knn . find\_nearest ( người mới , 3 )

# Kết quả cũng sẽ chứa 10 nhãn.

# Hiểu SVM

## mục tiêu

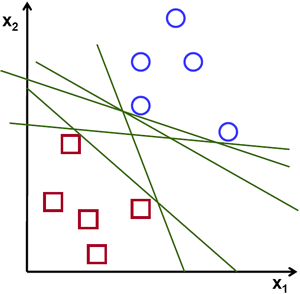
**Trong chương này**

* Chúng ta sẽ thấy một sự hiểu biết trực quan về SVM

## Lý thuyết

### Tuyến tính tách dữ liệu

Hãy xem xét hình ảnh dưới đây có hai loại dữ liệu, đỏ và xanh. Trong kNN, đối với dữ liệu thử nghiệm, chúng tôi đã sử dụng để đo khoảng cách của nó với tất cả các mẫu đào tạo và lấy mẫu có khoảng cách tối thiểu. Phải mất nhiều thời gian để đo tất cả các khoảng cách và nhiều bộ nhớ để lưu trữ tất cả các mẫu đào tạo. Nhưng xem xét dữ liệu được đưa ra trong hình ảnh, chúng ta có cần nhiều đến vậy không?



Hãy xem xét một ý tưởng khác. Chúng tôi tìm thấy một dòng, f (x) = ax_1 + bx_2 + cphân chia cả hai dữ liệu cho hai khu vực. Khi chúng tôi nhận được một test\_data mới X, chỉ cần thay thế nó trong f (x). Nếu f (X)> 0, nó thuộc nhóm màu xanh, khác với nhóm màu đỏ. Chúng ta có thể gọi dòng này là **Ranh giới quyết định** . Nó rất đơn giản và hiệu quả bộ nhớ. Những dữ liệu như vậy có thể được chia thành hai với một đường thẳng (hoặc siêu phẳng ở kích thước cao hơn) được gọi là Phân **tách tuyến tính** .

Vì vậy, trong hình ảnh trên, bạn có thể thấy rất nhiều dòng như vậy là có thể. Chúng ta sẽ lấy cái nào? Rất trực giác chúng ta có thể nói rằng dòng nên được đi càng xa càng tốt từ tất cả các điểm. Tại sao? Bởi vì có thể có tiếng ồn trong dữ liệu đến. Dữ liệu này sẽ không ảnh hưởng đến độ chính xác phân loại. Vì vậy, đi một dòng xa nhất sẽ cung cấp miễn dịch hơn chống lại tiếng ồn. Vì vậy, những gì SVM làm là tìm một đường thẳng (hoặc siêu phẳng) với khoảng cách tối thiểu lớn nhất đến các mẫu đào tạo. Xem dòng in đậm trong hình ảnh bên dưới đi qua trung tâm.



Vì vậy, để tìm ra Ranh giới quyết định này, bạn cần dữ liệu đào tạo. Bạn có cần tất cả không? KHÔNG. Chỉ cần những người gần với nhóm đối diện là đủ. Trong hình ảnh của chúng tôi, chúng là một hình tròn đầy màu xanh và hai hình vuông đầy màu đỏ. Chúng ta có thể gọi chúng là **Vectơ hỗ trợ** và các dòng đi qua chúng được gọi là **Mặt phẳng hỗ trợ** . Họ là đủ để tìm ranh giới quyết định của chúng tôi. Chúng tôi không cần phải lo lắng về tất cả các dữ liệu. Nó giúp giảm dữ liệu.

Điều gì đã xảy ra là, hai siêu máy bay đầu tiên được tìm thấy đại diện tốt nhất cho dữ liệu. Đối với ví dụ, dữ liệu xanh được đại diện bởi w ^ Tx + b_0> 1trong khi dữ liệu được thể hiện bằng màu đỏ w ^ Tx + b_0 <-1ở đâu wlà **vector trọng số** ( w = [w_1, w_2, ..., w_n]) và xlà vector đặc trưng ( x = [x_1, x_2, ..., x_n]). b_0là sự **thiên vị** . Trọng lượng vector quyết định hướng của ranh giới quyết định trong khi điểm thiên vị quyết định vị trí của nó. Bây giờ ranh giới quyết định được xác định là ở giữa các siêu phẳng, do đó được thể hiện là w ^ Tx + b_0 = 0. Khoảng cách tối thiểu từ vectơ hỗ trợ đến ranh giới quyết định được đưa ra bởi , distance_ {hỗ trợ \, vectơ} = \ frac {1} {|| w ||}. Số dư là gấp đôi khoảng cách này và chúng ta cần tối đa hóa mức ký quỹ này. tức là chúng ta cần giảm thiểu một hàm mới L (w, b_0)với một số ràng buộc có thể được trình bày dưới đây:

\ min_ {w, b_0} L (w, b_0) = \ frac {1} {2} || w | | ^ 2 \;  \ văn bản {chủ đề} \;  t_i (w ^ Tx + b_0) \ geq 1 \;  tôi sẽ không

nơi t_ilà nhãn của từng loại, t_i \ trong [-1,1].

### Phi tuyến tính tách dữ liệu

Hãy xem xét một số dữ liệu không thể chia thành hai với một đường thẳng. Ví dụ: hãy xem xét dữ liệu một chiều trong đó 'X' ở -3 & +3 và 'O' ở -1 & +1. Rõ ràng nó không thể phân tách tuyến tính. Nhưng có những phương pháp để giải quyết những vấn đề này. Nếu chúng ta có thể ánh xạ tập dữ liệu này bằng một hàm, f (x) = x ^ 2chúng ta sẽ nhận được 'X' tại 9 và 'O' tại 1 có thể phân tách tuyến tính.

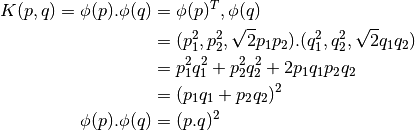
Nếu không, chúng ta có thể chuyển đổi dữ liệu một chiều thành hai chiều này. Chúng ta có thể sử dụng f (x) = (x, x ^ 2)chức năng để ánh xạ dữ liệu này. Sau đó, 'X' trở thành (-3,9) và (3,9) trong khi 'O' trở thành (-1,1) và (1,1). Đây cũng là phân tách tuyến tính. Nói tóm lại, cơ hội nhiều hơn cho một dữ liệu có thể phân tách phi tuyến tính trong không gian chiều thấp hơn để trở thành phân tách tuyến tính trong không gian chiều cao hơn.

Nói chung, có thể ánh xạ các điểm trong không gian chiều sang một không gian D chiều (D> d)để kiểm tra khả năng phân tách tuyến tính. Có một ý tưởng giúp tính toán sản phẩm chấm trong không gian chiều cao (nhân) bằng cách thực hiện các tính toán trong không gian đầu vào (tính năng) chiều thấp. Chúng ta có thể minh họa bằng ví dụ sau.

Xem xét hai điểm trong không gian hai chiều, p = (p_1, p_2)và q = (q_1, q_2). Hãy \ philà một hàm ánh xạ ánh xạ một điểm hai chiều vào không gian ba chiều như sau:

\ phi (p) = (p_ {1} ^ 2, p_ {2} ^ 2, \ sqrt {2} p_1 p_2) \ phi (q) = (q_ {1} ^ 2, q_ {2} ^ 2, \ sqrt {2} q_1 q_2)

Hãy để chúng tôi xác định một hàm kernel K (p, q), một sản phẩm chấm giữa hai điểm, được hiển thị bên dưới:

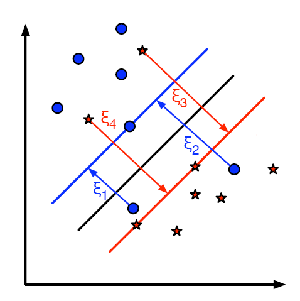


Điều đó có nghĩa là, một sản phẩm chấm trong không gian ba chiều có thể đạt được bằng cách sử dụng sản phẩm chấm bình phương trong không gian hai chiều. Điều này có thể được áp dụng cho không gian chiều cao hơn. Vì vậy, chúng ta có thể tính toán các tính năng chiều cao hơn từ chính kích thước thấp hơn. Khi chúng tôi ánh xạ chúng, chúng tôi có được một không gian chiều cao hơn.

Ngoài tất cả các khái niệm này, còn có vấn đề phân loại sai. Vì vậy, chỉ cần tìm ranh giới quyết định với biên độ tối đa là không đủ. Chúng ta cũng cần xem xét vấn đề lỗi phân loại sai. Đôi khi, có thể tìm thấy một ranh giới quyết định với biên độ ít hơn, nhưng với việc phân loại sai. Dù sao, chúng ta cần sửa đổi mô hình của mình sao cho nó sẽ tìm thấy ranh giới quyết định với biên độ tối đa, nhưng với ít phân loại sai hơn. Các tiêu chí tối thiểu hóa được sửa đổi như:

tối thiểu \;  | | w | | ^ 2 + C (khoảng cách \; của \; phân loại sai \; mẫu \; đến \; của họ \; đúng \; vùng)

Dưới đây hình ảnh cho thấy khái niệm này. Đối với mỗi mẫu dữ liệu huấn luyện, một tham số mới \ xi_iđược xác định. Đó là khoảng cách từ mẫu đào tạo tương ứng của nó đến khu vực quyết định chính xác của họ. Đối với những người không được phân loại sai, họ rơi vào các mặt phẳng hỗ trợ tương ứng, vì vậy khoảng cách của họ bằng không.



Vì vậy, vấn đề tối ưu hóa mới là:

\ min_ {w, b_ {0}} L (w, b_0) = || w || ^ {2} + C \ sum_ {i} {\ xi_ {i}} \ text {chủ đề} y_ {i} (w ^ {T} x_ {i} + b_ {0}) \ geq 1 - \ xi_ {i} \ text {và} \ xi_ {i} \ geq 0 \ text {} \ forall i

Nên chọn tham số C như thế nào? Rõ ràng là câu trả lời cho câu hỏi này phụ thuộc vào cách dữ liệu đào tạo được phân phối. Mặc dù không có câu trả lời chung, nhưng rất hữu ích khi tính đến các quy tắc sau:

* Các giá trị lớn của C đưa ra các giải pháp với ít lỗi phân loại sai hơn nhưng biên độ nhỏ hơn. Hãy xem xét rằng trong trường hợp này rất tốn kém để mắc lỗi phân loại sai. Vì mục đích của tối ưu hóa là để giảm thiểu đối số, nên có ít lỗi phân loại sai được cho phép.
* Các giá trị nhỏ của C đưa ra các giải pháp có biên độ lớn hơn và nhiều lỗi phân loại hơn. Trong trường hợp này, việc tối thiểu hóa không xem xét nhiều đến hạn của tổng nên nó tập trung nhiều hơn vào việc tìm một siêu phẳng có biên độ lớn.

# Hiểu K-Phương tiện Clustering

## mục tiêu

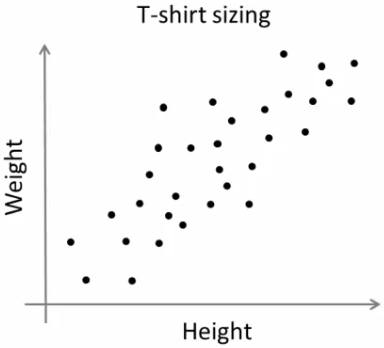
Trong chương này, chúng ta sẽ hiểu các khái niệm về K-Means Clustering, cách thức hoạt động, v.v.

## Lý thuyết

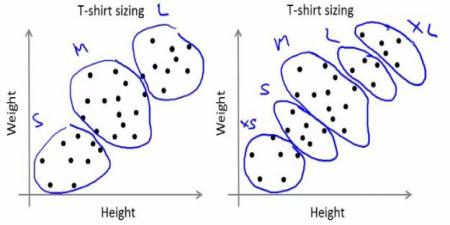
Chúng tôi sẽ giải quyết điều này với một ví dụ thường được sử dụng.

### T-shirt vấn đề kích thước

Hãy xem xét một công ty, sẽ phát hành một mẫu áo phông mới ra thị trường. Rõ ràng họ sẽ phải sản xuất các mô hình ở các kích cỡ khác nhau để làm hài lòng mọi người ở mọi quy mô. Vì vậy, công ty tạo ra một dữ liệu về chiều cao và cân nặng của mọi người, và vẽ chúng vào biểu đồ, như sau:



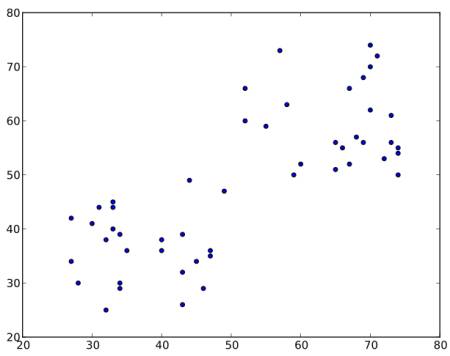
Công ty không thể tạo áo thun với tất cả các kích cỡ. Thay vào đó, họ phân chia mọi người thành Nhỏ, Trung bình và Lớn và chỉ sản xuất 3 mô hình này sẽ phù hợp với tất cả mọi người. Việc nhóm người này thành ba nhóm có thể được thực hiện bằng cách phân cụm k-mean và thuật toán cung cấp cho chúng tôi 3 kích thước tốt nhất, sẽ làm hài lòng tất cả mọi người. Và nếu không, công ty có thể chia mọi người thành nhiều nhóm hơn, có thể là năm, v.v. Kiểm tra hình ảnh dưới đây:



### Làm thế nào nó hoạt động ?

Thuật toán này là một quá trình lặp đi lặp lại. Chúng tôi sẽ giải thích từng bước với sự trợ giúp của hình ảnh.

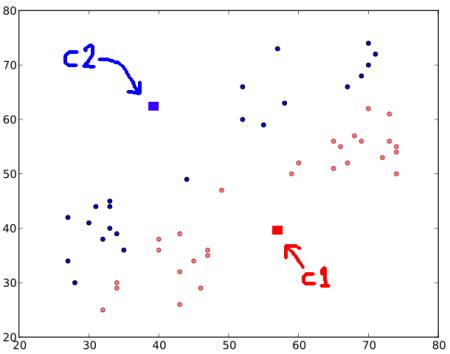
Xem xét một tập hợp dữ liệu như dưới đây (Bạn có thể coi đó là sự cố áo phông). Chúng ta cần phân cụm dữ liệu này thành hai nhóm.



**Bước: 1** - Thuật toán chọn ngẫu nhiên hai trọng tâm C1và C 2(đôi khi, bất kỳ hai dữ liệu nào cũng được lấy làm trọng tâm).

**Bước: 2** - Nó tính toán khoảng cách từ mỗi điểm đến cả hai tâm. Nếu dữ liệu kiểm tra gần hơn C1, thì dữ liệu đó được gắn nhãn '0'. Nếu nó gần hơn C 2, thì được gắn nhãn là '1' (Nếu có nhiều centroid hơn, được gắn nhãn là '2', '3', v.v.).

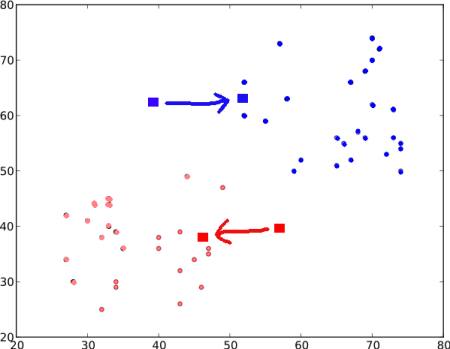
Trong trường hợp của chúng tôi, chúng tôi sẽ tô màu tất cả '0' được dán nhãn màu đỏ và '1' được dán nhãn màu xanh. Vì vậy, chúng tôi nhận được hình ảnh sau khi hoạt động trên.



**Bước: 3** - Tiếp theo, chúng tôi tính trung bình của tất cả các điểm xanh và điểm đỏ riêng biệt và đó sẽ là trọng tâm mới của chúng tôi. Đó là C1và C 2chuyển sang centroid mới được tính toán. (Hãy nhớ rằng, hình ảnh hiển thị không phải là giá trị thực và không đúng tỷ lệ, nó chỉ dành cho trình diễn).

Và một lần nữa, thực hiện bước 2 với trọng tâm mới và dữ liệu nhãn thành '0' và '1'.

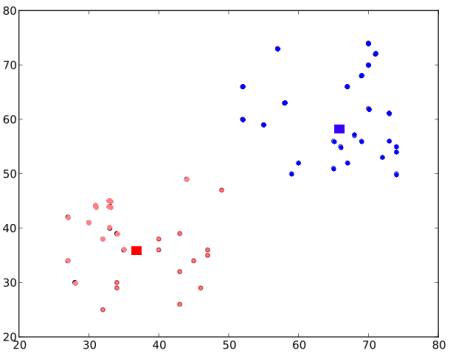
Vì vậy, chúng tôi nhận được kết quả như sau:



Bây giờ **Bước - 2** và **Bước - 3** được lặp lại cho đến khi cả hai tâm được hội tụ đến các điểm cố định. (Hoặc có thể bị dừng tùy thuộc vào các tiêu chí chúng tôi cung cấp, như số lần lặp tối đa hoặc độ chính xác cụ thể đạt được, v.v.) **Những điểm này sao cho tổng khoảng cách giữa dữ liệu thử nghiệm và trọng tâm tương ứng của chúng là tối thiểu** . Hoặc đơn giản, tổng khoảng cách giữa C1 \ leftrightarrow Red \ _Pointsvà C2 \ leftrightarrow Blue \ _Pointslà tối thiểu.

thu nhỏ \; \ bigg [J = \ sum_ {Tất cả \: Red_Points} khoảng cách (C1, Red \ _Point) + \ sum_ {Tất cả \: Blue \ _Points} khoảng cách (C2, Blue \ _Point) \ bigg]

Kết quả cuối cùng trông giống như dưới đây:



Vì vậy, đây chỉ là một sự hiểu biết trực quan về K-Means Clustering. Để biết thêm chi tiết và giải thích toán học, vui lòng đọc bất kỳ sách giáo khoa máy học tiêu chuẩn hoặc kiểm tra các liên kết trong các tài nguyên bổ sung. Nó chỉ là một lớp trên cùng của cụm K-Means. Có rất nhiều sửa đổi cho thuật toán này như, làm thế nào để chọn trọng tâm ban đầu, làm thế nào để tăng tốc quá trình lặp, v.v.