**MỤC LỤC**

1. Nhắc lại bài toán tính khoảng cách
2. Nhắc lại bài toán phân chia hai classes ([Perceptron](https://machinelearningcoban.com/2017/01/21/perceptron/))
3. Thuật toán SVM
   1. Ý tưởng
   2. Xây dựng bài toán tối ưu cho SVM
   3. Soft Margin Support Vector Machine
   4. Kernel Support Vector Machine
4. **Nhắc lại bài toán tính khoảng cách**

Trong không gian 2 chiều, ta biết rằng khoảng cách từ một điểm có toạ độ  tới đường thẳng có phương trình  được xác định bởi:

Trong không gian ba chiều, khoảng cách từ một điểm có toạ độ tới một  mặt phẳng có phương trình  được xác định bởi:

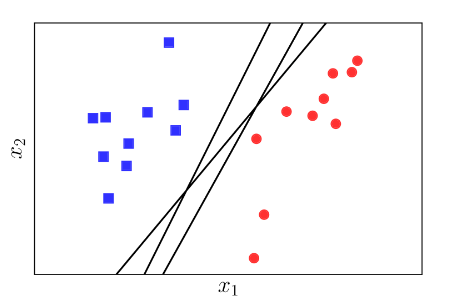
Hơn nữa, nếu ta bỏ dấu trị tuyệt đối ở tử số, ta có thể xác định được điểm đó nằm về phía nào của đường thẳng hay mặt phẳng đang xét. Những điểm làm cho biểu thức trong dấu giá trị tuyệt đối mang dấu dương nằm về cùng 1 phía (tôi tạm gọi đây là phía dương của đường thẳng), những điểm làm cho biểu thức trong dấu giá trị tuyệt đối mang dấu âm nằm về phía còn lại (gọi là phía âm). Những điểm nằm trên đường thẳng/mặt phẳng sẽ làm cho tử số có giá trị bằng 0, tức khoảng cách bằng 0.

Việc này có thể được tổng quát lên không gian nhiều chiều: Khoảng cách từ một điểm (vector) có toạ độ  tới siêu mặt phẳng (hyperplane) có phương trình được xác định bởi:

Với với d là số chiều của không gian

1. **Nhắc lại bài toán phân chia hai classes (**[**Perceptron**](https://machinelearningcoban.com/2017/01/21/perceptron/)**)**

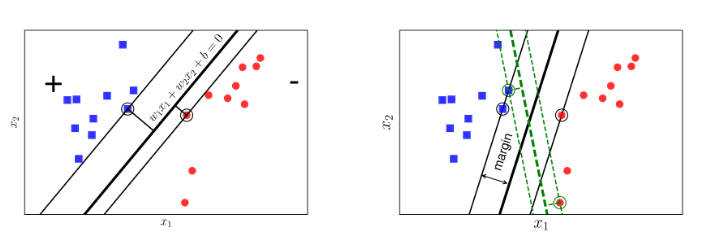
Giả sử rằng có hai class khác nhau được mô tả bởi các điểm trong không gian nhiều chiều, hai classes phân tách tuyến tính *(linearly separable)*, tức tồn tại một siêu phẳng phân chia chính xác hai classes đó. Hãy tìm một siêu mặt phẳng phân chia hai classes đó, tức tất cả các điểm thuộc một class nằm về cùng một phía của siêu mặt phẳng đó và ngược phía với toàn bộ các điểm thuộc class còn lại. Chúng ta đã biết rằng, thuật toán [Perceptron Learning Algorithm (PLA)](https://machinelearningcoban.com/2017/01/21/perceptron/) có thể làm được việc này nhưng nó có thể cho chúng ta vô số nghiệm như hình dưới đây:



Hình 1: Các mặt phân cách hai classes linearly separable.

1. **Thuật toán SVM**
   1. **Ý tưởng**

Từ hình 1, ta có thể vẽ vô số đường phân cách. Trong ba đường thẳng ở hình 1 phía trên, có hai đường thẳng khá lệch về phía class hình tròn đỏ. Điều này có thể khiến cho lớp màu đỏ không vui vì lãnh thổ xem ra bị lấn nhiều quá. Liệu có cách nào để tìm được đường phân chia tối ưu nhất cho 2 classes.



Hình 2: Margin của hai classes là bằng nhau và lớn nhất có thể.

Để làm điều này, chúng ta cần một đường phân chia sao cho khoảng cách từ điểm gần nhất của mỗi class (các điểm được khoanh tròn ở hình 2 bên trái) tới đường phân chia là như nhau, như thế thì mới công bằng. Khoảng cách như nhau này được gọi là margin (lề).

Chúng ta xét tiếp hình 2 bên phải khi khoảng cách từ đường phân chia tới các điểm gần nhất của mỗi class là như nhau. Xét hai cách phân chia bởi đường nét liền màu đen và đường nét đứt màu lục, đường nào sẽ làm cho cả hai class hạnh phúc hơn? Rõ ràng đó phải là đường nét liền màu đen vì nó tạo ra một margin rộng hơn.

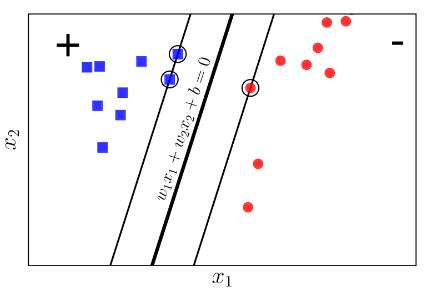
Việc margin rộng hơn sẽ mang lại hiệu ứng phân lớp tốt hơn vì sự phân chia giữa hai classes là rạch ròi hơn. Việc này là một điểm khá quan trọng giúp Support Vector Machine mang lại kết quả phân loại tốt hơn so với Neural Network với 1 layer, tức Perceptron Learning Algorithm.

Bài toán tối ưu trong Support Vector Machine (SVM) chính là bài toán đi tìm đường phân chia sao cho margin là lớn nhất. Đây cũng là lý do vì sao SVM còn được gọi là Maximum Margin Classifier.

* 1. **Xây dựng bài toán tối ưu cho SVM**

Giả sử rằng các cặp dữ liệu của *training set* là , , ,…,, với vector  thể hiện *đầu vào* của một điểm dữ liệu và  là *nhãn* của điểm dữ liệu đó. d là số chiều của dữ liệu và N là số điểm dữ liệu. Giả sử rằng *nhãn* của mỗi điểm dữ liệu được xác định bởi  (class 1) hoặc (class 2) giống như trong PLA.

Để dễ hình dung, chúng ta cùng xét trường hợp trong không gian hai chiều dưới đây. *Không gian hai chiều giúp ta dễ hình dung, các phép toán hoàn toàn có thể được tổng quát lên không gian nhiều chiều.*



Hình 3: Phân tích bài toán SVM

Giả sử rằng các điểm vuông xanh thuộc class 1, các điểm tròn đỏ thuộc class -1 và mặt là mặt phân chia giữa hai classes (Hình 3). Hơn nữa, class 1 nằm về phía dương, class -1 nằm về phía âm của mặt phân chia. Nếu ngược lại, ta chỉ cần đổi dấu của w và b. Chú ý rằng chúng ta cần đi tìm các hệ số w và b.

Ta quan sát thấy một điểm quan trọng sau đây: với cặp dữ liệu  bất kỳ, khoảng cách từ điểm đó tới mặt phân chia là:

Điều này có thể dễ nhận thấy vì theo giả sử ở trên,  luôn cùng dấu với phía của . Từ đó suy ra  cùng dấu với , và tử số luôn là 1 số không âm.

Với mặt phần chia như trên, margin được tính là khoảng cách gần nhất từ 1 điểm tới mặt đó (bất kể điểm nào trong hai classes):

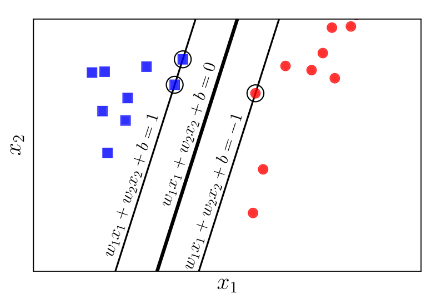
Bài toán tối ưu trong SVM chính là bài toán tìm w và b sao cho margin này đạt giá trị lớn nhất:

**(1)**

Việc giải trực tiếp bài toán này sẽ rất phức tạp, nhưng ta thấy có cách để đưa nó về bài toán đơn giản hơn.

Nhận xét quan trọng nhất là nếu ta thay vector hệ số w bởi kw và b bởi kb trong đó k là một hằng số dương thì mặt phân chia không thay đổi, tức khoảng cách từ từng điểm đến mặt phân chia không đổi, tức margin không đổi. Dựa trên tính chất này, ta có thể giả sử:

**với những điểm nằm gần mặt phân chia nhất**như hình 4 dưới đây:



Hình 4: Các điểm gần mặt phân cách nhất của hai classes được khoanh tròn.

Như vậy, với mọi n, ta có:

Vậy bài toán tối ưu (1) có thể đưa về bài toán tối ưu có ràng buộc sau đây:

**(2)**

Bằng 1 biến đổi đơn giản, ta có thể đưa bài toán này về bài toán dưới đây:  
 **(3)**

Ở đây, chúng ta đã lấy nghịch đảo hàm mục tiêu, bình phương nó để được một hàm khả vi, và nhân với  để biểu thức đạo hàm đẹp hơn.

Xác định class cho một điểm dữ liệu mới: Sau khi tìm được mặt phân cách , class của bất kỳ một điểm nào sẽ được xác định đơn giản bằng cách:

Trong đó hàm sgn là hàm xác định dấu, nhận giá trị 1 nếu đối số là không âm và -1 nếu ngược lại.

Tóm lại với bài toán binary classification mà 2 classes là *linearly separable*, có vô số các siêu mặt phẳng giúp phân biệt hai classes, tức mặt phân cách. Với mỗi mặt phân cách, ta có một *classifier*. Khoảng cách gần nhất từ 1 điểm dữ liệu tới mặt phân cách ấy được gọi là *margin* của classifier đó.

Support Vector Machine là bài toán đi tìm mặt phân cách sao cho *margin* tìm được là lớn nhất, đồng nghĩa với việc các điểm dữ liệu *an toàn nhất* so với mặt phân cách. Với các bài toán mà dữ liệu gần linearly separable hoặc nonlinear separable, có những cải tiền khác của SVM để thích nghi với dữ liệu đó.

* **Lagrangian của bài toán SVM**

[Lagrangian](https://machinelearningcoban.com/2017/04/02/duality/#-lagrangian) của bài toán (3) là:

(4)

với và

* **Hàm đối ngẫu Lagrange**

[Hàm đối ngẫu Lagrange](https://machinelearningcoban.com/2017/04/02/duality/#-ham-doi-ngau-lagrange-the-lagrange-dual-function) được định nghĩa là:

với ⪰ 0

Việc tìm giá trị nhỏ nhất của hàm này theo w và b có thể được thực hiện bằng cách giải hệ phương trình đạo hàm của theo w và b bằng 0:

(5)

(6)

Thay (5) và (6) vào (4) ta thu được g(λ)

(7)

Xét ma trận :

Và vector , ta có thể viết lại g(λ) dưới dạng:

(8)

Đặt , ta có một quan sát quan trọng: K là một [ma trận nửa xác định dương](https://machinelearningcoban.com/2017/03/12/convexity/#positive-semidefinite). Thật vậy, với mọi vector λ, ta có:

Vậy là một [hàm concave](https://machinelearningcoban.com/2017/03/12/convexity/#concave-function).

* **Bài toán đối ngẫu Lagrange**

Từ đó, kết hợp hàm đối ngẫu Lagrange và các điều kiện ràng buộc của λ, ta sẽ thu được [bài toán đối ngẫu Lagrange](https://machinelearningcoban.com/2017/04/02/duality/#-bai-toan-doi-ngau-lagrange-the-lagrange-dual-problem):

Ràng buộc thứ hai được lấy từ (6).

Đây là một bài toán lồi vì ta đang đi tìm giá trị lớn nhất của một hàm mục tiêu là *concave* trên một [*polyhedron*](https://machinelearningcoban.com/2017/03/12/convexity/#-giao-cua-cac-tap-loi-la-mot-tap-loi).

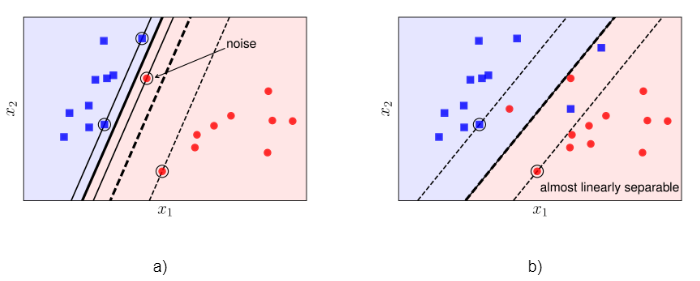
Bài toán này cũng được là một Quadratic Programming và cũng có thể được giải bằng các thư viện như CVXOPT.

Trong bài toán đối ngẫu này, số tham số (parameters) phải tìm là N, là chiều của λ, tức số điểm dữ liệu. Trong khi đó, với bài toán gốc (3), số tham số phải tìm là d + 1, là tổng số chiều của w và b, tức số chiều của mỗi điểm dữ liệu cộng với 1. Trong rất nhiều trường hợp, số điểm dữ liệu có được trong *training set* lớn hơn số chiều dữ liệu rất nhiều.

* 1. **Soft Margin Support Vector Machine**

Giống như [Perceptron Learning Algorithm (PLA)](https://machinelearningcoban.com/2017/01/21/perceptron/), [Support Vector Machine (SVM) thuần](https://machinelearningcoban.com/2017/04/09/smv/) chỉ làm việc khi dữ liệu của 2 classes là [linearly separable](https://machinelearningcoban.com/2017/01/21/perceptron/#bai-toan-perceptron). Một cách tự nhiên, chúng ta cũng mong muốn rằng SVM có thể làm việc với dữ liệu gần linearly separable giống như [Logistic Regression](https://machinelearningcoban.com/2017/01/27/logisticregression/) đã làm được.

Xét 2 ví dụ trong hình 5 dưới đây:



Hình 5: Soft margin SVM. Khi a) có nhiễu hoặc b) dữ liệu gần linearly separable, SVM thuần sẽ không hoạt động hiệu quả.

Có hai trường hợp dễ nhận thấy SVM làm việc không hiệu quả hoặc thậm chí không làm việc:

Trường hợp 1: Dữ liệu vẫn *linearly separable* như hình 5a) nhưng có một điểm *nhiễu* của lớp tròn đỏ ở quá gần so với lớp vuông xanh. Trong trường hợp này, nếu ta sử dụng SVM *thuần* thì sẽ tạo ra một *margin* rất nhỏ. Ngoài ra, đường phân lớp nằm quá gần lớp vuông xanh và xa lớp tròn đỏ. Trong khi đó, nếu ta *hy sinh* điểm nhiễu này thì ta được một *margin* tốt hơn rất nhiều được mô tả bởi các đường nét đứt. SVM *thuần* vì vậy còn được coi là *nhạy cảm với nhiễu* (*sensitive to noise*).

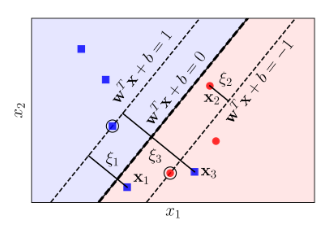
Trường hợp 2: Dữ liệu không *linearly separable* nhưng *gần linearly separable* như Hình 5b). Trong trường hợp này, nếu ta sử dụng SVM *thuần* thì rõ ràng bài toán tối ưu là *infeasible*, tức *feasible set* là một tập rỗng, vì vậy bài toán tối ưu SVM trở nên vô nghiệm. Tuy nhiên, nếu ta lại *chịu hy sinh một chút* những điểm ở gần biên giữa hai classes, ta vẫn có thể tạo được một đường phân chia khá tốt như đường nét đứt đậm. Các *đường support* đường nét đứt mảnh vẫn giúp tạo được một margin lớn cho bộ phân lớp này. Với mỗi điểm nằm lần sang phía bên kia của các đường suport (hay *đường margin*, hoặc *đường biên*) tương ứng, ta gọi điểm đó rơi vào *vùng không an toàn*. Chú ý rằng vùng an toàn của hai classes là khác nhau, giao nhau ở phần nằm giữa hai đường support.

Trong cả hai trường hợp trên, *margin* tạo bởi đường phân chia và đường nét đứt mảnh còn được gọi là *soft margin* (*biên mềm*). Cũng theo cách gọi này, SVM *thuần* còn được gọi là *Hard Margin SVM* (*SVM biên cứng*).

Trong phần này, ta sẽ đề cập đến một biến thể của Hard Margin SVM có tên gọi là Soft Margin SVM.

Như đã đề cập phía trên, để có một margin lớn hơn trong Soft Margin SVM, chúng ta cần hy sinh một vài điểm dữ liệu bằng cách chấp nhận cho chúng rơi vào vùng không an toàn. Tất nhiên, chúng ta phải hạn chế sự hy sinh này, nếu không, chúng ta có thể tạo ra một biên cực lớn bằng cách hy sinh hầu hết các điểm. Vậy hàm mục tiêu nên là một sự kết hợp để tối đa margin và tối thiểu sự hy sinh.

Giống như với Hard Margin SVM, việc tối đa margin có thể đưa về việc tối thiểu . Để xác định sự hy sinh, chúng ta cùng theo dõi hình 6 dưới đây:



Hình 6: Giới thiệu các biến slack ξn. Với những điểm nằm trong vùng an toàn ξn = 0. Những điểm nằm trong vùng không an toàn nhưng vẫn đúng phía so với đường phân chia tương ứng với các 0 < ξn <1, ví dụ x2. Những điểm nằm ngược phía với class của chúng so với đường boundary ứng với các ξn > 1, ví dụ như x1 và x3.

Với mỗi điểm xn trong tập toàn bộ dữ liệu huấn luyện, ta giới thiệu thêm một biến đo sự hy sinh ξn tương ứng. Biến này còn được gọi là slack variable. Với những điểm xn nằm trong vùng an toàn, ξn = 0. Với mỗi điểm nằm trong vùng không an toàn như x1, x2 hay x3, ta có ξi > 0. Nhận thấy rằng nếu yi = ± 1 là nhãn của xi trong vùng không an toàn thì

Với bài toán toán tối ưu cho Hard Margin SVM, ta có:

Với Soft Margin SVM, hàm mục tiêu sẽ có thêm một số hạng nữa giúp tối thiểu sự hy sinh. Từ đó ta có hàm mục tiêu:

trong đó C là một hằng số dương và

Hằng số C được dùng để điều chỉnh tầm quan trọng giữa margin và sự hy sinh. Hằng số này được xác định từ trước bởi người lập trình hoặc có thể được xác định bởi [cross-validation](https://machinelearningcoban.com/2017/03/04/overfitting/#-cross-validation).

Điều kiện ràng buộc sẽ thay đổi một chút. Với mỗi cặp dữ liệu (xn, yn) thay vì ràng buộc cứng , chúng ta sẽ có ràng buộc mềm:

và ràng buộc phụ ,

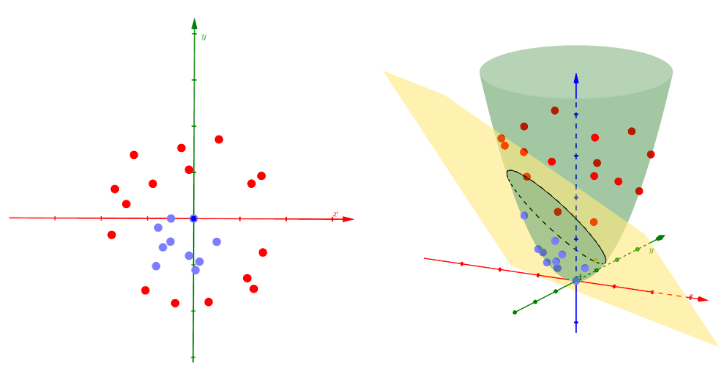
Tóm lại, ta sẽ có bài toán tối ưu ở dạng chuẩn cho Soft-margin SVM:

* 1. **Kernel Support Vector Machine**

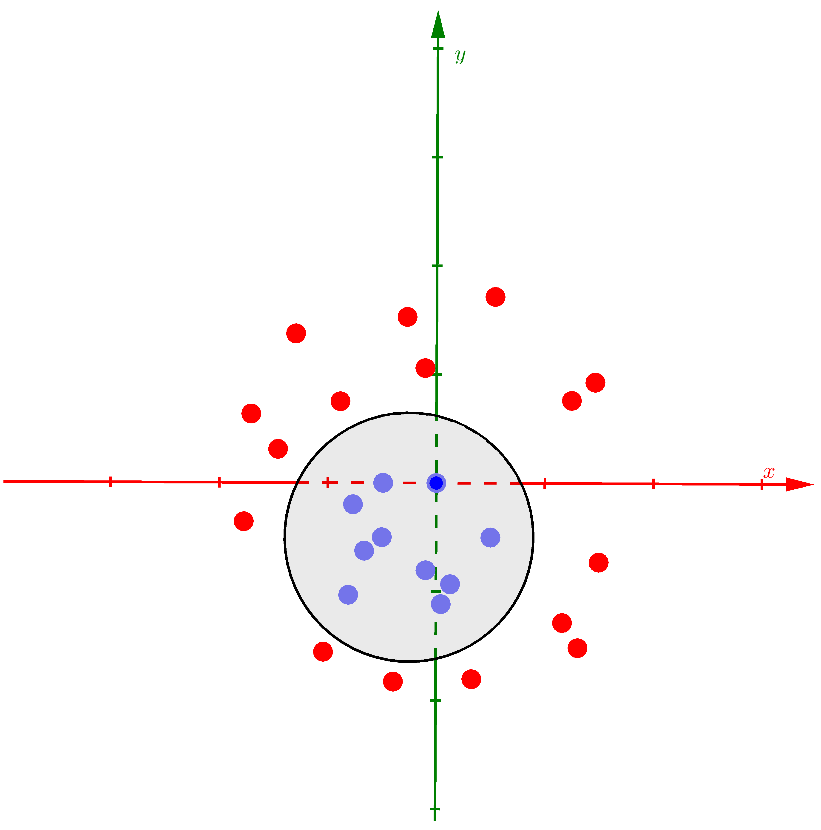
Kernel SVM là việc áp dụng SVM lên bài toán mà dữ liệu giữa hai classes là hoàn toàn không linear separable (không phân biệt tuyến tính).

Ý tưởng cơ bản của Kernel SVM và các phương pháp kernel nói chung là tìm một phép biến đổi sao cho dữ liệu ban đầu là *không phân biệt tuyến tính* được biến sang không gian mới. Ở không gian mới này, dữ liệu trở nên *phân biệt tuyến tính*.

Xét ví dụ tại hình 7 dưới đây với việc biến dữ liệu *không phân biệt tuyến tính* trong không gian hai chiều thành *phân biệt tuyến tính* trong không gian ba chiều bằng cách giới thiệu thêm một chiều mới:



1. (b)



(c)

Hình 7: Ví dụ về Kernel SVM. a) Dữ liệu của hai classes là không phân biệt tuyến tính trong không gian hai chiều. b) Nếu coi thêm chiều thứ ba là một hàm số của hai chiều còn lại , các điểm dữ liệu sẽ được phân bố trên 1 parabolic và đã trở nên phân biệt tuyến tính . Mặt phẳng màu vàng là mặt phân chia, có thể tìm được bởi Hard/Soft Margin SVM. c) Giao điểm của mặt phẳng tìm được và mặt parabolic là một đường ellipse, khi chiếu toàn bộ dữ liệu cũng như đường ellipse này xuống không gian hai chiều ban đầu, ta đã tìm được đường phân chia hai classses.

Kernel SVM là việc đi tìm một hàm số biến đổi dữ liệu x từ không gian feature ban đầu thành dữ liệu trong một không gian mới bằng hàm số Φ(x). Trong ví dụ này, hàm Φ() đơn giản là giới thiệu thêm một chiều dữ liệu mới (một feature mới) là một hàm số của các features đã biết. Hàm số này cần thỏa mãn mục đích của chúng ta: trong không gian mới, dữ liệu giữa hai classes là phân biệt tuyến tính hoặc gần như phần biệt tuyến tính. Khi đó, ta có thể dùng các bộ phân lớp tuyến tính thông thường như PLA, Logistic Regression, hay Hard/Soft Margin SVM.

Các hàm Φ() thường tạo ra dữ liệu mới có số chiều cao hơn số chiều của dữ liệu ban đầu, thậm chí là vô hạn chiều. Nếu tính toán các hàm này trực tiếp, chắc chắn chúng ta sẽ gặp các vấn đề về bộ nhớ và hiệu năng tính toán. Có một cách tiếp cận là sử dụng các kernel functions mô tả quan hệ giữa hai điểm dữ liệu bất kỳ trong không gian mới, thay vì đi tính toán trực tiếp từng điểm dữ liệu trong không gian mới.

* **Bảng tóm tắt các kernel thông dụng**

| **Tên** | **Công thức** | **kernel** | **Thiết lập hệ số** |
| --- | --- | --- | --- |
| linear |  | 'linear' | không có hệ số |
| polynomial |  | 'poly' | d: degree, γ: gamma, r: coef0 |
| sigmoid |  | 'sigmoid' | γ: gamma, r: coef0 |
| rbf |  | 'rbf' | γ>0: gamma |

1. **Bài toán minh họa**
   1. **Ý tưởng**

Giả sử một công ty muốn quảng bá sản phẩm cho nhãn hiệu ô tô mới của họ và muốn biết loại người nào có khả năng mua chiếc ô tô mới của họ để họ có thể có quảng cáo nhắm mục tiêu đến những người đó. Vì vậy, công ty đã liên hệ với một công ty quảng cáo mạng xã hội có dữ liệu từ một chiến dịch quảng bá thành công tương tự khác. Bây giờ, công ty muốn tạo ra một mô hình giúp đạt được mục tiêu này.

* 1. **Dữ liệu**

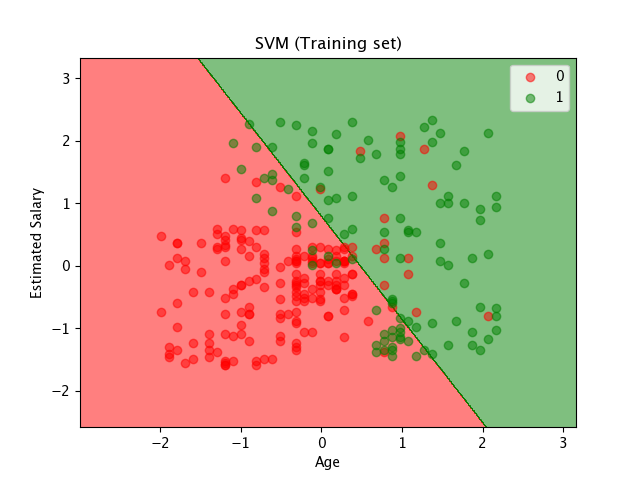
Tập dữ liệu chứa 400 mục bao gồm các thông tin là userId, gender, age, estimatedsalary và the purchased. Ma trận các feature (tính năng) được tính đến là độ tuổi và mức lương ước tính sẽ dự đoán người dùng có mua xe mới hay không (1 = Có, 0 = Không).

* 1. **Giải pháp**

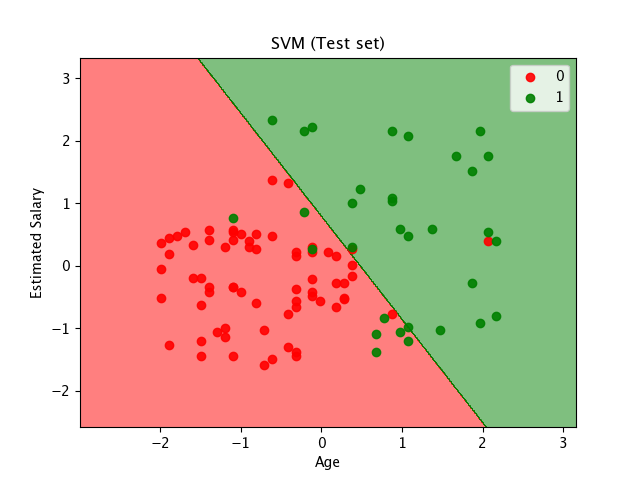
Thực hiện xử lý dữ liệu và sau đó dự đoán được thực hiện bằng SVM. Tập dữ liệu được chia thành tỷ lệ 75/25 (training set = 0,75 & test set = 0,25).

* 1. **Kết quả**

Hình 8: dữ liệu của tập huấn luyện (training set) và hình 9: tập kiểm tra (test set) được đưa ra dưới đây. Vì trong svm, kernel được chọn là tuyến tính, do đó ta thấy đồ thị tuyến tính với một đường thẳng



Hình 8: Tập huấn luyện



Hình 9: Tập test

* 1. **Mô tả chạy thực nghiệm**

Tập dữ liệu chứa 400 mục bao gồm các thông tin là userId, gender, age, estimatedsalary và the purchased. Ma trận các feature (tính năng) được tính đến là độ tuổi và mức lương ước tính sẽ dự đoán người dùng có mua xe mới hay không (1 = Có, 0 = Không).

Dữ liệu huấn luyện

(75% data)

Các thuật toán

Bộ phân lớp

(Mô hình)

**Dữ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User ID** | **Gender** | **Age** | **EstimatedSalary** | **Purchased** |
| **15624510** | **Male** | **19** | **19000** | **0** |
| **15810944** | **Male** | **35** | **20000** | **0** |
| **15668575** | **Female** | **26** | **43000** | **1** |
| **15603246** | **Female** | **27** | **57000** | **0** |

(Hình:Xây dựng mô hình)

Dữ liệu chưa phân lớp

0  
1

Bộ phân lớp

Dữ liệu kiểm tra

(25% data)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User ID** | **Gender** | **Age** | **EstimatedSalary** | **Purchased** |
| **15624510** | **Male** | **19** | **19000** | **0** |
| **15810944** | **Male** | **35** | **20000** | **0** |
| **15668575** | **Female** | **26** | **43000** | **1** |
| **15603246** | **Female** | **27** | **57000** | **0** |

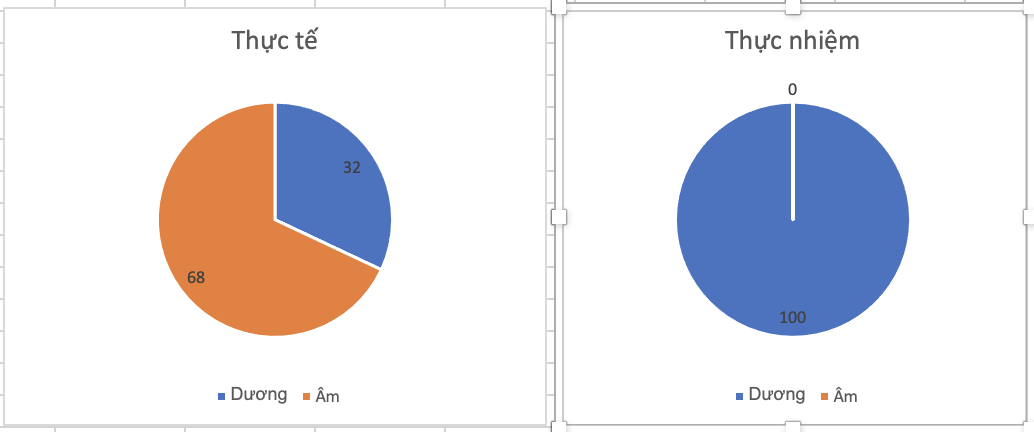
(Hình ứng dụng mô hình)

Kết quả thực nghiệm áp dụng theo công thức.

Cách thứ nhất dựa theo bài toán (9) và các công thức (15) và (16).

Gõ lại công thức (15) (16) giùm anh svm bài 19.

Kết dữ liệu trước và sau khi chạy thực nghiệm:

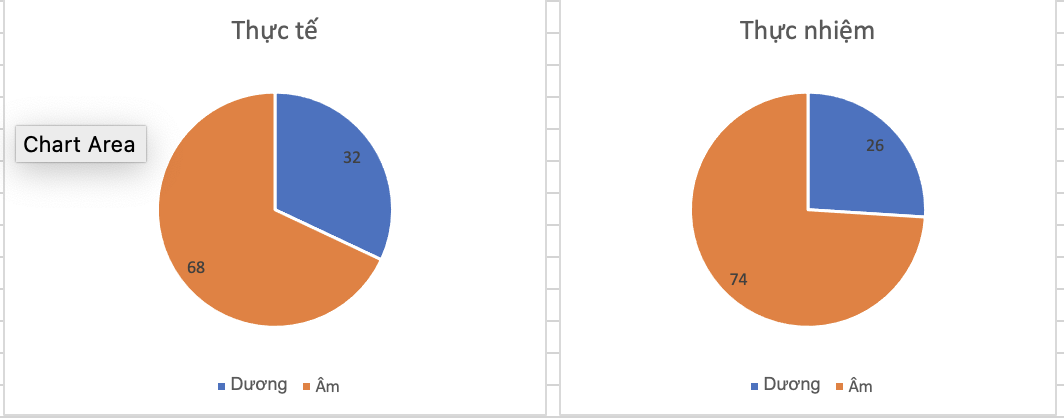


Nhận xét rằng với việc tìm margin theo công thức (15) và (16) trên tập dữ liệu mà ta đang xét không thể tìm ra được đường phân cách giữa hai lớp dữ liệu. Cần phải tối ưu lại thuật toán sử dụng **Soft Margin.**

**Tìm tham số C theo công thức:**

Ta chọn C= 5, và viết lại thuật toán

Kết quả thực nghiệm như sau:



Câu hỏi đặt ra làm sao để tối ưu C để có thể phân lớp dữ liệu một cách tốt nhất.

Ứng dụng thư viện Sklearn và tìm C tối ưu:

From sklearn.svm import SVC

classifier = SVC(kernel = 'linear', C=5)

classifier.fit(X\_Train, Y\_Train)

w = classifier.coef\_

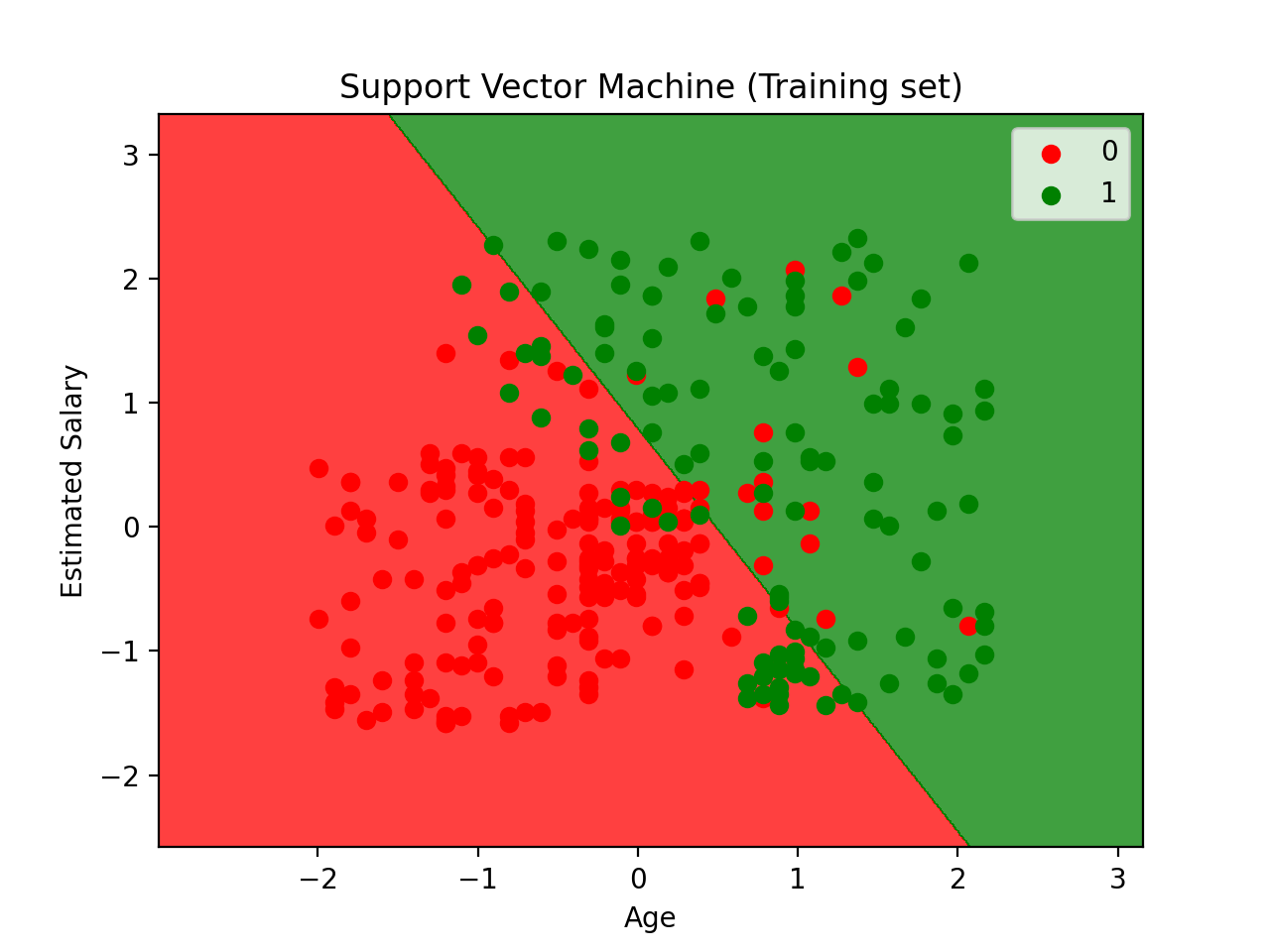
b = classifier.intercept\_

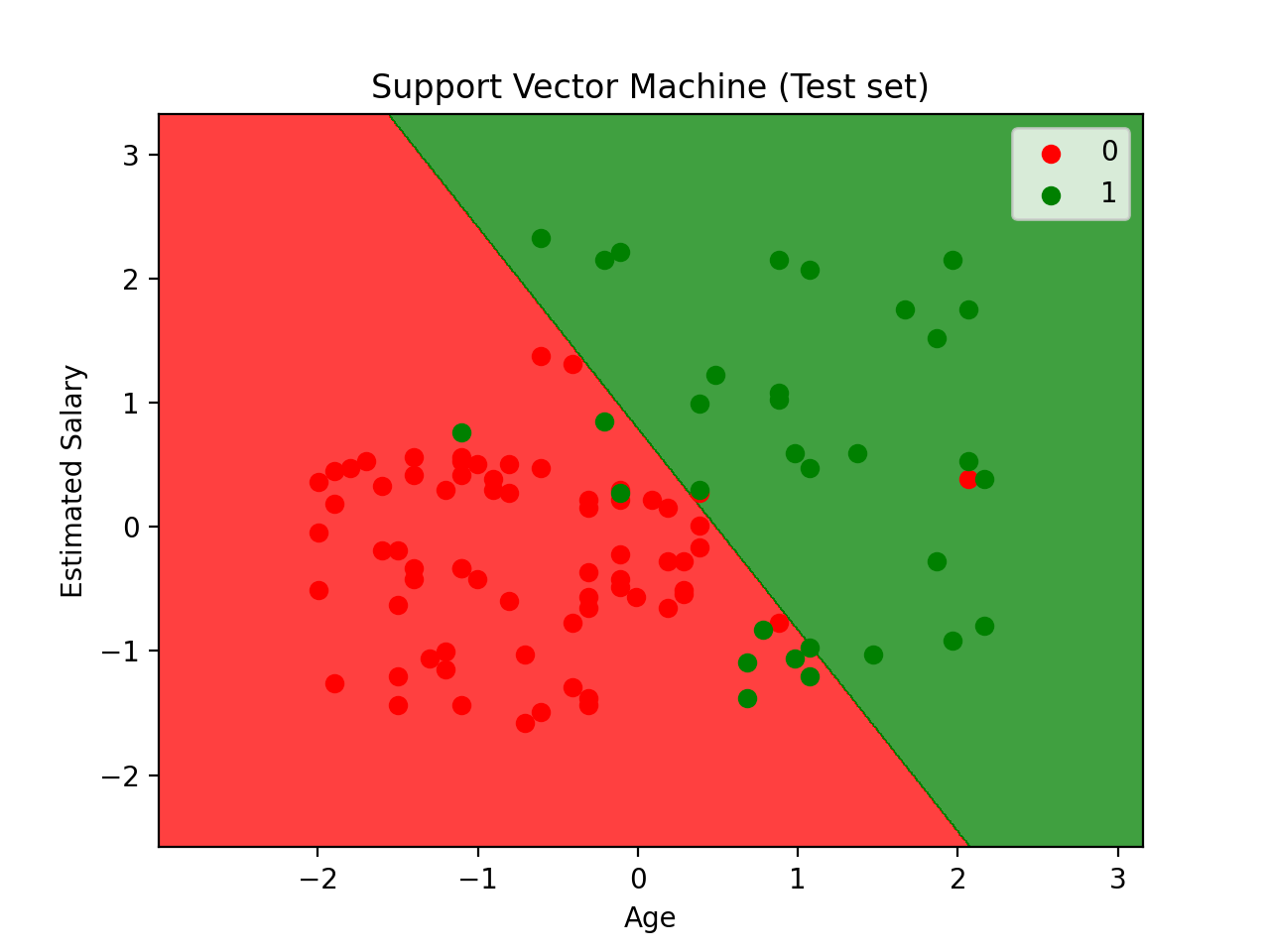
Y\_Pred = classifier.predict(X\_Test)

from sklearn.metrics import confusion\_matrix

print("Accuracy: %.2f %%" %(100\*accuracy\_score(Y\_Test, Y\_Pred)))

#Kết quả đạt được 89%.





Cách tìm tham số C để tối ưu nhất với thư viện:

from sklearn.model\_selection import GridSearchCV

parameter\_candidates = [

{'C': [0.001, 0.01, 0.1, 1, 5, 10, 100, 1000,1e5], 'kernel': ['linear']},

]

clf = GridSearchCV(estimator=SVC(), param\_grid=parameter\_candidates, n\_jobs=-1)

clf.fit(X\_Train, Y\_Train)

print('Best score:', clf.best\_score\_)

print('Best C:',clf.best\_estimator\_.C)

Kết qủa được C=5.

Nhận xét: Với việc tối ưu C bằng công thức hoặc thư viện Sklearn đều cho kết qủa tốt và phân lớp được dữ liệu một cách rõ ràng đạt tỉ lệ cao khoảng 89%.