**CHƯƠNG 1:**

**THUẬT TOÁN SVM**

**1.1. Định nghĩa:**

SVM - viết tắt của Support Vector Machine.

Thuật toán SVM ban đầu được tìm ra bởi Vladimir N. Vapnik và dạng chuẩn hiện nay sử dụng lề mềm được tìm ra bởi Vapnik và Corinna Cortes năm 1995.

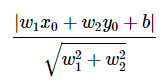
SVM là một phương pháp phân loại xuất phát từ lý thuyết thống kê, dựa trên nguyên tắc tối thiểu rủi ro cấu trúc (Structuaral Risk Minimisation). SVM sẽ cố gắng tìm cách phân loại dữ liệu sao cho có lỗi xảy ra trên tập kiểm tra là nhỏ nhất (Test Error Minimization).

Đây là một phương pháp trong lĩnh vực trí tuệ nhân tạo. Vào thời kì đầu SVM xuất hiện, khả năng tính toán của máy tính còn rất hạn chế, nên phương pháp SVM không được lưu tâm. Tuy nhiên những năm trở lại đây các thuật toán sử dụng cho SVM phát triển rất nhanh, cùng với khả năng tính toán mạnh mẽ của máy tính, đã có được những ứng dụng rất to lớn.

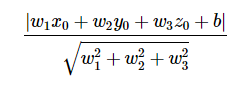
**1.2. Cơ sở lí thuyết:**

***1.2.1. Khoảng cách từ một điểm tới một siêu mặt phẳng:***

Trong không gian 2 chiều, ta biết rằng khoảng cách từ một điểm có toạ độ (x0,y0) tới đường thẳng có phương trình w1x+w2y+b=0 được xác định bởi:



Trong không gian ba chiều, khoảng cách từ một điểm có toạ độ (x0,y0,z0) tới một mặt phẳng có phương trình w1x+w2y+w3z+b=0 được xác định bởi:



Hơn nữa, nếu bỏ dấu trị tuyệt đối ở tử số, chúng 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 (gọi 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/măt phẳng sẽ làm cho tử số có giá trị bằng 0, tức khoảng cách bằng 0.

Tổng quát lên không gian nhiều chiều: Khoảng cách từ một điểm (vector) có toạ độ x0 tới siêu mặt phẳng (hyperplane) có phương trình wTx+b=0 được xác định bởi:



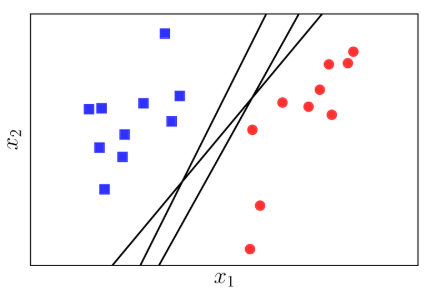
Trong đó với d là số chiều của không gian.

***1.2.2. Bài toán phân chia hai classes:***

Bài toán phân chia hai classes có thể thực hiện bằng thuật toán Perceptron Learning Algorithm (PLA)

Giả sử rằng có hai class khác nhau được mô tả bởi các điểm trong không gian nhiều chiều, hai classes này tuyến tính (linearly separable), tức tồn tại một siêu phẳng phân chia chính xác hai classes đó. Chúng ta phải 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.

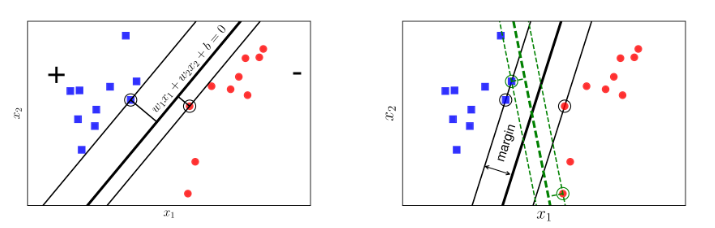
Chúng ta biết rằng, thuật toán PLA có thể làm được việc này nhưng nó có thể cho chúng ta vô số nghiệm như Hình 1 dưới đây:



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

Câu hỏi đặt ra là: trong vô số các mặt phân chia đó, đâu là mặt phân chia tốt nhất?

Chúng ta cần một đường phân chia sao cho khoảng cách từ điểm gần nhất của mỗi class (các điểm được khoanh tròn) tới đường phân chia là như nhau. Khoảng cách như nhau này được gọi là margin (lề).



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

Bài toán phân lớp có thể được mô tả như sau:

Cho tập mẫu 

Tìm ánh xạ 

Trong đó:

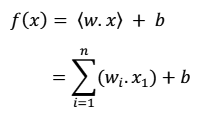
* N: số mẫu
* Xi: mẫu dữ liệu thứ I;
* Ci: lớp của mẫu dữ liệu thứ i
* Ánh xạ f ở đây có thể hiểu là một mô hình các quy tắc, các luật để xác định từng đối tượng xét thuộc về lớp nào dựa trên các đặc trưng của chúng. Trong thực tế, việc xác định các đặc trưng để phân lớp bị ảnh hưởng nhiều bởi nhiều yếu tố gây khó khăn cho quá trình phân lớp như: đối tượng có nhiều thuộc tính, xác định các thuộc tính nào là cần thiết, các thuộc tính nào là không cần thiết.

Bài toán tối ưu trong 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.

**1.2.3. Cơ sở toán học cho bài toán phân chia hai classes:**

Việc phân lớp nhị phân được thực hiện bằng cách sử dụng hàm giá trị thực  là hàm tuyến tính, tương ứng với đầu ra , được phát biểu như sau:

Đầu vào  sẽ được gán vào lớp có nhãn 1 nếu , trường hợp ngược lại sẽ được gán vào lớp có nhãn 0. Trường hợp f(x) là hàm tuyến tính của  ta có thể viết như sau :



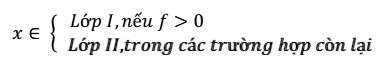
Trong đó  là biểu thị tích vô hướng

Về mặt hình học, các phần tử của không gian đầu vào X sẽ được rơi vào một trong hai phần được phân tách bởi siêu phẳng xác định bằng biểu thức:



Trong đó: w là vector pháp tuyến của siêu phẳng, gí trị ngưỡng b thay đổi có thể tạo ra các siêu phẳng song song với nhau.

Với mỗi mẫu dữ liệu x chưa xác định sẽ được phân chia thành



**1.2.4. Ma trận GRAM:**

Cho tập  các vector trong không gian tích vô hướng X, ma trận G kích thước  được gọi là ma trận GRAM.

Đặc điểm quan trọng của ma trận GRAM là: các dữ liệu đầu vào cho các chương trình tổng hợp hoặc khái quát hoàn toàn có thể biểu diễn thông qua ma trận GRAM.

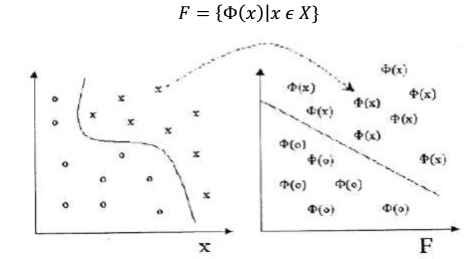
**1.2.5. Không gian đặc trưng:**

Sự phực tạp của hàm mục tiêu dẫn đến quá trình học phụ thuộc vào cách nó được diễn tả. Khi diễ tả dữ liệu một cách phù hợp, vấn đề học sẽ trở nên dễ dàng. Vì vậy, một việc làm phổ biến trong máy học là chuyển đổi dữ liệu từ không gian đầu vào X sang không gian đặc trưng:



Trong đó n là số chiều của đầu vào (số thuộc tính) và N là số chiều của không gian đặc trưng. Dữ liệu sẽ được chuyển vào không gian đặc trưng với N > n.

Không gian đặc trưng ký hiệu là F:



Hình 3. Ánh xạ từ không gian dữ liệu X sang không gian đặc trưng F

**1.2.5. Không gian đặc trưng:**

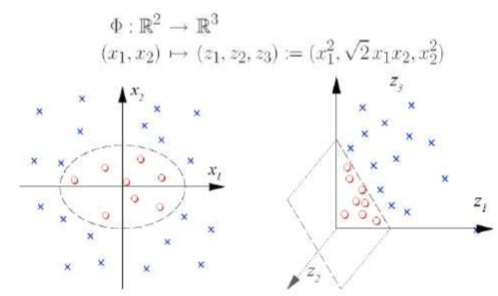
**a) Khái niệm**

Một hàm hạt nhân là một hàm K sao cho với mọi  , ta có:



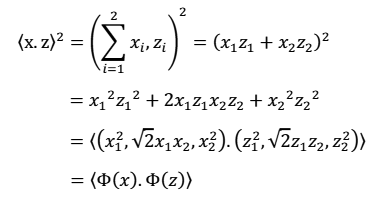
Ở đây là tích vô hướng trong không gian đặc trưng.

Ví dụ: Xét phép biến đổi dữ liệu từ không gian đầu vào  vào không gian đặc trưng  được cho bởi:



Hình 4: Ví dụ hàm hạt nhân

Khi ta chọn x = (x1, x2) và z = (z1, z2) ta có:



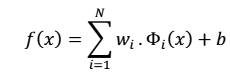
Ta thấy, khi chọn . Vì vậy ta không cần tính ánh xạ .

**b) Hàm hạt nhân trong máy học tuyến tính:**

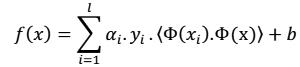
Máy học không tuyến tính trên không gian đầu vào được xây dựng qua hai bước:

* Sử dụng một ánh xạ không tuyến tính để chuyển đổi dữ liệu vào không gian đặc trưng;
* Sử dụng máy học phân lớp tuyến tính trong không gian đặc trưng.

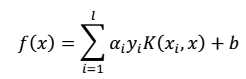
Máy học tuyến tính trong không gian đặc trưng tưng ứng với hàm:



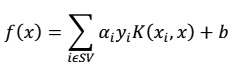
Chúng ta không cần xác định tường minh trọng số w, khi triển khai tiếp bằng cách đưa vào vector , ta có:



Hơn nữa, cũng không cần xây dựng tường minh ánh xạ  , nhờ sử dụng hàm hạt nhân nên:

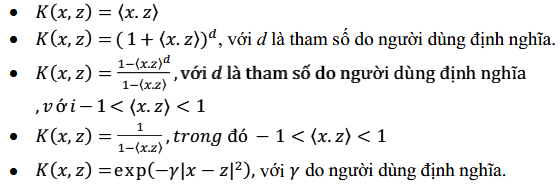


Đặt SV là tập các chỉ số i thỏa mãn , ta được:



Các xi ứng với được gọi là cá vector trợ giúp (Support vector), hàm phân lớp sẽ được xác định duy nhất qua các vector này.

c) Một số hàm hạt nhân:



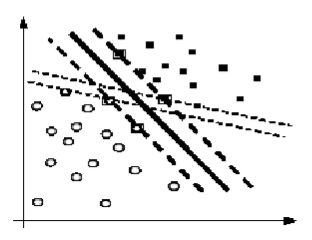
**1.3. Phương pháp SVM:**

**1.3.1. Ý tưởng:**

Cho trước một tập huấn luyện, được biểu diễn trong không gian vector, trong đó mỗi tài liệu là một điểm, phương pháp này tìm ra một siêu phẳng f quyết định tốt nhất cóa thể chia các điểm trên không gian này thành hai lớp riêng biệt tương ứng là lớp + và lớp -. Chất lượng của siêu phẳng này quyết định bở khoảng cách (gọi là biên) của diểm dữ liệu gần nhất của mỗi lớp đến mặt phẳng này. Khi đó khoảng cách biên càng lớn thì mặt phẳng quyết định càng tốt, đồng thời việc phân loại càng chính xác.

Ý tưởng của nó là ánh xạ (tuyến tính hoặc phi tuyến) dữ liệu vào không gian các vector đặc trưng (space of feature vectors) mà ở đó một siêu phẳng tối ưu được tìm ra để tách dữ liệu thuộc hai lớp khác nhau.

Mục đích của phương pháp SVM là tìm được khoảng cách biên lớn nhất:



Hình Mô tả phương pháp SVM

Đường tô đậm là siêu phẳng tốt nhất và các điểm được bao bởi hình chữ nhật là những điểm gần siêu phẳng nhất, chúng được gọi là các vector hỗ trợ (support vector). Các đường nét đứt mà các support vector nằm trên đó gọi là lề (margin).

**1.3.2. Cơ sở lí thuyết:**

SVM thực chất là một bài toán tối ưu, mục tiêu của thuật toán này là tìm được một không gian F và siêu phẳng quyết định f trên F sao cho sai số phân loại là thấp nhất.

Cho tập mẫu , thuộc vào hai lớp nhãn  là nhãn lớp tương ứng của các xi (-1 biểu thị lớp I, 1 biểu thị lớp II).

Ta có, phương trình siêu phẳng chứa vector  trong không gian:



Đặt 

Như vậy  biểu diễn sự phân lớp của  vào hai lớp như đã nêu.

Ta nói  nếu  thuộc lớp I và  nếu  thuộc lớp II.

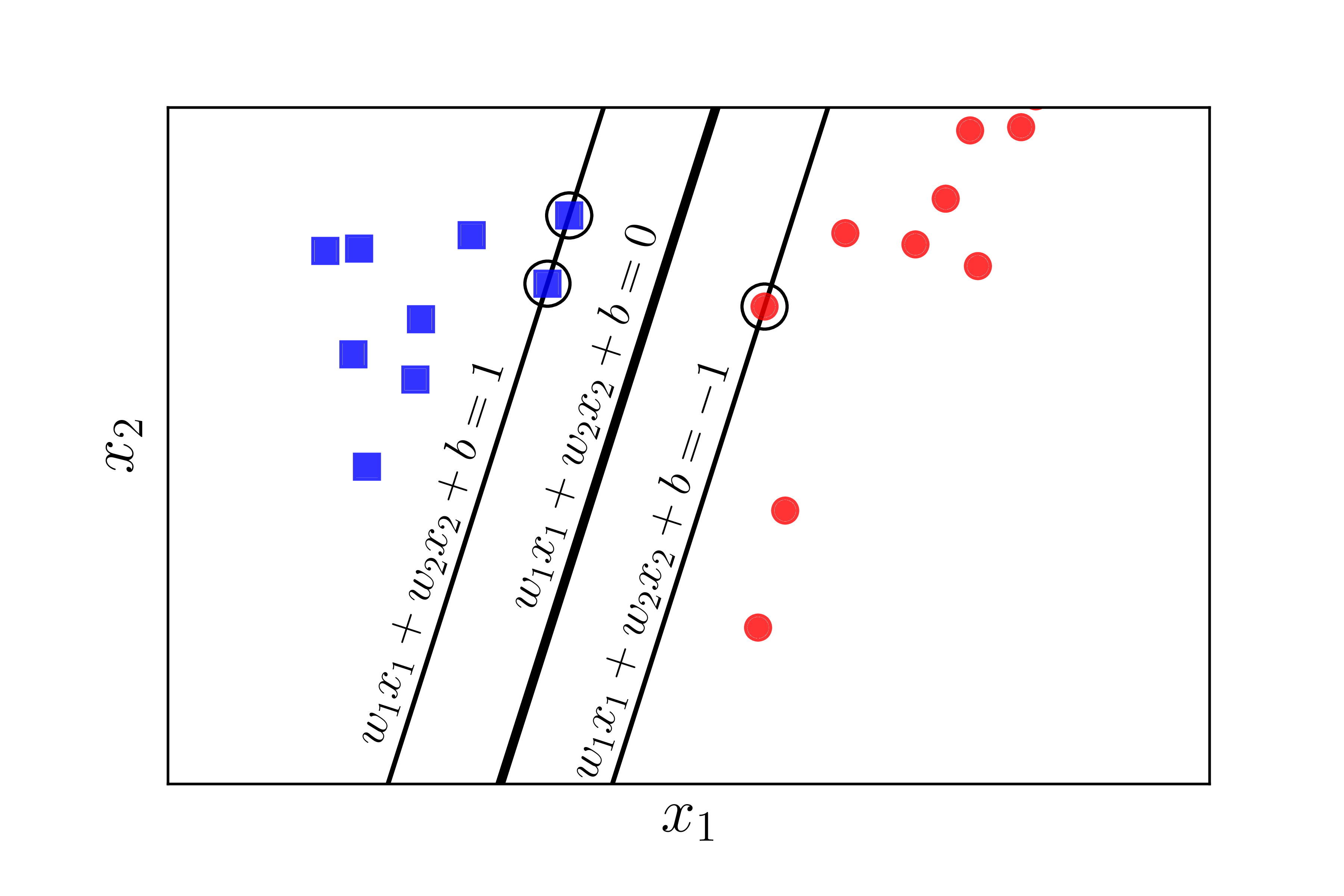
**1.3.3. Bài toán phân hai lớp với SVM:**

Bài toán đặt ra: Xác định hàm phân lớp các mẫu trong tương lai, nghĩa là với một mẫu dự liệu mới xi thì cần phải cách định xi được phân vào lớp +1 hay lớp -1.

Ta xét ba trường hợp, mỗi trường hợp sẽ có một bài toán tối ưu, giải bài toán tối ưu đó ta sẽ tìm được siêu phẳng cần tìm.

**1.3.3.1 Trường hợp 1:**

Tập D có thể phân chia tuyến tính được mà không có nhiễu (tất cả các điểm được gán nhãn +1 thuộc về phía dương của siêu phẳng, tất cả các điểm được gán nhã -1 thuộc về phía âm của siêu phẳng).

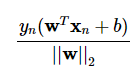


Hình3 tập dữ liệu được phân chia tuyến tính

Giả sử rằng các điểm vuông xanh thuộc class 1, các điểm tròn đỏ thuộc class -1 và mặt wTx+b=w1x1+w2x2+b=0 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.

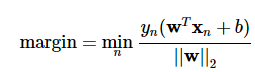
Ta quan sát thấy một điểm quan trọng sau đây: với cặp dữ liệu (xn,yn) bất kỳ, khoảng cách từ điểm đó tới mặt phân chia là:

Ta sẽ tìm siêu phẳng tách với  là vector trọng số,  là hệ số tự do, sao cho:

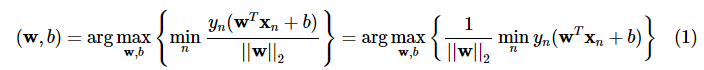


Điều này có thể dễ nhận thấy vì theo giả sử ở trên, yn luôn cùng dấu với phía của xn. Từ đó suy ra yn cùng dấu với (wTxn+b) và tử số luôn là 1 số không âm.

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:

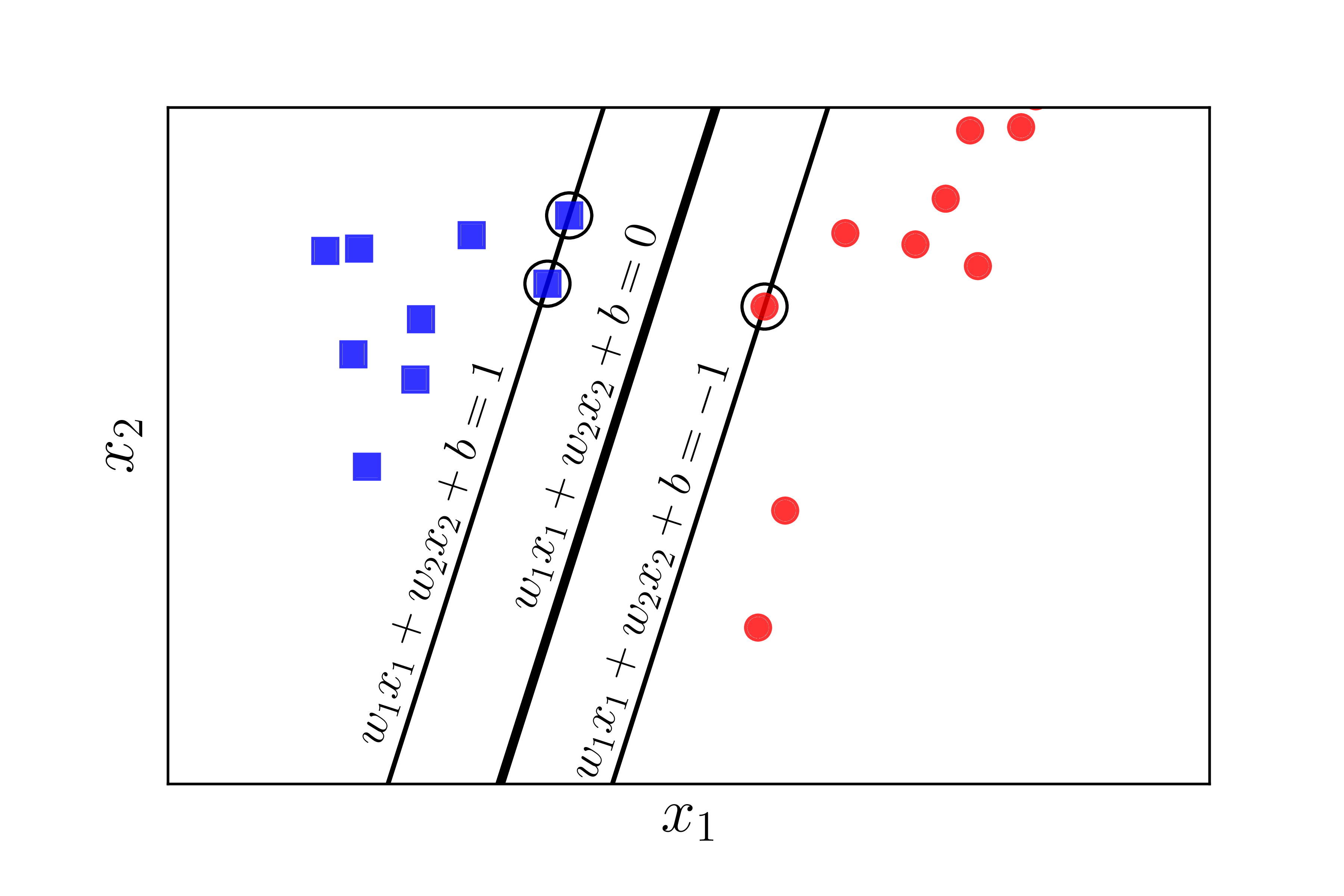


Việc giải trực tiếp bài toán này sẽ rất phức tạp, nhưng các bạn sẽ thấy có cách để đưa nó về bài toán đơn giản hơn.

Nhận xét quan trọng nhất là nếu ta thay vector hệ số w bởi kw và b bởi kb trong đó kk 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:

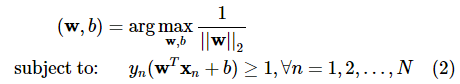


Hình 4 Các điểm gần mặt phân cách nhất của hai classes được khoanh tròn.

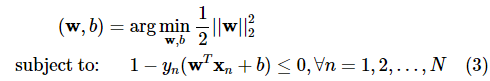
Như vậy, với mọi n, ta có:



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



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



Ở đây, chúng ta đã lấy nghịch đảo hàm mục tiêu, bình phương nó để được một hàm khả vi, và nhân với 1/2 để biểu thức đạo hàm đẹp hơn.

**Nhận xét:** Trong bài toán (3), [hàm mục tiêu là một norm, nên là một hàm lồi](https://machinelearningcoban.com/2017/03/12/convexity/#-norms). Các hàm bất đẳng thức ràng buộc là các hàm tuyến tính theo w và b, nên chúng cũng là các hàm lồi. Vậy bài toán tối ưu (3) có hàm mục tiêu là lồi, và các hàm ràng buộc cũng là lồi, nên nó là một bài toán lồi. Hơn nữa, nó là một [Quadratic Programming](https://machinelearningcoban.com/2017/03/19/convexopt/#-quadratic-programming). Thậm chí, hàm mục tiêu là *strictly convex* vì  và **I** là ma trận đơn vị - là một ma trận xác định dương. Từ đây có thể suy ra nghiệm cho SVM là *duy nhất*.

Đến đây thì bài toán này có thể giải được bằng các công cụ hỗ trợ tìm nghiệm cho Quadratic Programing, ví dụ [CVXOPT](https://machinelearningcoban.com/2017/03/19/convexopt/#-gioi-thieu-thu-vien-cvxopt).

Tuy nhiên, việc giải bài toán này trở nên phức tạp khi số chiều d của không gian dữ liệu và số điểm dữ liệu N tăng lên cao.

Người ta thường giải [bài toán đối ngẫu](https://machinelearningcoban.com/2017/04/02/duality/#-bai-toan-doi-ngau-lagrange-the-lagrange-dual-problem) của bài toán này. Thứ nhất, bài toán đối ngẫu có những tính chất thú vị hơn khiến nó được giải hiệu quả hơn. Thứ hai, trong quá trình xây dựng bài toán đối ngẫu, người ta thấy rằng SVM có thể được áp dụng cho những bài toán mà dữ liệu không *linearly separable*, tức các đường phân chia không phải là một mặt phẳng mà có thể là các mặt có hình thù phức tạp 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.

**1.3.3.1.1 Bài toán đối ngẫu cho SVM:**

Nhắc lại rằng bài toán tối ưu (3) là một bài toán lồi.

Chúng ta biết rằng: nếu một [bài toán lồi thoả mãn tiêu chuẩn Slater thì *strong duality* thoả mãn](https://machinelearningcoban.com/2017/04/02/duality/#-strong-duality-va-slaters-constraint-qualification). Và nếu *strong duality* thoả mãn thì nghiệm của bài toán chính là nghiệm của hệ [điều kiện KKT](https://machinelearningcoban.com/2017/04/02/duality/#-kkt-optimality-conditions).

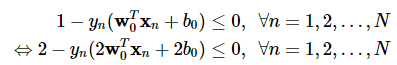
1. **Kiểm tra tiêu chuẩn Slater**

Bước tiếp theo, chúng ta sẽ chứng minh bài toán tối ưu (3) thoả mãn điều kiện Slater. Điều kiện Slater nói rằng, nếu tồn tại w,b thoả mãn:



thì *strong duality* thoả mãn.

Việc kiểm tra này tương đối đơn giản. Vì ta biết rằng luôn luôn có một (siêu) mặt phẳng phân chia hai classes nếu hai class đó là *linearly separable*, tức bài toán có nghiệm, nên *feasible set* của bài toán tối ưu (3) phải khác rỗng. Tức luôn luôn tồn tại cặp (w0,b0) sao cho:



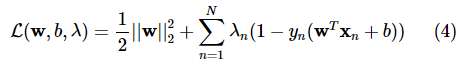
Vậy chỉ cần chọn w1=2w0 và b1=2b0, ta sẽ có:



Từ đó suy ra điều kiện Slater thoả mãn.

1. **Lagrangian của bài toán SVM**

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



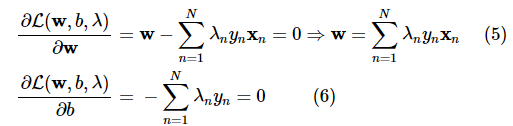
Với 

1. **Hàm đối ngẫu Lagrange**

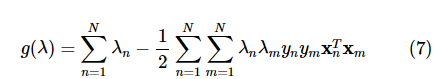
[Hàm đối ngẫu Lagrange](https://machinelearningcoban.com/2017/04/02/duality/#-ham-doi-ngau-lagrange-the-lagrange-dual-function) được định nghĩa là:

với  λ⪰0

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



Thay (5) và (6) vào (4) ta thu được g(λ) (*phần này tôi rút gọn, coi như một bài tập nhỏ cho bạn nào muốn hiểu sâu*):



**Đây là hàm số quan trọng nhất trong SVM**, các bạn sẽ thấy rõ hơn ở bài sau.

Xét ma trận:



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



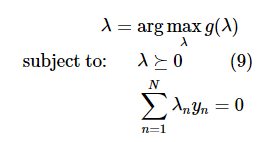
Với  ta có một quan sát quan trọng: KK là một [ma trận nửa xác định dương](https://machinelearningcoban.com/2017/03/12/convexity/#positive-semidefinite). Thật vậy, với mọi vector λ, ta có:



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

1. **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).

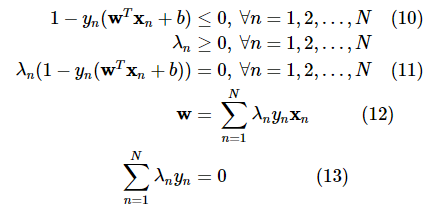
Đây là một bài toán lồi vì ta đang đi tìm giá trị lớn nhất của một hàm mục tiêu là *concave* trên một [*polyhedron*](https://machinelearningcoban.com/2017/03/12/convexity/#-giao-cua-cac-tap-loi-la-mot-tap-loi).

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. Nếu giải trực tiếp bằng các công cụ giải Quadratic Programming, có thể bài toán đối ngẫu còn phức tạp hơn (tốn thời gian hơn) so với bài toàn gốc.

1. **Điều kiện KKT**

Quay trở lại bài toán, vì đây là một bài toán lồi và *strong duality* thoả mãn, nghiệm của bài toán sẽ thoả mãn hệ [điều kiện KKT](https://machinelearningcoban.com/2017/04/02/duality/#-kkt-optimality-conditions) sau đây với biến số là w,b và λ:



Trong những điều kiện trên, điều kiện (11) là thú vị nhất. Từ đó ta có thể suy ra ngay, với n bất kỳ, hoặc hoặc . Trường hợp thứ hai chính là:



với chú ý rằng 

Những điểm thoả mãn (14) chính là những điểm nằm gần mặt phân chia nhất, là những điểm được khoanh tròn trong Hình 4 phía trên. Hai đường thẳng   *tựa* lên các điểm thoả mãn (14). Vậy nên những điểm (vectors) thoả mãn (14) còn được gọi là các *Support Vectors*. Và từ đó, cái tên *Support Vector Machine* ra đời.

Nhận xét, số lượng những điểm thoả mãn (14) thường chiếm số lượng rất nhỏ trong số N điểm. Chỉ cần dựa trên những *support vectors* này, chúng ta hoàn toàn có thể xác định được mặt phân cách cần tìm. Mặt khác, hầu hết các λn bằng 0. Vậy là mặc dù vector λ∈RN có số chiều có thể rất lớn, số lượng các phần tử khác 0 của nó rất ít. Nói cách khác, vector λ là một *sparse*vector. Support Vector Machine vì vậy còn được xếp vào *Sparse Models*. Các *Sparse Models* thường có cách giải hiệu quả (nhanh) hơn các mô hình tương tự với nghiệm là *dense* (hầu hết khác 0). Đây chính là lý do thứ hai của việc bài toán đối ngẫu SVM được quan tâm nhiều hơn là bài toán gốc.

Với những bài toán có số điểm dữ liệu N nhỏ, ta có thể giải hệ điều kiện KKT phía trên bằng cách xét các trường hợp λn=0 hoặc λn≠0. Tổng số trường hợp phải xét là 2N. Với N>50, đây là một con số rất lớn, giải bằng cách này sẽ không khả thi.

Sau khi tìm được λ từ bài toán (9), ta có thể suy ra được w dựa vào (12) và b dựa vào (11) và (13). Rõ ràng ta chỉ cần quan tâm tới λn≠0.

Gọi tập hợp  và NS là số phần tử của tập S. Với mỗi n∈S, ta có:



Mặc dù từ chỉ một cặp (xn,yn), ta có thể suy ra ngay được bb nếu đã biết w, tuy nhiên để tính b thường được sử dụng là trung bình cộng của mọi cách tính b:



Ta có:  theo (12)

Suy ra để xác định một điểm xx mới thuộc vào class nào, ta cần xác định dấu của biểu thức:



Biểu thức này phụ thuộc vào cách tính tích vô hướng giữa các cặp vector x và từng xn∈S.

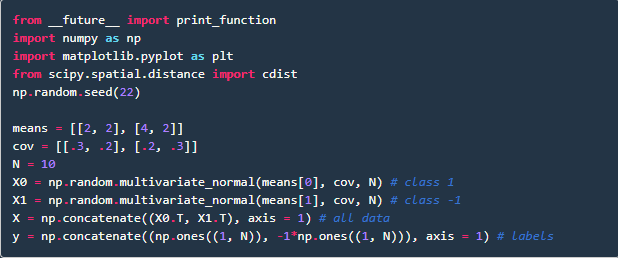
**1.3.3.1.2 Lập trình nghiệm cho SVM**

Trong mục này, chúng ta sẽ tìm hiểu hai cách tính nghiệm cho SVM

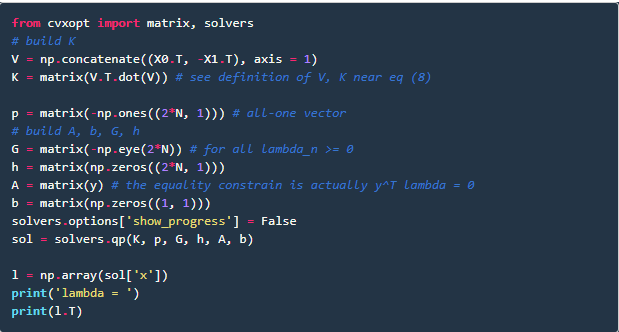
* Cách 1: Dựa theo bài toán (9) và các công thức (15) và (16)
* Cách 2: Sử dụng trực tiếp thư viện sklearn

**Cách 1: Tìm nghiệm theo công thức**

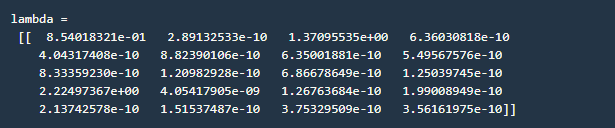
Trước tiên chúng ta gọi các *modules* cần dùng và tạo dữ liệu giả (dữ liệu này chính là dữ liệu tôi dùng trong các hình phía trên nên chúng ta biết chắc rằng hai classes là *linearly separable*):



Tiếp theo, chúng ta giải bài toán (9) bằng CVXOPT:

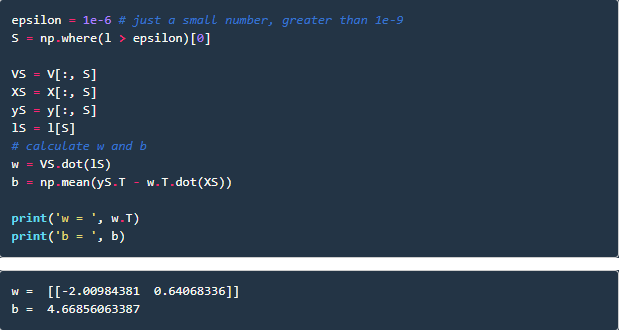


Kết quả:

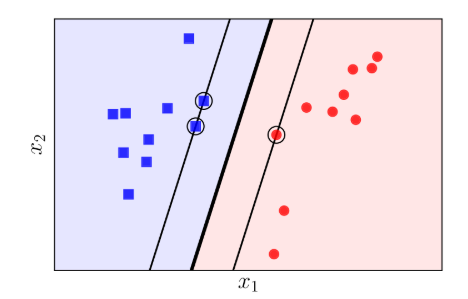


Ta nhận thấy rằng hầu hết các giá trị của lambda đều rất nhỏ, tới 10−9 hoặc 10−10. Đây chính là các giá trị bằng 0 nhưng vì sai số tính toán nên nó khác 0 một chút. Chỉ có 3 giá trị khác 0, ta dự đoán là sẽ có 3 điểm là *support vectors*.

Ta đi tìm *support set* S rồi tìm nghiệm của bài toán:



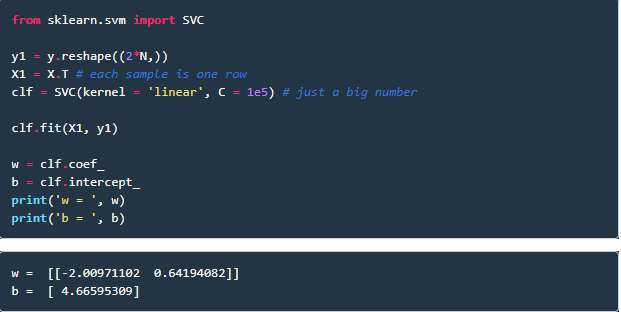
Minh hoạ kết quả:



Hình 5 Minh họa kết quả tìm được bởi SVM

**Cách 2: Tìm nghiệm theo thư viện**

Chúng ta sẽ sử dụng hàm [sklearn.svm.SVC](http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html) ở đây. Các bài toán thực tế thường sử dụng thư viện [libsvm](https://www.csie.ntu.edu.tw/~cjlin/libsvm/) được viết trên ngôn ngữ C, có API cho Python và Matlab.

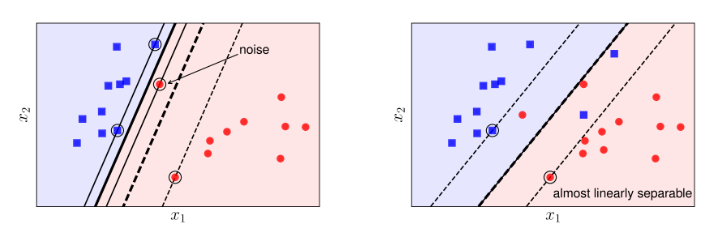


Kết quả này khá giống với kết quả chúng ta tìm được ở phần trên.

**1.3.3.2 Trường hợp 2:**

Tập D có thể phân chia tuyến tính được nhưng có nhiễu.

Trong trường hợp này, hầu hết các điểm đều được phân chia đúng bở siêu phẳng. Tuy nhiên có 1 số điểm bị nhiễu, nghĩa là: điểm có nhãn dương nhưng lại thuộc về phía âm của siêu phẳng, điểm có nhãn âm nhưng lại thuộc về phía dương của siêu phẳng.



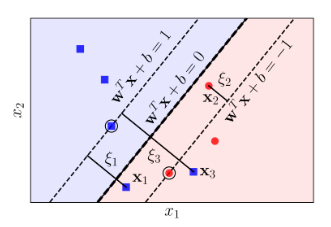
Hình a Hình b

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).

**1.3.3.2.1 Phân tích toán học:**

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 ||w||2. Để xác định sự hy sinh, chúng ta cùng theo dõi Hình 2 dưới đây:



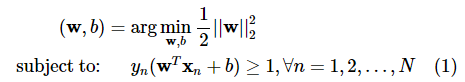
Hình 2 Giới thiệu các biến slack ξn

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.

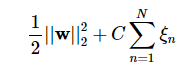
Với mỗi điểm xnxn trong tập toàn bộ dữ liệu huấn luyện, ta giới thiệu thêm một biến đo sự hy sinh ξnξn tương ứng. Biến này còn được gọi là slack variable. Với những điểm xnxn nằm trong vùng an toàn, ξn=0. Với mỗi điểm nằm trong vùng không an toàn như x1,x2 hay x3 ta có ξi>0. Nhận thấy rằng nếu yi=±1là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



Nhắc lại bài toán tối ưu cho Hard Margin SVM:



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 

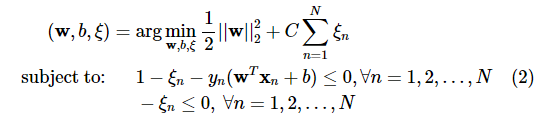
Hằng số C được dùng để điều chỉnh tầm quan trọng giữa margin và sự hy sinh. Hằng số này được xác định từ trước bởi người lập trình hoặc có thể được xác định bởi cross-validation.

Điều kiện ràng buộc sẽ thay đổi một chút. Với mỗi cặp dữ liệu (xn,yn), thay vì ràng buộc cứng yn(wTxn+b)≥1yn(wTxn+b)≥1, 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*:



Nhận xét:

* Nếu CC nhỏ, việc *sự hy sinh* cao hay thấp không gây ảnh hưởng nhiều tới giá trị của hàm mục tiêu, thuật toán sẽ điều chỉnh sao cho  là nhỏ nhất, tức *margin* là lớn nhất, điều này sẽ dẫn tới  sẽ lớn theo. Ngược lại, nếu C quá lớn, để hàm mục tiêu đạt giá trị nhỏ nhất, thuật toán sẽ tập trung vào làm giảm . Trong trường hợp CC rất rất lớn và hai classes là *linearly separable*, ta sẽ thu được . Chú ý rằng giá trị này không thể nhỏ hơn 0. Điều này đồng nghĩa với việc không có điểm nào phải *hy sinh*, tức ta thu được nghiệm cho *Hard Margin SVM*. Nói cách khác, *Hard Margin SVM* chính là một trường hợp đặc biệt của *Soft Margin SVM*.
* Bài toán tối ưu (2) có thêm sự xuất hiện của *slack variables* ξn. Những ξn=0 tương ứng với những điểm dữ liệu nằm trong *vùng an toàn*. Những 0<ξn≤1 tương ứng với những điểm nằm trong *vùng không an toàn* những vẫn được phân loại đúng, tức vẫn nằm về đúng phía so với đường phân chia. Những ξn>1 tương ứng với các điểm bị phân loại sai.
* Hàm mục tiêu trong bài toán tối ưu (2) là một hàm lồi vì nó là tổng của hai hàm lồi: hàm norm và hàm tuyến tính. Các hàm ràng buộc cũng là các hàm tuyến tính theo (w,b,ξ). Vì vậy bải toán tối ưu (2) là một bài toán lồi, hơn nữa nó có thể biểu diễn dưới dạng một [Quadratic Programming (QP)](https://machinelearningcoban.com/2017/03/19/convexopt/#-quadratic-programming).

**1.3.3.2.2 Bài toán đối ngẫu Lagrange**

Trước kết, ta cần kiểm tra [tiêu chuẩn Slater](https://machinelearningcoban.com/2017/04/02/duality/#-strong-duality-va-slaters-constraint-qualification) cho bài toán tối ưu lồi (2)(2). Nếu tiêu chuẩn này được thoả mãn, *strong duality* sẽ thoả mãn, và ta sẽ có nghiệm của bài toán tối ưu (2)(2) là nghiệm của [hệ điều kiện KKT](https://machinelearningcoban.com/2017/04/02/duality/#-kkt-optimality-conditions).

1. **Kiểm tra tiêu chuẩn Slater**

Rõ ràng là với mọi n=1,2,…,N và mọi (w,b)(w,b), ta luôn có thể tìm được các số **dương** ξn,n=1,2,…,N đủ lớn sao cho:



Vậy nên bài toán này thoả mãn tiêu chuẩn Slater.

1. **Lagrangian của bài toán Soft-margin SVM**

Lagrangian cho bài toán (2) là:



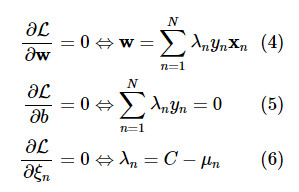
Với  là các biến đối ngẫu Lagrange (vector nhân tử Lagrange).

1. **Bài toán đối ngẫu**

Hàm số đối ngẫu của bài toán tối ưu (2) là:



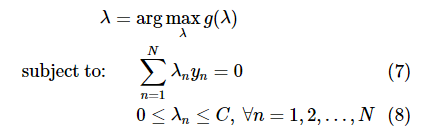
Với mỗi cặp (λ,μ), chúng ta sẽ quan tâm tới (w,b,ξ) thoả mãn điều kiện đạo hàm của Lagrangian bằng 0:



Từ (6) ta thấy rằng ta chỉ quan tâm tới những cặp (λ,μ)(λ,μ) sao cho . Từ đây ta cũng suy ra Thay các biểu thức này vào Lagrangian ta sẽ thu được hàm đối ngẫu:



Chú ý rằng hàm này không phụ thuộc vào μμ nhưng ta cần lưu ý ràng buộc (6), ràng buộc này và điều kiện không âm của λ có thể được viết gọn lại thành 0≤λn≤C, và ta đã giảm được biến μ. Lúc này, bài toán đối ngẫu được xác định bới:



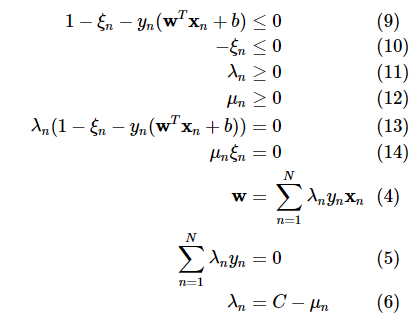
Bài toán này gần giống với [bài toán đối ngẫu của *Hard Margin SVM*](https://machinelearningcoban.com/2017/04/09/smv/#-ham-doi-ngau-lagrange), chỉ khác là ta có chặn trên cho mỗi λnλn. Khi C rất lớn, ta có thể coi hai bài toán là như nhau. Ràng buộc (8) còn được gọi là *box constraint* vì không gian các điểm λλ thoả mãn ràng buộc này giống như một hình hộp chữ nhật trong không gian nhiều chiều.

Bài toán này cũng hoàn toàn giải được bằng các công cụ giải QP thông thường, ví dụ CVXOPT như tôi đã thực hiện trong bài *Hard Margin SVM*.

Sau khi tìm được λ của bài toán đối ngẫu, ta vẫn phải quay lại tìm nghiệm (w,b,ξ) của bài toán gốc. Để làm điều này, chúng ta cùng xem xét hệ điều kiện KKT.

1. **Hệ điều kiện KKT**

[Hệ điều kiện KKT](https://machinelearningcoban.com/2017/04/02/duality/#-kkt-optimality-conditions) của bài toán tối ưu Soft Margin SVM là, với mọi n=1,2,…,N:



Nhận xét:

* Nếu λn=0 thì từ (6) ta suy ra μn=C≠0. Kết hợp với (14) ta suy ra ξn=0. Nói cách khác, không có *sự hy sinh* nào xảy ra ở xn, tức xn nằm trong vùng an toàn.
* Nếu λn>0, từ (13) ta có:

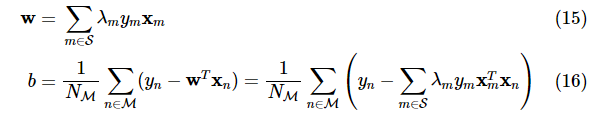


* Nếu 0<λn<C, từ (6) ta suy ra μn≠0 và từ (14) ta lại có ξn=0. Nói cách khác, yn(wTxn+b)=1, hay những điểm xn nằm chính xác trên margin.
* Nếu λn=C, thì μn=0 và ξn có thể nhận bất kỳ giá trị nào không âm. Nếu ξn≤1,xn sẽ được phân lớp đúng (vẫn đúng phía so với đường phân chia). Ngược lại, các điểm tương ứng với ξn>1 sẽ bị phân lớp sai.
* Nếu λn=C, thì μn=0 và ξn có thể nhận bất kỳ giá trị nào không âm. Nếu ξn≤1,xn sẽ được phân lớp đúng (vẫn đúng phía so với đường phân chia). Ngược lại, các điểm tương ứng với ξn>1 sẽ bị phân lớp sai.

Ngoài ra, những điểm tương ứng với 0<λn≤C bây giờ là sẽ là các *support vectors*. Mặc dù những điểm này có thể không nằm trên *margins*, chúng vẫn được coi là *support vectors* vì có công đóng góp cho việc tính toán w thông qua phương trình (4).

Như vậy, dựa trên các giá trị của λn ta có thể dự đoán được vị trí tương đối của xnxn so với hai *margins*.

Đặt  Tức M là tập hợp các chỉ số của các điểm nằm chính xác trên *margins* - hỗ trợ cho việc tính b, S là tập hợp các chỉ số của các *support vectors* - hỗ trợ trực tiếp cho việc tính w. Tương tự như với Hard Margin SVM, các hệ số w,b có thể được xác định bởi:



Nhắc lại rằng mục đích cuối cùng là xác định nhãn cho một điểm mới chứ không phải là tính w và b nên ta quan tâm hơn tới cách xác định giá trị của biếu thức sau với một điểm dữ liệu x bất kỳ:



Và trong cách tính này, ta chỉ cần quan tâm tới tích vô hướng của hai điểm bất kỳ.

**1.3.3.2.3 Bài toán tối ưu không ràng buộc cho Soft Margin SVM**

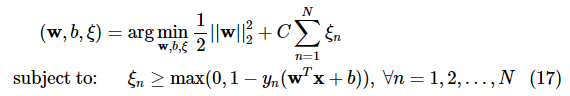
Trong mục này, chúng ta sẽ đưa bài toán tối ưu có ràng buộc (2)(2) về một bài toán tối ưu không ràng buộc, và có có khả năng giải được bằng các phương pháp Gradient Descent.

1. **Bài toán tối ưu không ràng buộc tương đương**

Để ý thấy rằng điều kiện ràng buộc thứ nhất:



Kết hợp với điều kiện ξn≥0 ta sẽ thu được bài toán ràng buộc tương đương với bài toán (2) như sau:



Tiếp theo, để đưa bài toán (17) về dạng không ràng buộc, chúng ta sẽ chứng minh nhận xét sau đây bằng phương pháp phản chứng:

Nếu (w,b,ξ) là nghiệm của bài toán tối ưu (17)(17), tức tại đó hàm mục tiêu đạt giá trị nhỏ nhất, thì:



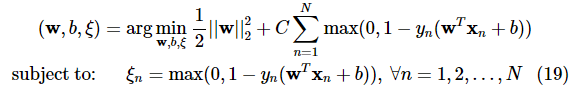
Thật vậy, giả sử ngược lại, tồn tại n sao cho:



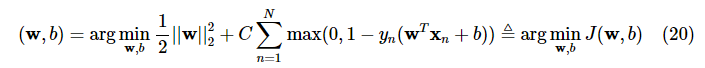
ta chọn , ta sẽ thu được một giá trị thấp hơn của hàm mục tiêu, trong khi tất cả các ràng buộc vẫn được thoả mãn. Điều này mâu thuẫn với việc hàm mục tiêu đã đạt giá trị nhỏ nhất.

Vậy nhận xét (18)(18) được chứng minh.

Khi đó, ta thay toàn bộ các giá trị của ξnξn trong (18)(18) vào hàm mục tiêu:



Rõ ràng rằng biến số ξ không còn quan trọng trong bài toán này nữa, ta có thể lược bỏ nó mà không làm thay đổi nghiệm của bài toán:



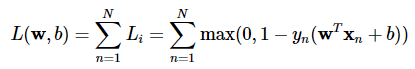
Đây là một bài toán tối ưu không ràng buộc với hàm mất mát J(w,b). Bài toán này có thể giải được bằng các phương pháp Gradient Descent. Nhưng trước hết, chúng ta cùng xem xét hàm mất mát này từ một góc nhìn khác, bằng định nghĩa của một hàm gọi là *hinge loss.*

1. **Xây dựng hàm mất mát**

Bây giờ, nếu ta xem xét bài toán Soft Margin SVM dưới góc nhìn hinge loss:

Với mỗi cặp (w,b)(w,b), đặt:  


Lấy tổng tất cả các *loss* này (giống như cách mà Logistic Regression hay Softmax Regression lấy tổng của tất cả các cross entropy loss) theo n ta được:

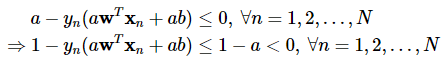


Câu hỏi đặt ra là, nếu ta trực tiếp tối ưu tổng các hinge loss này thì điều gì sẽ xảy ra?

Trong trường hợp dữ liệu trong hai class là *linearly separable*, ta sẽ có giá trị tối ưu tìm được của L(w,b) là bằng 0. Điều này có nghĩa là:



Nhân cả hai về với một hằng số a>1 ta có:

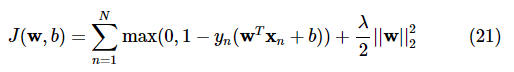


Điều này nghĩa là (aw,ab) cũng là nghiệm của bài toán. Nếu không có điều kiện gì thêm, bài toán có thể dẫn tới nghiệm không ổn định vì các hệ số của nghiệm có thể lớn tuỳ ý.

Để tránh *bug* này, chúng ta cần thêm một số hạng nữa vào L(w,b) gọi là số hạng [*regularization*](https://machinelearningcoban.com/2017/03/04/overfitting/#-regularization), giống như cách chúng ta đã làm để tránh *overfitting* trong neural networks. Lúc này, ta sẽ có hàm mất mát tổng cộng là:



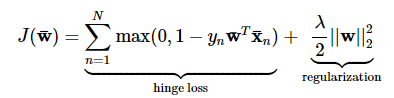
với λ là một số dương, gọi là *regularization parameter*, hàm R() sẽ giúp hạn chế việc các hệ số (w,b)(w,b)trở nên quá lớn. Có nhiều cách chọn hàm R(), nhưng cách phổ biến nhất là l2, khi đó hàm mất mát của Soft Margin SVM sẽ là:



Kỹ thuật này còn gọi là weight decay. Chú ý rằng weight decay thường không được áp dụng lên thành phần bias b.

Ta thấy rằng hàm mất mát (21) giống với hàm mất mát (20) với λ=1/C. Ở đây, tôi đã lấy λ/2 để biểu thức đạo hàm được *đẹp hơn*.

Nhận thấy rằng ta có thể khiến biểu thức (19) gọn hơn một chút bằng cách sử dụng *bias trick* như đã làm trong Linear Regression hay các bài về neurel networks. Bằng cách *mở rộng* thêm một thành phần bằng 1 vào các điểm dữ liệu xn∈Rdđể được xn∈Rd+1và kết hợp w,b thành một vector  ta sẽ có một biểu thức gọn hơn. Khi đó, hàm mất mát trong (21) có thể được viết gọn thành:



1. **Tối ưu hàm mất mát**

Trước hết ta cần tính được đạo hàm của hàm mất mát theo . Việc này thoáng qua có vẻ hơi phức tạp vì ta cần tính đạo hàm của hàm maxmax, nhưng nếu chúng ta nhìn vào đạo hàm của hinge loss, ta có thể tính được đạo hàm theo  một cách đơn giản.

Chúng ta tạm quên đi đạo hàm của phần regularization vì nó đơn giản bằng  với thành phần 0 ở cuối chính là đạo hàm theo bias của thành phần regularization.

Với phần hinge loss, xét từng điểm dữ liệu, ta có hai trường hợp:

* TH1: Nếu , ta có ngay đạo hàm theo  bằng 0.
* TH2: Nếu , ngay đạo hàm theo  bằng 0

Để tính gradient cho toàn bộ dữ liệu, chúng ta cần một chút kỹ năng biến đổi đại số tuyến tính.

Đặt: 

với chú ý rằng u là một vector hàng.

Tiếp tục, ta cần xác định các vị trí của uu có giá trị nhỏ hơn 1, tức ứng với TH2 ở trên. Bằng cách đặt:

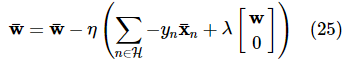


ta có thể suy ra cách tính đạo hàm theo của hàm mất mát là



Các bạn sẽ thấy cách tính toán giá trị này một cách hiệu quả trong phần lập trình.

Vậy quy tắc cập nhật của sẽ là:



với η là *learning rate*.

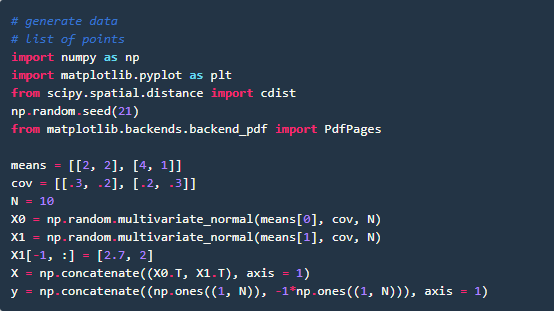
Với các bài toán large-scale, ta có thể sử dụng phương pháp Mini-batch Gradient Descent để tối ưu. Đây chính là một ưu điểm của hướng tiếp cận theo hinge loss.

**1.3.3.2.4 Kiểm chứng bằng lập trình**

Trong mục này, chúng ta cùng làm hai thí nghiệm nhỏ. Thứ nghiệm thứ nhất sẽ đi tìm nghiệm của một bài toán Soft Margin SVM bằng ba cách khác nhau: Sử dụng thư viện sklearn, Giải bài toán đối ngẫu bằng CVXOPT, và Tối ưu hàm mất mát không ràng bằng phương pháp Gradient Descent. Nếu mọi tính toán ở trên là chính xác, nghiệm của ba cách làm này sẽ giống nhau, khác nhau có thể một chút bởi sai số trong tính toán. Ở thí nghiệm thứ hai, chúng ta sẽ thay CC bởi những giá trị khác nhau và cùng xem các margin thay đổi như thế nào.

1. **Giải bài toán Soft Margin bằng 3 cách khác nhau**

Khai báo thư viện và tạo dữ liệu giả



* Giải bài toán bằng thư viện sklearn

Ta chọn C=100 trong thí nghiệm này:

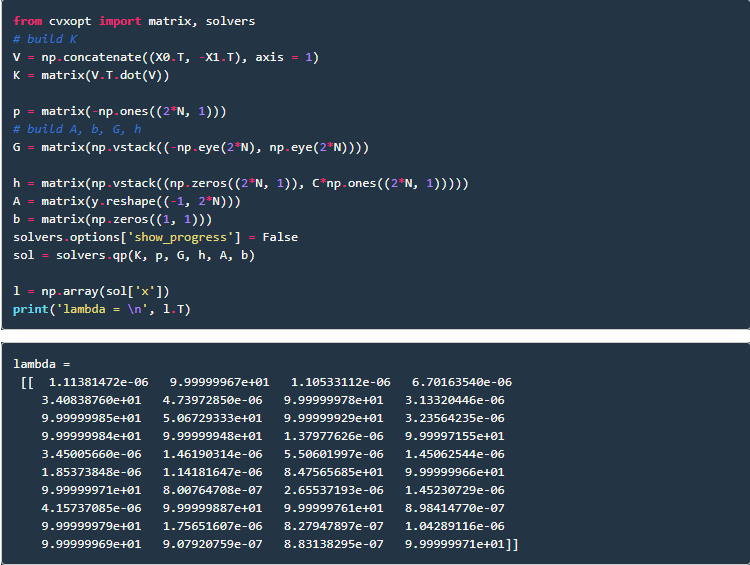


Nghiệm tìm được:



* Tìm nghiệm bằng giải bài toán đối ngẫu

Tương tự như việc giải bài toán Hard Margin SVM, chỉ khác rằng ta có thêm ràng buộc về chặn trên của các nhân thử Lagrange:



Trong các thành phần của lambda tìm được, có rất nhiều thành phần nhỏ tới 1e-6 hay 1e-7. Đây chính là các lambda\_i = 0. Có rất nhiều phần tử xấp xỉ 9.99e+01, đây chính là các lambda\_i bằng với C = 100, tương ứng với các support vectors không nằm trên margins, các sai số nhỏ xảy ra do tính toán. Các giá trị còn lại nằm giữa 0 và 100 là các giá trị tương ứng với các điểm nằm chính xác trên hai margins.

Tiếp theo, ta cần tính w và b theo công thức (15) và (16) Trước đó ta cần tìm tập hợp các điểm support và những điểm nằm trên margins.

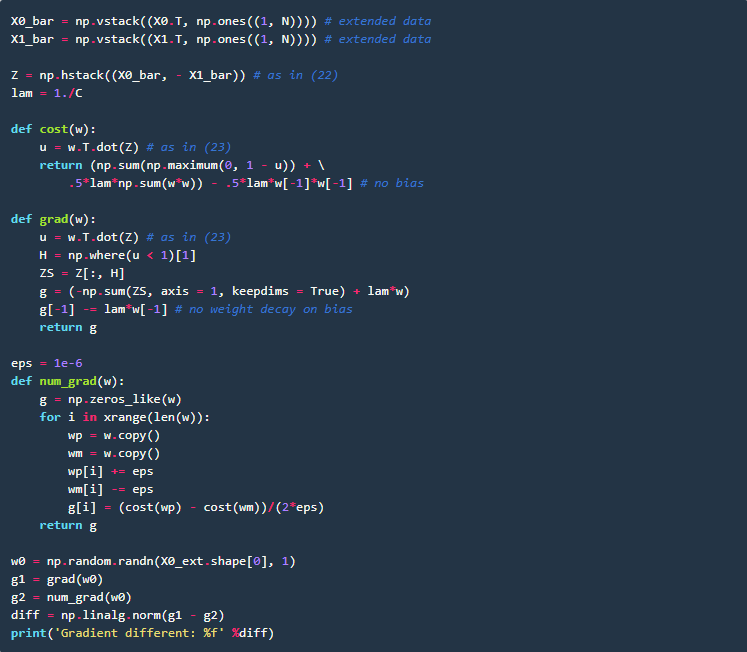
Kết quả:



* Tìm nghiệm bằng giải bài toán không ràng buộc

Trong phương pháp này, chúng ta cần tính gradient của hàm mất mát. Như thường lệ, chúng ta cần kiểm chứng này bằng cách so sánh với *numerical gradient*.

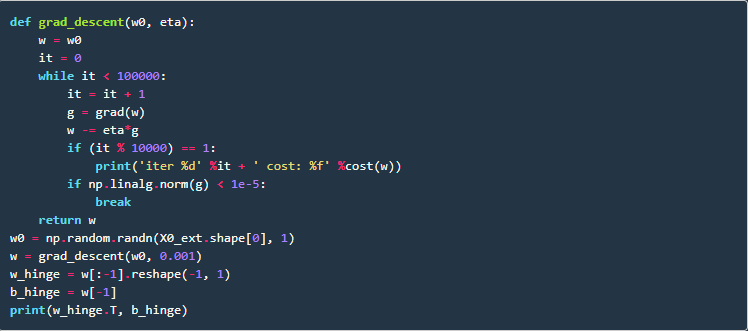
Chú ý rằng trong phương pháp này, ta cần dùng tham số lam = 1/C.





Vì sự khác nhau giữa hai cách tính gradient là bằng 0, ta có thể yên tâm rằng gradient tính được là chính xác.

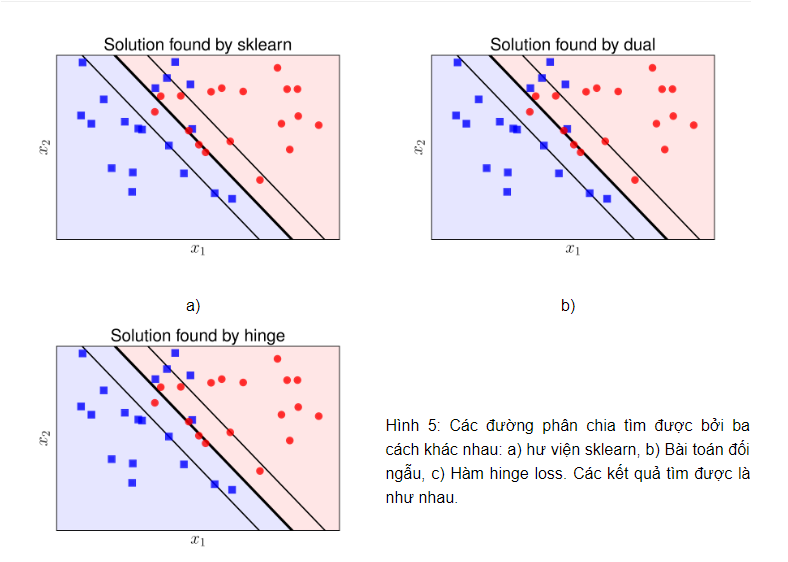
Sau khi chắc chắn rằng gradient tìm được đã chính xác, ta có thể bắt đầu làm Gradient Descent:



Kết quả:

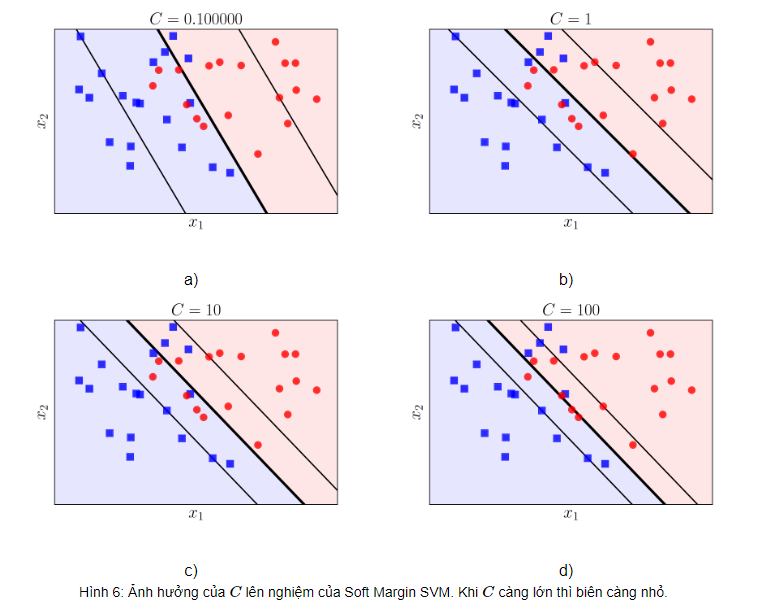


Ta thấy rằng kết quả tìm được bằng ba cách là như nhau. Hình 5 dưới đây minh hoạ kết quả bằng ba cách tính:



1. Ảnh hưởng của C lên nghiệm

Hình 6 dưới đây minh hoạ nghiệm tìm được cho bài toán phía trên nhưng với các giá trị C khác nhau. Nghiệm được tìm bằng thư viện sklearn.



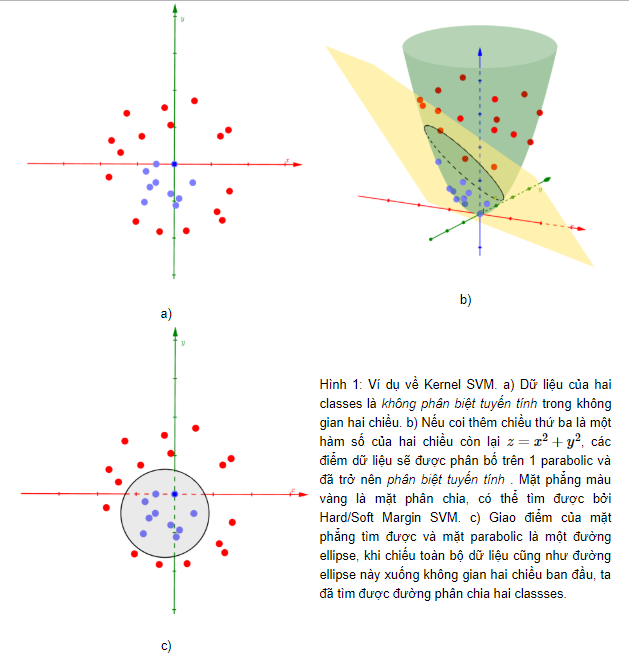
Chúng ta nhận thấy rằng khi (C) càng lớn thì biên càng nhỏ đi. Điều này phù hợp với suy luận của chúng ta ở trên.

**1.3.3.3 Trường hợp 3:**

Tập D không thể phân chia 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 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:



Hình Ví dụ về Kernel SVM

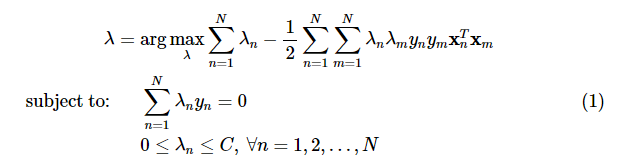
Nói một cách ngắn gọn, Kernel SVM là việc đi tìm một hàm số biến đổi dữ liệu x từ không gian *feature*ban đầu thành dữ liệu trong một không gian mới bằng hàm số Φ(x). Trong ví dụ này, hàm Φ() đơn giản là giới thiệu thêm một chiều dữ liệu mới (một feature mới) là một hàm số của các *features* đã biết. 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.

Nếu phải so sánh, ta có thể thấy rằng hàm biến đổi Φ() tương tự như [*activation functions*](https://machinelearningcoban.com/2017/02/24/mlp/#-activation-functions) trong Neural Networks. Tuy nhiên, có một điểm khác biệt ở đây là: trong khi nhiệm vụ của activation function là phá vỡ tính tuyến tính của *mô hình*, hàm biến đổi Φ() đi biến *dữ liệu* không phân biệt tuyến tính thành phân biệt tuyến tính. Như vậy là để đạt được mục đích chung, ta có hai cách nhìn khác nhau về cách giải quyết.

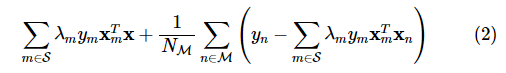
Các hàm Φ() thường tạo ra dữ liệu mới có số chiều cao hơn số chiều của dữ liệu ban đầu, thậm chí là vô hạn chiều. Nếu tính toán các hàm này trực tiếp, chắc chắn chúng ta sẽ gặp các vấn đề về bộ nhớ và hiệu năng tính toán. Có một cách tiếp cận là sử dụng các *kernel functions* mô tả quan hệ giữa hai điểm dữ liệu bất kỳ trong không gian mới, thay vì đi tính toán trực tiếp từng điểm dữ liệu trong không gian mới. Kỹ thuật này được xây dựng dựa trên quan sát về [bài toán đối ngẫu của SVM](https://machinelearningcoban.com/2017/04/09/smv/#-bai-toan-doi-ngau-cho-svm).

**1.3.3.3.1 Cơ sở toán học:**

 Bài toán đối ngẫu trong Soft Margin SVM cho dữ liệu *gần phân biệt tuyến tính*:



Sau khi giải được λ cho bài toán (1), *nhãn* của một điểm dữ liệu mới sẽ được xác định bởi dấu của biểu thức:

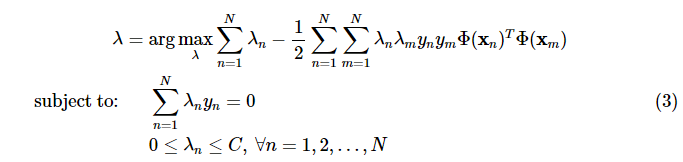


Trong đó:

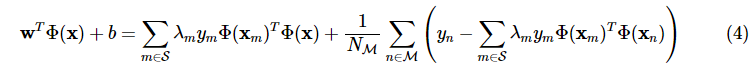
* là tập hợp những điểm nằm trên margin.
* là tập hợp các điểm support.
* là số phần tử của M.

Với dữ liệu thực tế, rất khó để có dữ liệu *gần phân biệt tuyến tính*, vì vậy nghiệm của bài toán (1) có thể không thực sự tạo ra một bộ phân lớp tốt. Giả sử rằng ta có thể tìm được hàm số Φ() sao cho sau khi được biến đổi sang không gian mới, mỗi điểm dữ liệu xx trở thành Φ(x), và trong không gian mới này, dữ liệu trở nên *gần phân biệt tuyến tính*.

Trong không gian mới, bài toán (1) trở thành:



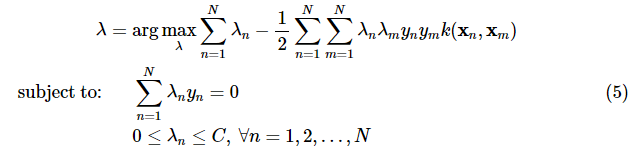
và *nhãn* của một điểm dữ liệu mới được xác định bởi dấu của biểu thức:

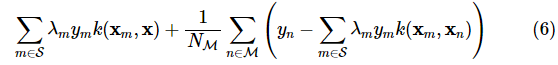


Như đã nói ở trên, việc tính toán trực tiếp Φ(x) cho mỗi điểm dữ liệu có thể sẽ tốn rất nhiều bộ nhớ và thời gian vì số chiều của Φ(x) thường là rất lớn, có thể là vô hạn! Thêm nữa, để tìm *nhãn* của một điểm dữ liệu mới xx, ta lại phải tìm biến đổi của nó Φ(x) trong không gian mới rồi lấy tích vô hướng của nó với tất cả các Φ(xm) với mm trong tập hợp support. Để tránh việc này, ta quan sát thấy một điều thú vị sau đây.

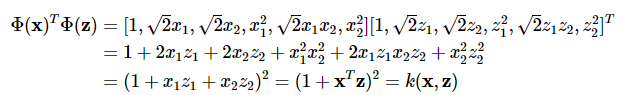
Trong bài toán (3) và biểu thức (4), chúng ta không cần tính trực tiếp Φ(x) cho mọi điểm dữ liệu. Chúng ta chỉ cần tính được Φ(x)TΦ(z) dựa trên hai điểm dữ liệu x,z bất kỳ. Kỹ thuật này còn được gọi là **kernel trick**. Những phương pháp dựa trên kỹ thuật này, tức thay vì trực tiếp tính tọa độ của một điểm trong không gian mới, ta đi tính tích vô hướng giữa hai điểm trong không gian mới, được gọi chung là **kerrnel method**.

Lúc này, bằng cách định nghĩa *hàm kernel* , ta có thể viết lại bài toán (3) và biểu thức (4) như sau:



Và: 

**Ví dụ:** Xét phép biến đổi 1 điểm dữ liệu trong không gian hai chiều thành một điểm trong không gian 5 chiều 

Ta có: 

Trong ví dụ này, rõ ràng rằng việc tính toán hàm kernel k() cho hai điểm dữ liệu dễ dàng hơn việc tính từng Φ() rồi nhân chúng với nhau.

**1.3.3.3.2 Hàm số kernel:**

**a) Tính chất của các hàm kerrnel**

Không phải hàm k() bất kỳ nào cũng được sử dụng. Các hàm kerrnel cần có các tính chất:

* Đối xứng: k(x,z)=k(z,x) Điều này dễ nhận ra vì tích vô hướng của hai vector có tính đối xứng.
* *Về lý thuyết*, hàm kerrnel cần thỏa mãn [điều kiện Mercer](https://en.wikipedia.org/wiki/Mercer%27s_theorem#Mercer.27s_condition):



Tính chất này để đảm bảo cho việc hàm mục tiêu của bài toán đối ngẫu (5) là *lồi*.

* *Trong thực hành*, có một vài hàm số k() không thỏa mãn điều kiện Merrcer nhưng vẫn cho kết quả chấp nhận được. Những hàm số này vẫn được gọi là kernel. Trong bài viết này, tôi chỉ tập trung vào các hàm kernel thông dụng và có sẵn trong các thư viện.

Nếu một hàm kerrnel thỏa mãn điều kiện (7)(7), xét , ta sẽ có:



với K là một ma trận đối xứng mà phần tử ở hàng thứ n cột thứ m của nó được định nghĩa bởi:



Từ (8) ta suy ra K là một ma trận nửa xác định dương. Vì vậy, bài toán tối ưu (5) có ràng buộc là lồi và hàm mục tiêu là một hàm lồi (một quadratic form). Vì vậy chúng ta có thể giải quyết bài toán này một cách hiệu quả.

Trong bài viết này, tôi sẽ không đi sâu vào việc giải quyết bài toán (5) vì nó hoàn toàn tương tự như bài toán đối ngẫu của Soft Margin SVM. Thay vào đó, tôi sẽ trình bày các hàm kernel thông dụng và hiệu năng của chúng trong các bài toán thực tế. Việc này sẽ được thực hiện thông qua các ví dụ và cách sử dụng thư viện sklearn.

**b) Một số hàm kernel thông dụng**

* **Linear**

Đây là trường hợp đơn giản với kernel chính tích vô hướng của hai vector:



Hàm số này,  thỏa mãn điều kiện (7).

Khi sử dụng hàm sklearn.svm.SVC, kernel này được chọn bằng cách đặt kernel = 'linear'

* **Polynomial**



Với dd là một số dương để chỉ bậc của đa thức. dd có thể không là số tự nhiên vì mục đích chính của ta không phải là bậc của đa thức mà là cách tính kernel. Polynomial kernel có thể dùng để mô tả hầu hết các đa thức có bậc không vượt quá dd nếu dd là một số tự nhiên.

Khi sử dụng thư viện sklearn, kerrnel này được chọn bằng cách đặt kernel = 'poly'.

* **Radial Basic Function**

Radial Basic Function (RBF) kernel hay Gaussian kernel được sử dụng nhiều nhất trong thực tế, và là lựa chọn mặc định trong sklearn. Nó được định nghĩa bởi:



Trong sklearn, kernel = 'rbf'.

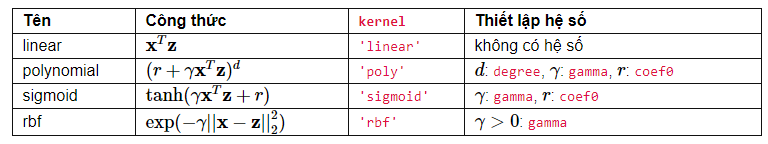
* **Sigmoid**

Sigmoid function cũng được sử dụng làm kernel:



kernel = 'sigmoid'

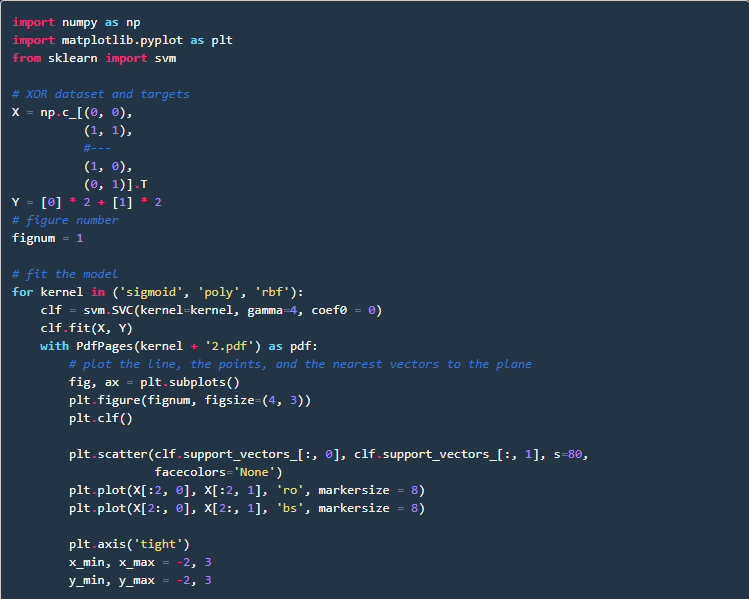
* **Bảng tóm tắt các kernel thông dụng**

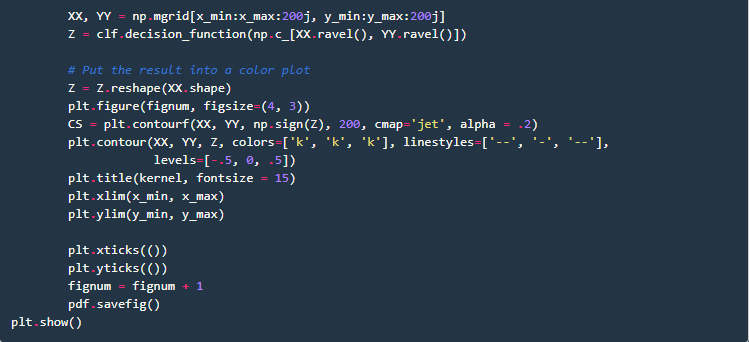


**1.3.3.3.3 Ví dụ minh họa:**

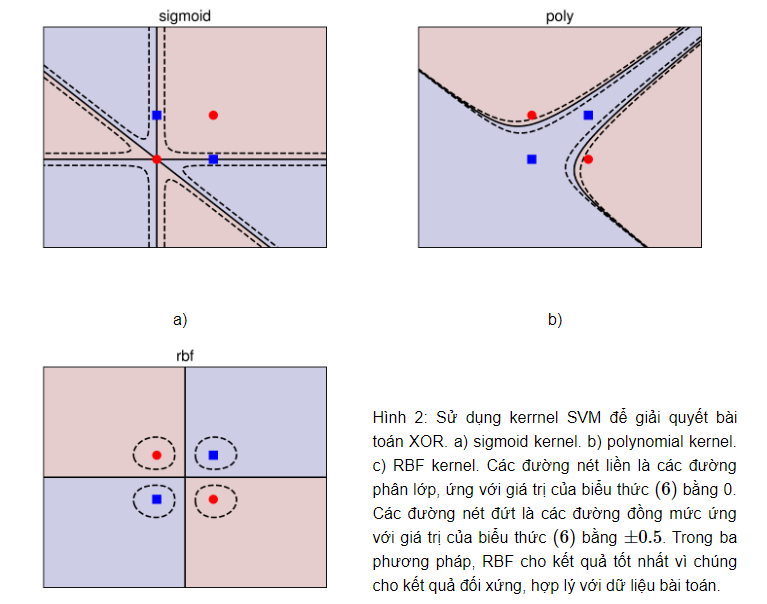
**a)  Bài toán XOR**

Chúng ta cùng quay lại với bài toán XOR. Chúng ta biết rằng [bài toán XOR không thể giải quyết nếu chỉ dùng một bộ phân lớp tuyến tính](https://machinelearningcoban.com/2017/02/24/mlp/#-pla-cho-cac-ham-logic-co-ban). Neurrel Network cần 2 layers để giải quyết bài toán này. Với SVM, chúng ta có cách để chỉ cần sử dụng một bộ phân lớp. Dưới đây là ví dụ:





Kết quả được cho trong Hình 2 dưới đây:

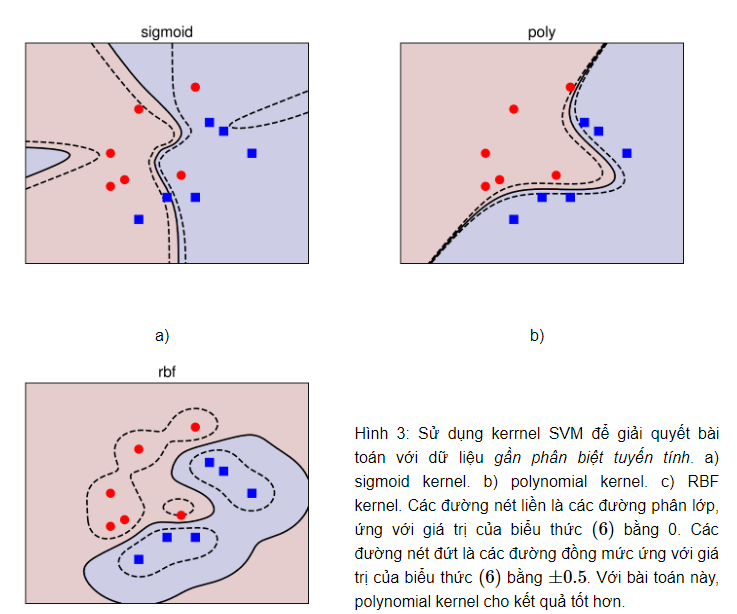


Ta có các nhận xét đối với mỗi kernel như sau:

* sigmoid: nghiệm tìm được không thật tốt vì có 3 trong 4 điểm nằm chính xác trên đường phân chia. Nói cách khác, nghiệm này rất *nhạy cảm với nhiễu*.
* poly: Nghiệm này có tốt hơn nghiệm của sigmoid nhưng kết quả có phần giống với [overfitting](https://machinelearningcoban.com/2017/03/04/overfitting/).
* rbf: Dữ liệu được tạo ra một cách đối xứng, đường phân lớp tìm được cũng tạo ra các vùng đối xứng với mỗi class. Nghiệm này được cho là *hợp lý hơn*. Trên thực tế, các rbf kernel được sử dụng nhiều nhất và cũng là lựa chọn mặc định trong hàm sklearn.svm.SVC.

**b) Dữ liệu gần linearly separable**

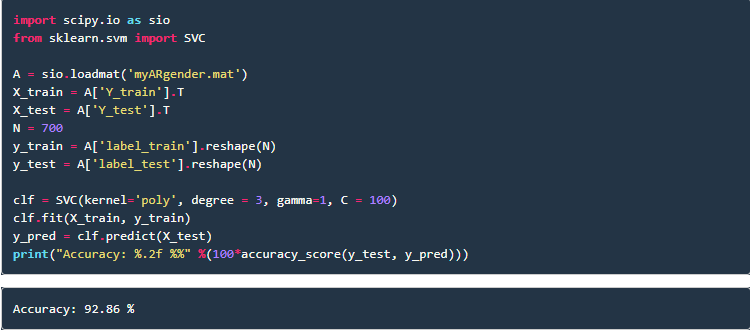
Xét một ví dụ khác với dữ liệu giữa hai classes là *gần phân biệt tuyến tính* như HÌnh 3 dưới đây:



Trong ví dụ này, kernel = 'poly' cho kết quả tốt hơn kernel = 'rbf' vì trực quan cho ta thấy rằng nửa bên phải của mặt phẳng nên hoàn thoàn thuộc vào class xanh. sigmoid kernel cho kết quả không thực sự tốt và ít được sử dụng.

1. **Bài toán phân biệt giới tính**

Dưới đây là ví dụ về cách sử dụng thư viện sklearn.svm.SVC để giải quyết bài toán:



Kết quả không tệ! Các bạn thử thay các kernel và thiết lập các tham số khác xem kết quả thay đổi như thế nào. Vì dữ liệu giữa hai classes là *gần phân biệt tuyến tính* nên không có sự khác nhau nhiều giữa các kernel.