**ĐỒ ÁN**

**TRÍ TUỆ NHÂN TẠO**

**Digital Recognizer**

**Using**

**Support Vector Machines**

Giảng viên hướng dẫn: Phạm Minh Tuấn

Sinh viên thực hiện: Nguyễn Đức Tuệ Anh

**Phần một: SVM và bài toán phân lớp**

**Bài toán phân lớp**

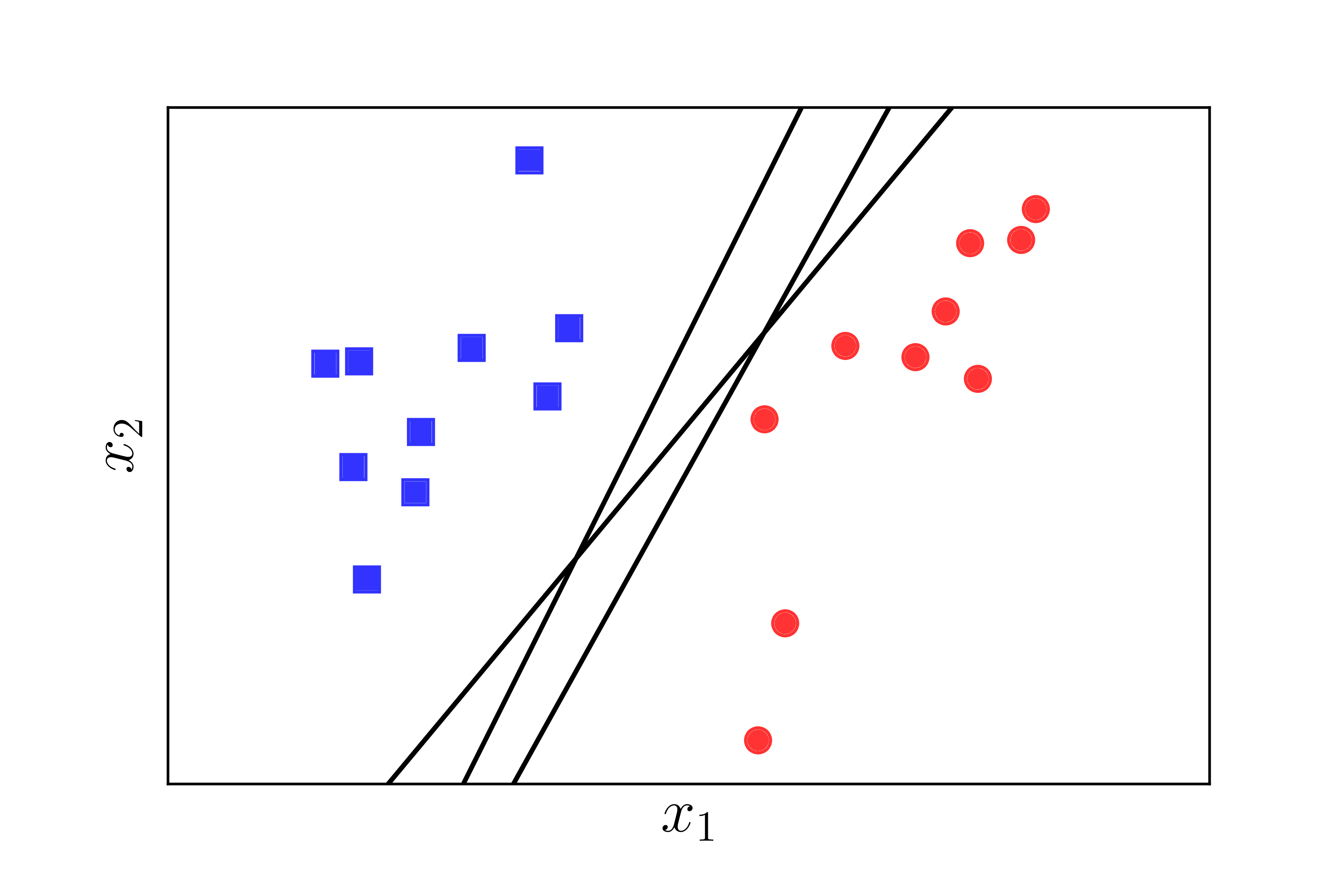
Đinh nghĩa: Cho tập dữ liệu D là một tập gồm m cặp xi, yi, với xi là một vector n chiều, yi là “nhãn” để chỉ lớp của vector đó.

Phân lớp nhị phân: yi = -1(class1) hoặc +1(class2)

Phân lớp nhị phân tuyến tính: Dữ liệu là một tập có thể phân chia tuyến tính nghĩa là ta có thể vẽ một đường thẳng để phân chia 2 lớp trong không gian 2 chiều và trong không gian n chiều là một hyperplane (siêu phẳng)

Một hyperplane trong không gian n chiều được định nghĩa bởi phương trình: wx+b = 0 với w là x là 2 vector n chiều

Vấn đề: Chúng ta có thể thấy có nhiều hyperplane để phân chia tập dữ liệu. Và có những phương pháp tìm ra separating hyperplane bằng ngẫu nhiên. Vậy làm thế nào để chúng ta lựa chọn ra phương án “tối ưu” dựa trên tập tiêu chí nào đó? Hay cụ thể hơn là chúng ta xem những điểm nào ảnh hưởng đến việc tối ưu.



Phương án 1: tất cả các điểm trên tập dữ liệu; Phương pháp áp dụng tiêu chí này: Linear regression, Neural nets

Phương án 2: Một số điểm gần với đường biên giới (decision boundary)

* Phương pháp SVM (Support Vector Machines)

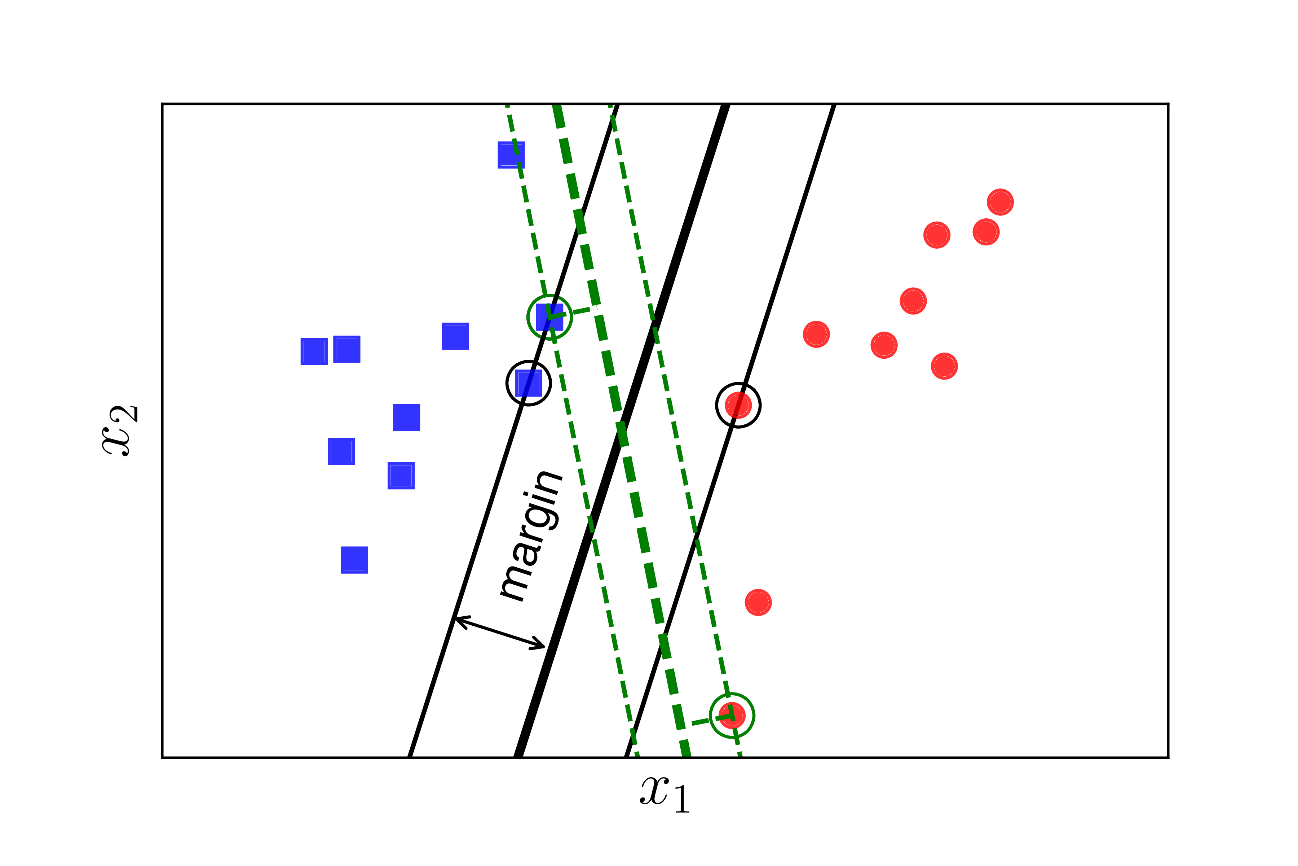
**SVM với bài toán phân lớp nhị phân:**

Cho một tập dữ liệu linearly separable

Và một hyperplane có vector pháp tuyến w và bias là b, ta gọi geometric margin M của hyperplane được định nghĩa bởi :

Với là geometric margin của mẫu

SVM đánh giá một hyperplane là tối ưu khi M là lớn nhất



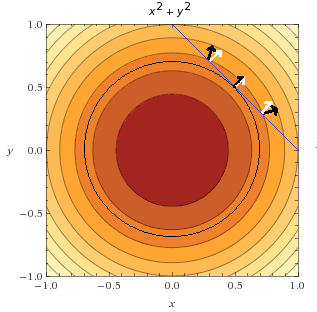
Bài toán tối ưu của chúng ta là:

Phép bình phương và hệ số 1/2 để ta đưa bài toán về dạng toàn phương

**Giải bài toán tối ưu của SVM**

**Phương pháp nhân tử Lagrange:**

Lagrange nhận xét là khi chúng ta tìm cách giải một bài toán tối ưu có dạng:



Cực tiểu của hàm f được tìm thấy khi gradient của f có cùng hướng với gradient của g.

Nghĩa là:

Vì vậy nếu chúng ta muốn tìm cực tiểu của hàm f với điều kiện là g thì ta cần giải quyết:

Hằng số được gọi là nhân tử Lagrange (Lagrange multiplier).

Để đơn giản hóa phương pháp, chúng ta nhận xét là nếu định nghĩa một hàm

Thì gradient của nó là

Giải cũng chính là tìm điểm cực tiểu của bài toán

Tổng quát hoát lên với dạng bài có nhiều ràng buộc:

Và Lagrangian function của chúng ta là:

Đây là hàm m + n ẩn, với n là số ẩn của x, m là số ẩn của . Lấy vi phân ta được n + m phương trình, mỗi phương trình bằng 0.

**Áp dụng Lagrange vào bài toán tối ưu**

Bài toán tối ưu của SVM là:

Vậy ở đây, chúng ta có:

Và m hàm ràng buộc:

Lagrangian function (primal form) với bài toán tối ưu của SVM:

Chú ý là mỗi hàm ràng buộc đều đi kèm với một nhân tử

Ta có thể giải bằng phương pháp giải tích với số lượng mẫu nhỏ. Bài toán Lagrangian ở đây là

Vì thế tại điểm cực tiểu:

Ta vừa phải minimize theo w, b vừa maximize theo cùng một lúc.

Chú ý: Phương pháp nhân tử Lagrange được sử dụng để giải bài toán có ràng buộc đẳng thức, nhưng trong trường hợp này ràng buộc ở đây là bất đẳng thức. Để phương pháp vẫn làm việc, ta có thêm điều kiện KKT sẽ được trình bày ở phần sau.

**Bài toán đối ngẫu**

Bài toán Lagrangian của SVM có m ràng buộc bất đẳng thức (m ở đây là số lượng mẫu trong tập training) và thông thường, ta giải nó ở dạng đối ngẫu (dual form). Nguyên lý đối ngẫu (dual principle) nói rằng: bài toán tối ưu có thể được nhìn từ 2 quan điểm. Một là bài toán gốc (primal problem), trong trường hợp này là bài toán cực tiểu hóa; và hai là bài toán đối ngẫu (dual problem), bài toán cực đại hóa.

Điểm thú vị ở đây là cực đại ở bài toán của bài toán đối ngẫu bé hơn hoặc bằng cực tiểu của bài toán gốc

Ở bài toán gốc (primal problem) mà ta đã nêu ở phần trước, ta có:

Giải bài toán bằng cách đạo hàm từng phần L theo w và b

Ta thu được

Thay w vào Lagrangian function, ta được:

Thay vào,ta thu được kết quả cuối cùng:

Đây được gọi là **Wolfe dual Lagrangian function.**

Bài toán tối ưu lúc này có dạng:

Lợi ích chính cùa bài toán đối ngẫu là hàm mục tiêu W chỉ phụ thuộc vào các nhân tử Lagrange. Ngoài ra nó còn giúp chúng ta giải dễ dàng với Python, cũng như cực kì có ích khi chúng ta định nghĩa “kernel”

**Tiêu chuẩn Slater**

Nếu tồn tại w, b thỏa:

thì strong duality thỏa mãn.

Vì luôn có siêu phẳng để phân chia 2 lớp *linearly separable,* tức là bài toán có nghiệm nên feasible set của bài toán tối ưu khác rỗng. tức là tồn tại sao cho:

Vậy ta chỉ cần chọn và , ta sẽ có:

Suy ra tiêu chuẩn Slater được thỏa mãn

**Điều kiện KKT (Karush-Kuhn-Tucker conditions)**

Bởi vì chúng ta đang làm việc với những ràng buộc bất đẳng thức, nên có yêu cầu kèm theo đó là: lời giải của chúng ta phải thỏa mãn điều kiện KKT

Với bài toán này, nếu lời giải chúng ta tìm ra thỏa mãn điều kiện KKT thì lời giải này tối ưu.

Điều kiện KKT gồm:

-Điều kiện về tính dừng: Điểm được chọn phải là điểm dừng, là điểm mà tại đó hàm ngừng tăng hoặc giảm. Khi không có ràng buộc thì điều kiện này chỉ là điểm mà ở đó gradient của hàm mục tiêu bằng 0. Khi có ràng buộc như trường hợp này, thì ta dùng gradient của Lagrangian.

-Điều kiện khả thi của bài toán gốc: Đây là điều kiện của bài toán primal

-Điều kiện khả thi của bài toán đối ngẫu:

-Điều kiện bổ sung: hoặc

* Support vectors là những mẫu có nhân tử Lagrange tương ứng dương. Là những mẫu có

**Vậy bước tiếp theo khi có các nhân tử là gì?**

Giải bài toán đối ngẫu Wolfe, giúp ta thu được vector chứa toàn bộ các nhân tử Lagrange. Tuy vậy mục đích của ta là tìm w và b.

Tính w: Từ gradient của Lagrangian, ta tính w theo công thức:

Tính b: Dựa vào điều kiện của bài toán gốc, những điểm gần nhất với mặt phân cách có:

Ta đã biết các giá trị còn lại ngoài b, vì thế ta tính b như bằng cách nhân 2 vế cho , vì nên:

Ngoài ra có nhiều phương pháp tính b như phương pháp trung bình được đề cập trong *Pattern Recognition and Machine Learning (Bishop, 2006)*

Với S là số lượng các support vectors

Hoặc phương pháp (Cristianini & Shawe-Taylor, 2000) và (Ng)

**Giải SVM bằng QP solver**

QP solver là một chương trình giải QP problem. Ở đây ta sử dụng thư viện CVXOPT của Python

Thư viện cung cấp phương pháp giải bài toán toàn phương có dạng:

Chuyển về dạng với biến để phù hợp với bài toan đối ngẫu

Ta chuyển bài toán đối ngẫu về dạng. Đầu tiên, ta chuyển bài toán cực đại hóa

Thành bài toán cực tiểu hóa bằng cách nhân với -1

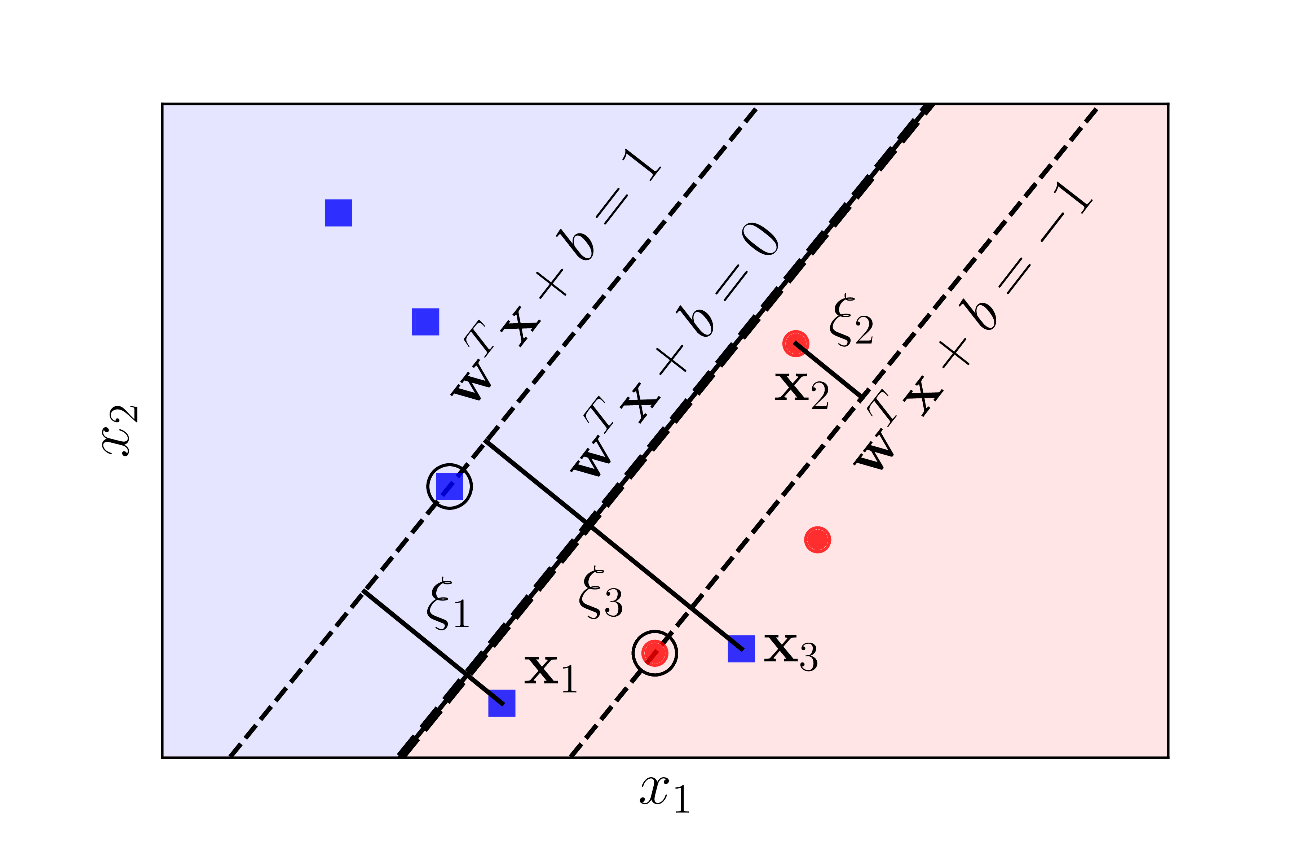
Ta đinh nghĩa , và ma trận Gram K của tất cả phép nhân vô hướng có thể có của vector

Ta dùng những cái trên để thiết lập phiên bản vector hóa của bài toán đối ngẫu với

**Soft SVM và Kernel SVM**

**Soft SVM**

Trong thực tế dữ liệu không thể phân chia tuyến tính bởi nhiễu. Vào năm 1995, Vappik và Cortes giới thiệu phiên bản sửa đổi của SVM nguyên bản cho phép việc phân loại có thể có một số lỗi. Mục đích không phải là phân loại không có lỗi, mà là có ít lỗi nhất có thể.



Để làm được điều đó, ta thêm biến zeta vào các ràng buộc:

Một vấn đề nảy sinh là nếu ta chọn gía trị cực lớn với mỗi mẫu thì toàn bộ ràng buộc sẽ bị thỏa mãn.

Để giải quyết, chúng ta thay đổi hàm mục tiêu như sau:

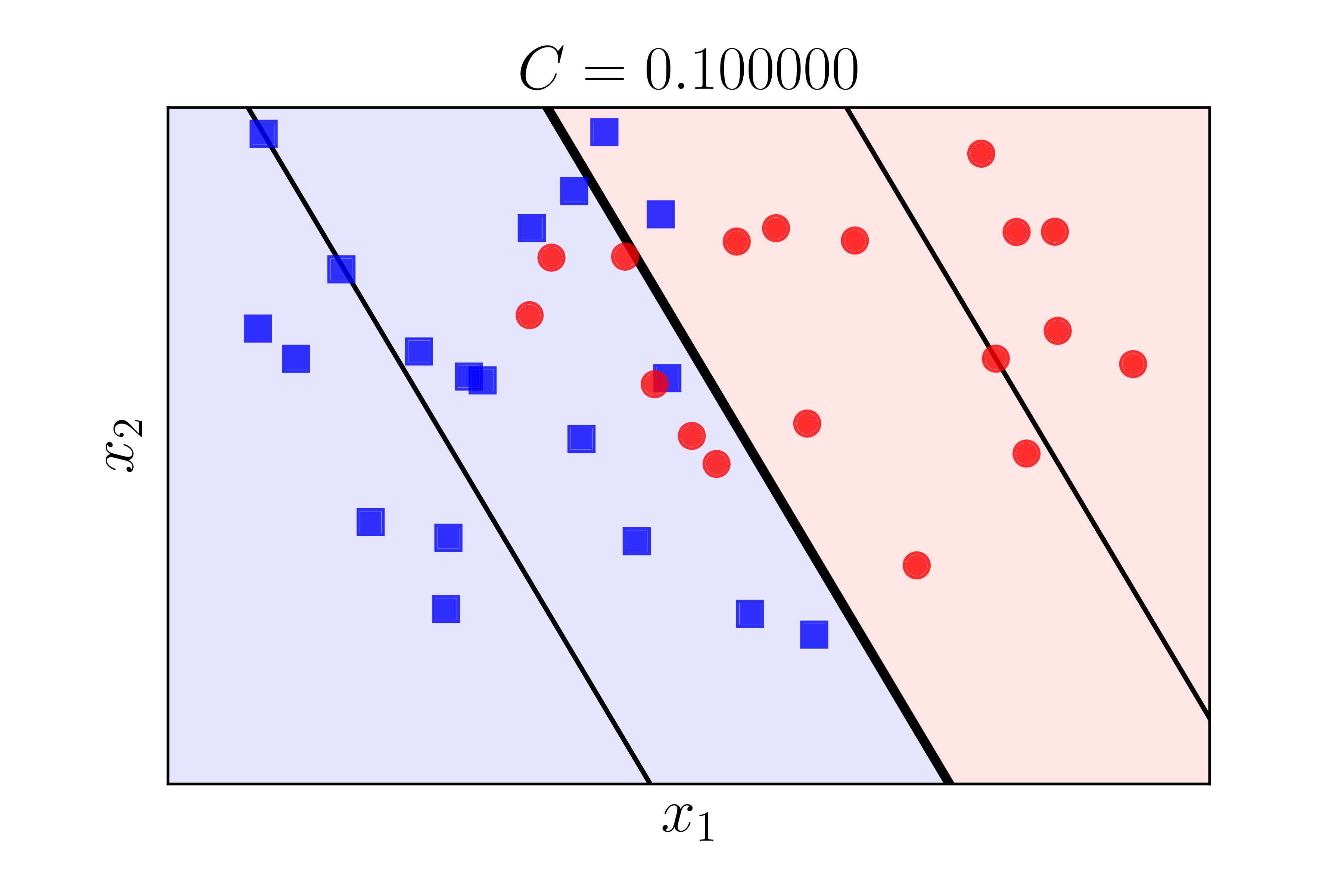
Để tránh việc tối thiểu hóa bằng giá trị âm, ta thêm điều kiện vào. Ngoài ra để kiểm soát soft margin, ta sử dụng tham số C, xác định tầm quan trọng của zeta

Soft margin function:

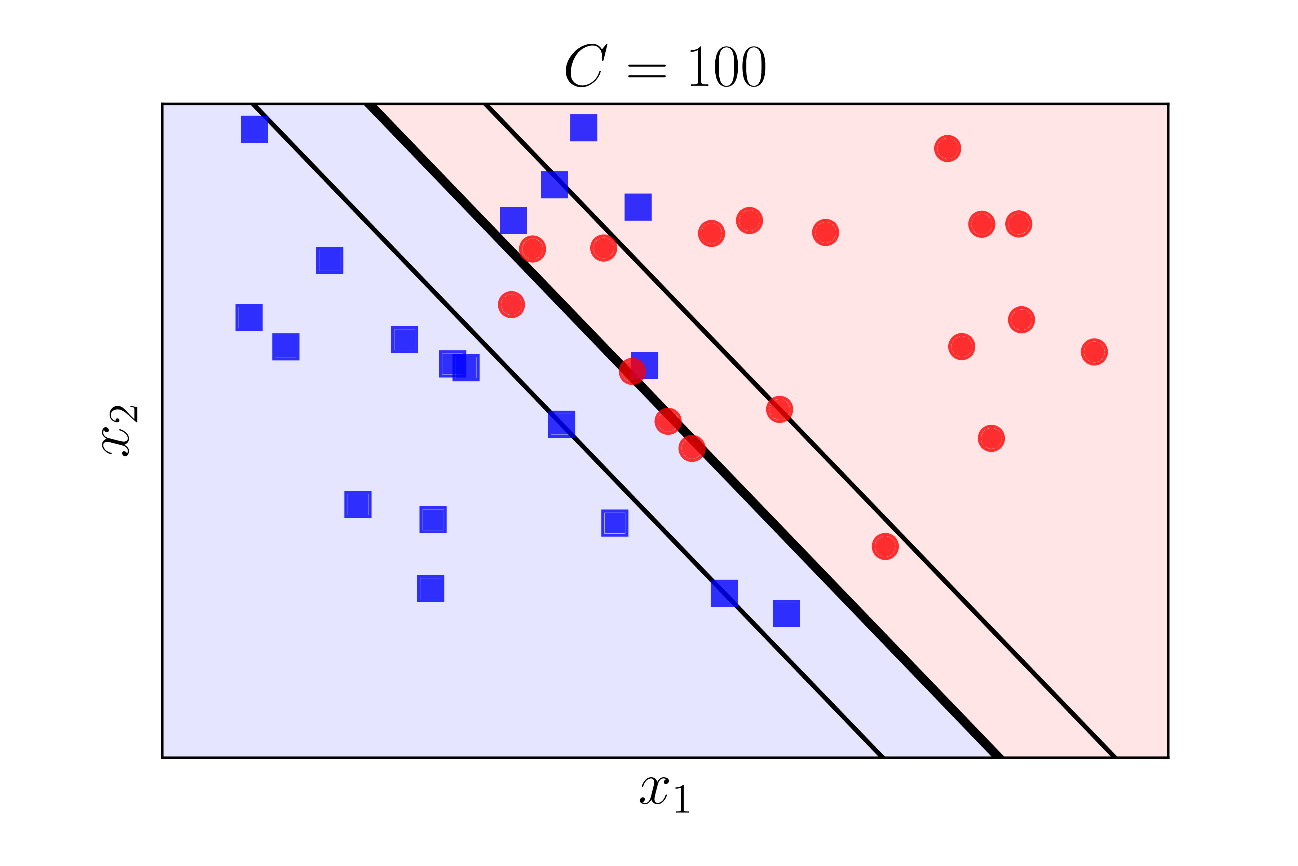
Tương tự, SVM nguyên bản (hard SVM) ta có được bài toán đối ngẫu với một chút khác biệt về ràng buộc:

Ràng buộc trở thành – box contrain: vector bị rang buộc bên trong một chiếc hộp có chiều dài mỗi cạnh là C.

Tham số C cho phép ta kiểm soát được việc xử lí các lỗi của SVM. Cụ thể:



Giá trị C nhỏ sẽ cho ta margin rộng hơn, với cái giá là một số phân loại bị sót.



Giá trị C cực lớn sẽ đưa ta về lại với hard margin và không có rang buộc nào bị vi phạm cả.

Mục tiêu là tìm giá trị C sao cho giá trị nhiễu không quá ảnh hưởng đến đáp án được đưa ra nhất.

Làm thế nào để tìm được giá trị C tốt nhất?

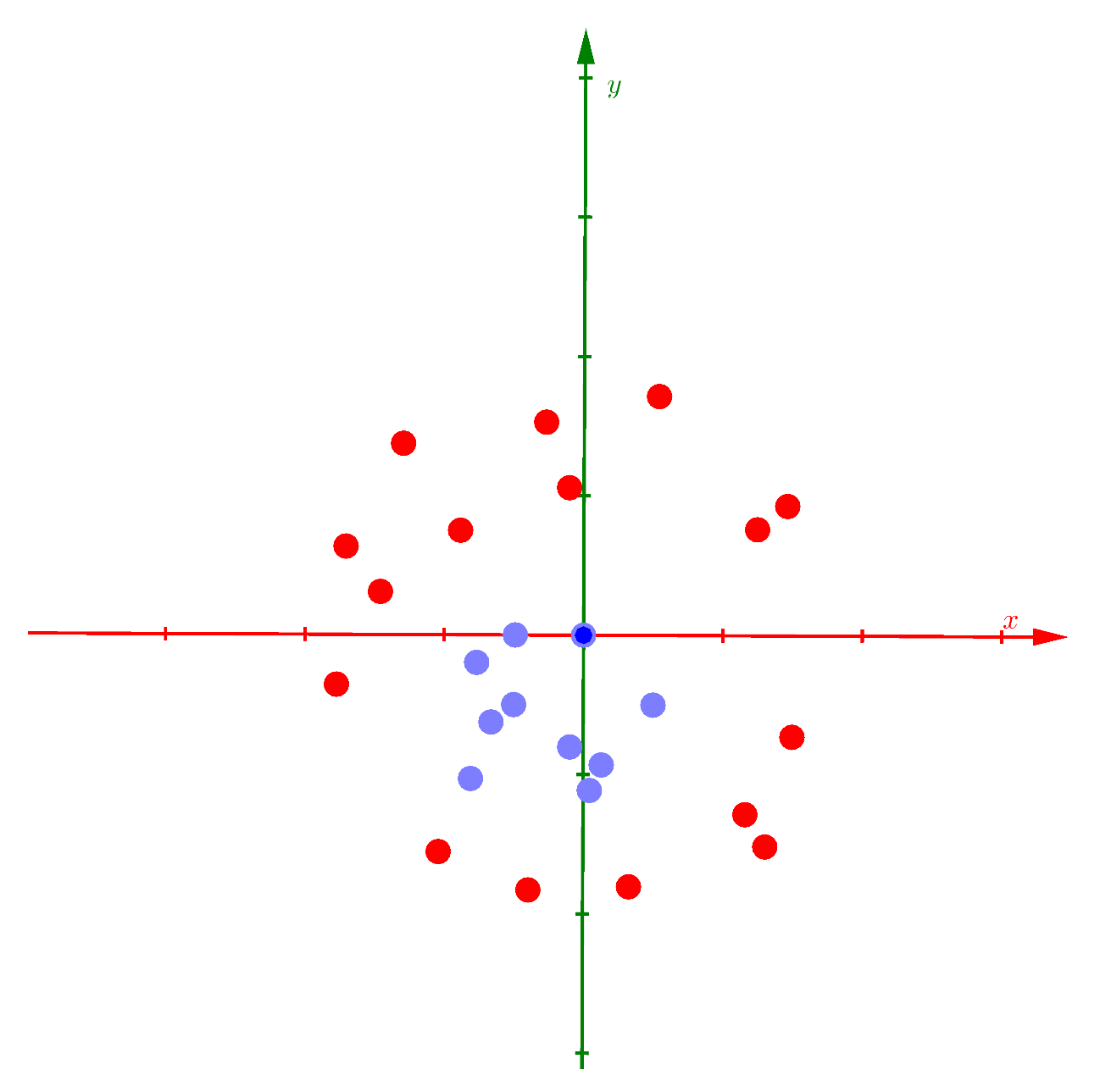
Không có giá trị C nào hoạt động với mọi bài toán. Phương pháp tiếp cận được đề nghị là lựa chọn C bằng cách sử dụng grid search với cross-validation *(Hsu, Chang, & Lin, A Practical Guide to Support Vector Classification)*

**Kernel SVM**

**Feature Transformation**

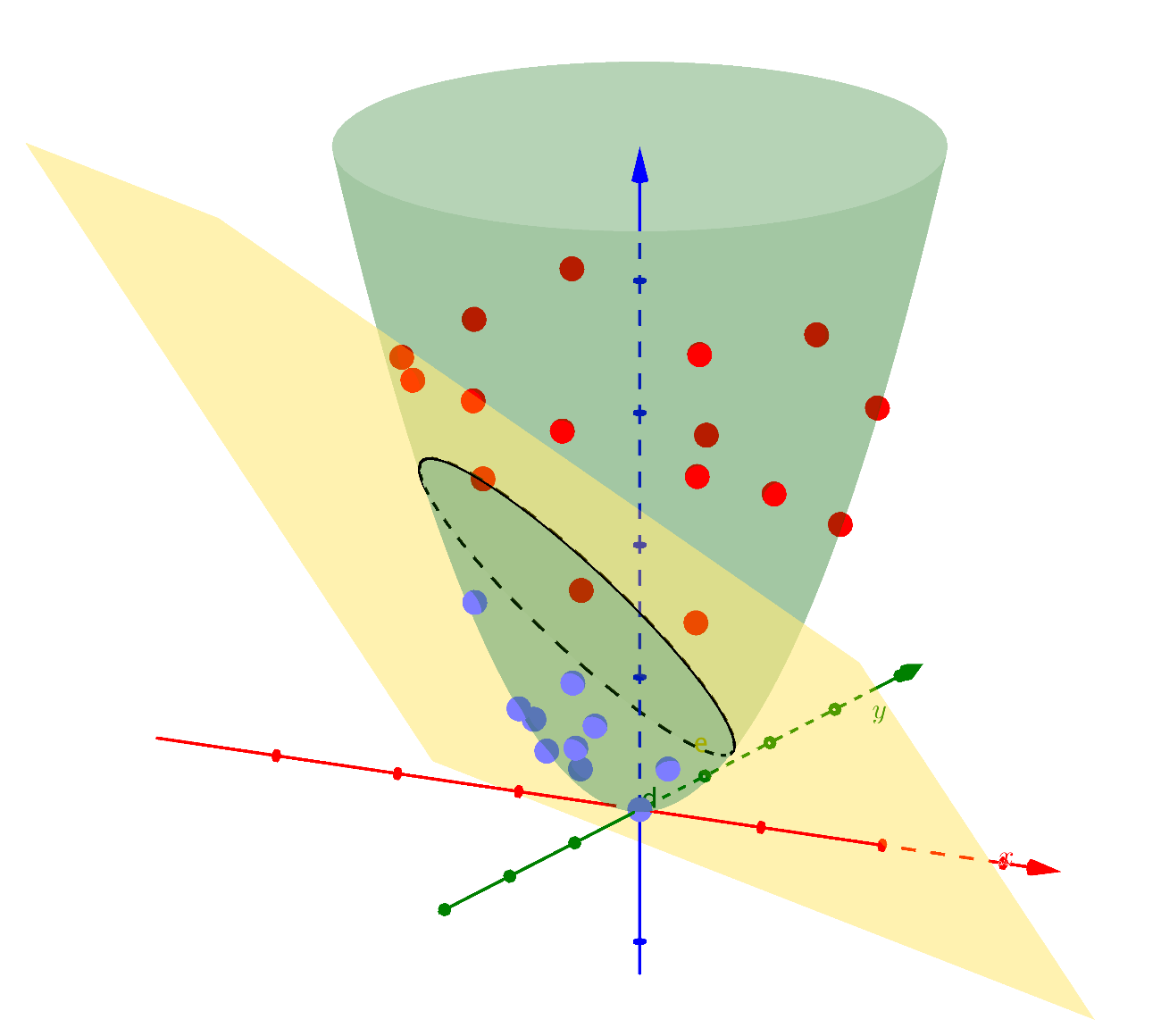
**Classify non-linearly separable data**

Liệu ta có thể sử dụng SVMs để phân loại với dữ liệu có mặt phân chia không tuyến tính (non-linearly separable data). Không thể. Hình bên dưới cho ta thấy dữ liệu không thể phân chia tuyến tính trong không gian 2 chiều.

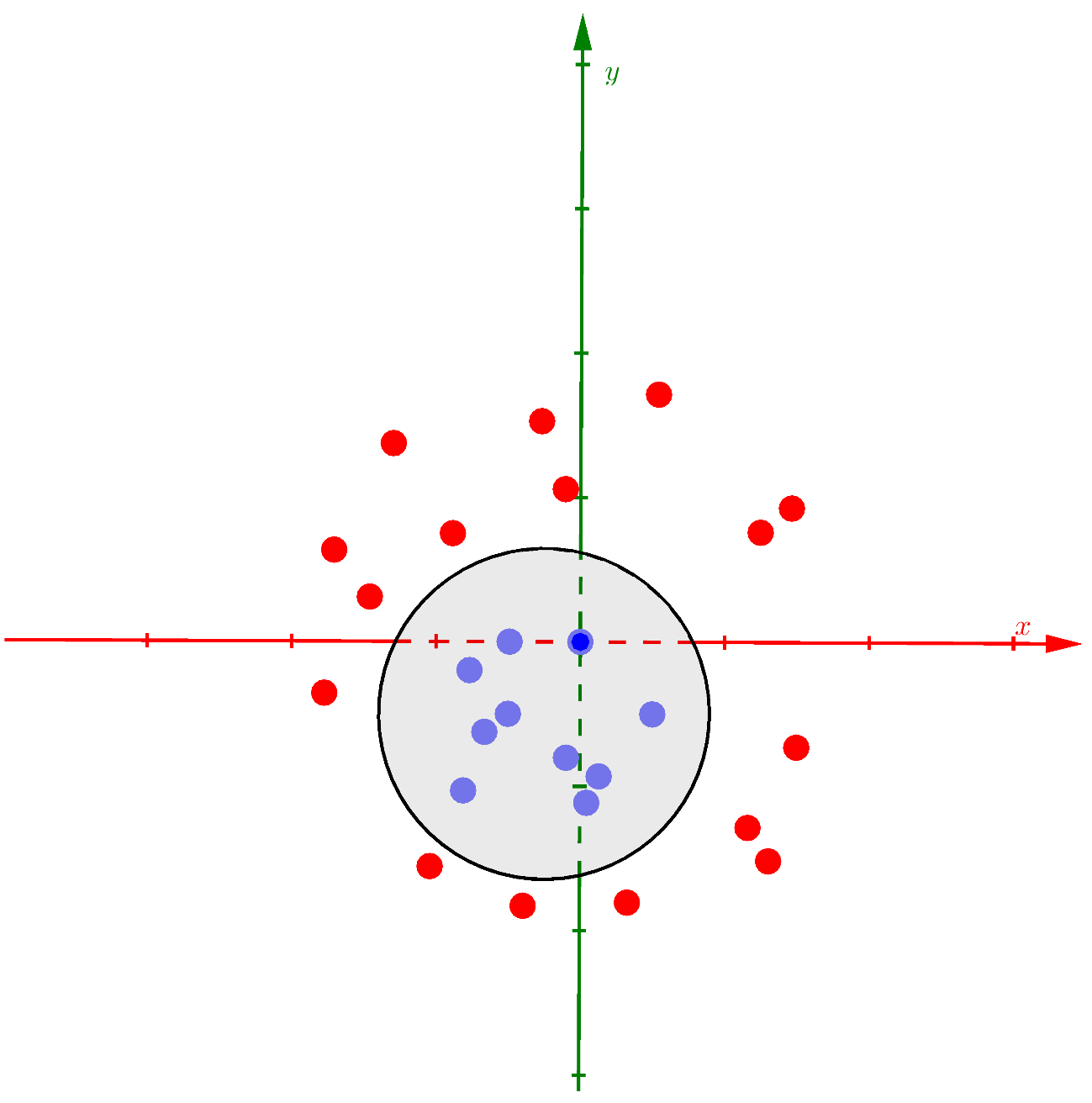


Vậy nếu ta biến đổi dữ liệu gốc 2 chiều rồi mới đưa vào SVMs thì thế nào? Một phép biến đổi khả thi, chẳng hạn như là chuyển toàn bộ vector 2 chiều x1 x2 thành vector 3 chiều.

Ví dụ ta thực hiện ánh xạ đa thức bằng cách sử dụng hàm được định nghĩa như sau:



Dữ liệu có thể phân chia bằng một mặt phẳng. Chúng ta không bắt buộc phải biến đổi sang không gian 3 chiều, có thể là nhiều chiều hơn.



Lựa chọn phép biển đổi phụ thuộc phần lớn vào tập dữ liệu.

**Kernel là gì?**

Chúng ta đã có được một phương thức để sử dụng SVMs với non-linearly separate data. Tuy vậy, trở ngại của việc này là ta phải biến đổi tất cả các mẫu. Nếu số lượng mẫu lớn, cũng như phép biến đổi phức tạp thì việc này sẽ tốn một lượng khổng lồ thời gian. Đấy là lúc ta cầu cứu cái gọi là kernel.

Nhắc lại về hàm đối ngẫu. Chúng ta không cần quan tâm đến giá trị của mẫu huấn luyện x, chúng ta chỉ cần giá trị của tích vô hướng của

Vậy có cách nào để tính giá trị này mà không cần phải biến đổi vector? Sử dụng kernel

Kernel là một hàm sẽ trả về kết quả của tích vô hướng được biểu diễn trong một không gian khác.

Định nghĩa: Cho hàm ánh xạ , ta gọi hàm được định nghĩa bởi với biểu thị tích trong (inner product) trong không gian V, là một hàm kernel.

**Kernel trick**

Nếu ta định nghĩa một Kernel: , chúng ta có thể viết lại hàm đối ngẫu như sau

Áp dụng kernel trick đơn giản chỉ là thay thế tích vô hướng của 2 mẫu bằng hàm kernel

Thay đổi trông có vẻ đơn giản nhưng chúng ta phải thực hiện một lượng lớn công việc để có được dạng đối ngẫu Wolf từ bài toán tối ưu gốc. Giờ ta có thể sử dụng sức mạnh của kernel vào phân loại những dữ liệu không thể tách biệt được.

SVMs còn được gọi là ***sparse kernel machines*** (máy thưa kernel), vì chúng ta chỉ phải tính các hàm kernel của các support vectors chứ không phải toàn bộ vector.

**Một số loại kernel**

**Linear kernel:** Là kernel đơn giản nhất, định nghĩa như sau:

Với x, x’ là hai vectors.

Linear kernel làm việc tốt với text classification.

**Polynomial kernel**

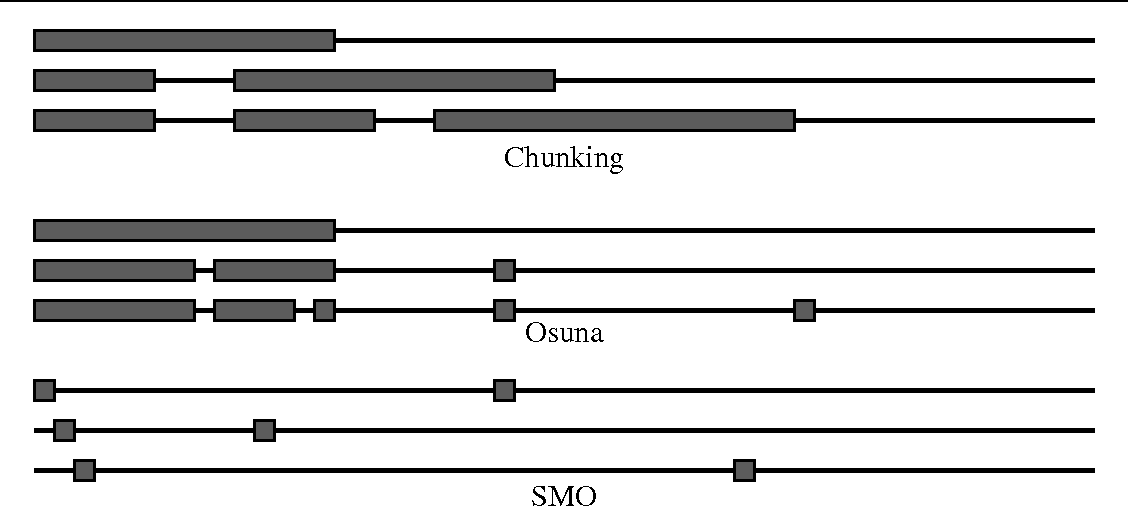
Nó có 2 tham số: hằng số c, và độ (degree) của kernel d

Ngoài ra còn có RBF/Gaussian kernel, sigmoid kernel,…

**Thuật toán SMO**

Chúng ta sẽ đến với một thuật toán mô tả một phương pháp giải bài toán tối ưu SVM một cách nhanh chóng: thuật toan SMO **(sequential minimal optimization)**. Phần lớp các thư viện ML đều sử dụng SMO hoặc bà con của nó.

SMO giải bài toán tối ưu sau:



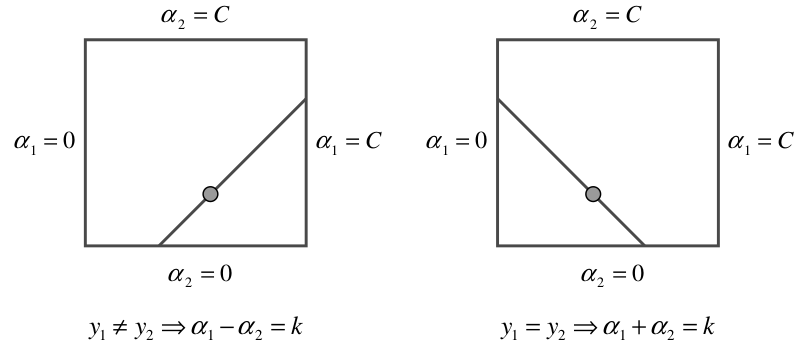
Bài toán trên có thể giải bằng cách tiếp cận truyền thống bằng việc sử dụng công cụ QP solver. Cách tíếp cận này nặng nề cũng như không thực tế khi làm việc với tập dữ liệu lớn bởi tiêu tốn khá nhiều thời gian và yêu cầu một lượng bộ nhớ đáng kể. Thuật toán SMO được Platt đưa ra vào năm 1999 tránh việc sử dụng các công cụ trên, thay vào đó bằng cách phân tích, giải quyết một lượng lớn các bài toán tối ưu con, nhỏ và những bài toán này chỉ gắn với 2 nhân tử Lagrange trong một thời điểm. Đây là bài toán tối ưu nhỏ nhất có thể bởi vì nhưng nhân tử Lagrange phải tuân theo ràng buộc đẳng thức tuyến tính.

Tại mỗi bước, SMO sẽ chọn 2 nhân tử để tham gia vào việc tối ưu, tìm kiếm giá trị tối ưu của những nhân tử, và cập nhập SVM để thu được một giá trị tối ưu mới

Điểm lợi của SMO là nằm ở thực tế giải 2 nhân tử có thể hoàn thanh bằng phương pháp giải tích. Ngoài ra SMO không yêu cầu lưu trữ ma trận.

Có 2 thành phần trong SMO: phương pháp phân tích để tối ưu 2 nhân tử Lagrange và bước heuristic để chọn 2 nhân tử cần tối ưu

**Tối ưu 2 nhân tử**



Hai nhân tử phải đáp ứng toàn bộ các ràng buộc của bài tóan chính. Ràng buộc bất đẳng thức làm cho các nhân tử nằm trong hộp. Ràng buộc đẳng thức tuyến tính làm cho chúng nằm trên diagonal line.

Gọi và là giá trị trước khi update. và là giá trị mới đã được tối ưu. Để bảo toàn ràng buộc đẳng thức của bài toán đối ngẫu

Với s =

Chúng ta sẽ cố định rồi tính dựa vào

Cận trên và cận dưới của dễ dàng được xác định như sau:

Nếu :

Nếu :

Cập nhập nhân tử

Kết quả của việc tối ưu W theo 2 nhân tử, ta thu được:

**Heuristic để chọn nhân tử:**

First heuristic: mỗi lần SMO xét một mẫu, nó sẽ kiểm tra điều kiện KKT có bị vi phạm không (với lỗi chấp nhận được là )

Lần lặp đầu tiên sẽ chạy toàn bộ các mẫu. Sau lần lặp đầu tiên, vòng lặp ngoài sẽ xen kẽ giữa lần duyệt đơn qua toàn bộ các mẫu, và duyệt nhiều lần những mẫu thường hay xảy ra vi phạm . Thuật toán kết thúc khi toàn bộ mẫu thỏa mãn điều kiện trong pham vi lỗi

Sau khi đã chọn được nhân tử đầu tiên, SMO sẽ chọn nhân tử thứ 2 mà nhân tử này sau khi cập nhập sẽ có thay đổi lớn nhất.

Chúng ta update

Việc này chọn bằng cách tính step của mỗi và lựa chọn step lớn nhất sẽ phải gọi hàm kernel 3 lần trong mỗi lần lặp. Thay vào đó Platt cho ta một công thức xấp xỉ:

**Tính toán b và w**

Việc tính b sẽ được lặp lại theo mỗi bước. sẽ hợp lệ nếu mới không nằm ở biên.

Tương tự với

Khi cả 2 đều hợp lệ thì chúng bằng nhau. Ngược lại thì tính trung binh cộng của chúng.

Với linear SVM w được tinh như sau:

Mã giả:

|  |
| --- |
| target = desired output vector point = training point matrix procedure takeStep(i1,i2) if (i1 == i2) return 0 alph1 = Lagrange multiplier for i1 y1 = target[i1] E1 = SVM output on point[i1] – y1 (check in error cache) s = y1\*y2 Compute L, H via equations (13) and (14) if (L == H) return 0 k11 = kernel(point[i1],point[i1]) k12 = kernel(point[i1],point[i2]) k22 = kernel(point[i2],point[i2]) eta = k11+k22-2\*k12 if (eta > 0) { a2 = alph2 + y2\*(E1-E2)/eta if (a2 < L) a2 = L else if (a2 > H) a2 = H } else { Lobj = objective function at a2=L Hobj = objective function at a2=H if (Lobj < Hobj-eps) a2 = L else if (Lobj > Hobj+eps) a2 = H else a2 = alph2 } if (|a2-alph2| < eps\*(a2+alph2+eps)) return 0 a1 = alph1+s\*(alph2-a2) Update threshold to reflect change in Lagrange multipliers Update weight vector to reflect change in a1 & a2, if SVM is linear Update error cache using new Lagrange multipliers Store a1 in the alpha array Store a2 in the alpha array return 1 endprocedure procedure examineExample(i2) y2 = target[i2] alph2 = Lagrange multiplier for i2 E2 = SVM output on point[i2] – y2 (check in error cache) r2 = E2\*y2 if ((r2 < -tol && alph2 < C) || (r2 > tol && alph2 > 0)) { if (number of non-zero & non-C alpha > 1) { i1 = result of second choice heuristic (section 2.2) if takeStep(i1,i2) return 1 }  loop over all non-zero and non-C alpha, starting at a random point { i1 = identity of current alpha if takeStep(i1,i2) return 1 } loop over all possible i1, starting at a random point { i1 = loop variable if (takeStep(i1,i2) return 1 } } return 0 endprocedure main routine: numChanged = 0; examineAll = 1; while (numChanged > 0 | examineAll) { numChanged = 0; if (examineAll) loop I over all training examples numChanged += examineExample(I) else loop I over examples where alpha is not 0 & not C numChanged += examineExample(I) if (examineAll == 1) examineAll = 0 else if (numChanged == 0) examineAll = 1 } |

**Phần hai: Áp dụng SVM vào nhận dạng chữ số viết tay**

**Nhận dạng chữ số viết tay**

**MNIST dataset**

[Bộ cơ sở dữ liệu MNIST](http://yann.lecun.com/exdb/mnist/) là bộ cơ sở dữ liệu lớn nhất về chữ số viết tay và được sử dụng trong hầu hết các thuật toán nhận dạng hình ảnh (Image Classification).

MNIST bao gồm hai tập con: tập dữ liệu huấn luyện (training set) có tổng cộng 60k ví dụ khác nhau về chữ số viết tay từ 0 đên 9, tập dữ liệu kiểm tra (test set) có 10k ví dụ khác nhau. Tất cả đều đã được gán nhãn. Hình dưới đây là ví dụ về một số hình ảnh được trích ra từ MNIST.



Mỗi bức ảnh là một ảnh đen trắng (có 1 channel), có kích thước 28x28 pixel (tổng cộng 784 pixels). Mỗi pixel mang một giá trị là một số tự nhiên từ 0 đến 255. Các pixel màu đen có giá trị bằng 0, các pixel càng trắng thì có giá trị càng cao (nhưng không quá 255)

**Áp dụng thuật toán**

|  |  |  |  |
| --- | --- | --- | --- |
|  | CVXOPT | SMO | Sklearn.SVC |
| Thời gian | 10’ với 1500 mẫu | 2 tiếng với 1500 mẫu | 15’ với 42000 mẫu |
| Kaggle result |  | 0.78985 |  |

**Tài liệu tham khảo**

CS229 Lecture notes-Andrew.Ng

CS 229, Autumn 2009 The Simplified SMO Algorithm

A Roadmap to SVM Sequential Minimal Optimization for Classification- Ruben Ramirez-Padron

A Fast Algorithm for Training Support Vector Machines-John C. Platt, Microsoft Research

SVM Succinctly- Alexandre Kowalczyk

<https://www.svm-tutorial.com/>

An Idiot’s guide to Support vector machines (SVMs)- R. Berwick, Village Idiot