**MỤC LỤC**

[**1 . Bài toán phân lớp dữ liệu: (Classification)** 2](#_Toc10773883)

[**2 . Support Vector Machine : (SVM)** 3](#_Toc10773884)

[**3 . Bài Toán của SVM :** 3](#_Toc10773885)

[**4 . Giải bài toán tối ưu:** 5](#_Toc10773886)

[**5 . Soft Margin :** 11](#_Toc10773887)

[**6 . Kernel SVM** 14](#_Toc10773888)

[**7 . Mutil – SVM** 16](#_Toc10773889)

[**8 . Thuật toán SMO** 17](#_Toc10773890)

[**9 . Áp dụng SVM cho nhận dạng chữ số viết tay :** 17](#_Toc10773891)

# **1 . Bài toán phân lớp dữ liệu: (Classification)**

-*Phân lớp dữ liệu* :  
 Là một trong những vấn đề cơ bản và quan trọng trong machinelearning . Là vấn đề liên quan đến việc dạy máy làm thế nào để phân lớp giữa các dữ liệu với nhau theo các tiêu chí cụ thể( thường là các nhãn ) dựa trên các đặc điểm được xác định trước và đây được gọi là học có giám sát. Được áp dụng trong nhiều lĩnh vực khác nhau trong đời sống .

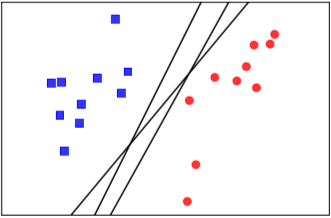
-*Các thuật toán điển hình nhằm giải bài toán phân lớp* :   
Logistic , K-NN , Perceptron , Decision Tree , Support Vector Machine , Neuron Network ….

-*Mô tả bài toán phân lớp nhị phân* :Cho tập D là một tập các dữ liệu có m cặp (,) :  
Trong đó : là vector n chiều : mô tả thuộc tính , đặc điểm dữ liệu  
 là “nhãn” : để chỉ lớp mà dữ liệu đó thuộc về  
(yi trong phân lớp nhị phân thường bằng -1 hoặc 1 đại diện cho 2 bên lớp dữ liệu)  
Việc máy làm là học tập trên tập dữ liệu đầu vào này , và xây dựng nên model đúng với dữ liệu phân chia tuyến tính(linearly separable) hoặc gần đúng với dữ liệu không thể phân chia tuyến tính (nonlinearly separable) để phục vụ cho việc phân lớp dữ liệu mới sau này .

-*Phân lớp nhị phân tuyến tính* :   
Dữ liệu là một tập có thể phân chia tuyến tính nghĩa là ta có thể tìm một đường thẳng để phân chia 2 lớp trong không gian 2 chiều , 1 mặt phẳng trong không gian 3 chiều và trong không gian n chiều là một hyperplane (siêu phẳng)   
Một siêu phẳng trong không gian n chiều có dạng :

+Trong đó : W là vector n chiều các hệ số của phương trình   
 X là vector n chiều các đặc trưng  
 b là số hạng tự do hay còn gọi là bias   
 (độ lệch giữa siêu phẳng và gốc tọa độ)

Có một vấn đề xảy ra ở đây là làm sao để tìm được siêu phẳng tốt nhất nhằm đạt sự khách quan nhất khi phân lớp dữ liệu mới sau này .



# **2 . Support Vector Machine : (SVM)**

-*Support Vector Machine (SVM)* : là một thuật toán phân lớp phổ biến và hiệu quả . Dựa trên cơ sở toán học vững chắc , SVM là một thuật toán cơ bản được coi là tốt bậc nhất trong machine learning .

-*Lịch sử phát triển* : Thuật toán SVM ban đầu được phát minh bởi [Vladimir N. Vapnik](https://en.wikipedia.org/wiki/Vladimir_N._Vapnik) và [Alexey Ya. Chervonenkis](https://en.wikipedia.org/wiki/Alexey_Chervonenkis) vào năm 1963. Năm 1992, Bernhard E. Boser, Isabelle M. Guyon và [Vladimir N. Vapnik đã](https://en.wikipedia.org/wiki/Vladimir_N._Vapnik) đề xuất một cách để tạo ra các trình phân loại phi tuyến bằng cách áp dụng [thủ thuật kernel](https://en.wikipedia.org/wiki/Kernel_trick) cho siêu phẳng biên tối đa. Tiêu chuẩn hiện tại hóa thân (lề mềm) đã được đề xuất bởi [Corinna Cortes](https://en.wikipedia.org/wiki/Corinna_Cortes) và Vapnik vào năm 1993 và được xuất bản vào năm 1995 .

-*Các SVM có thể được sử dụng để giải quyết các vấn đề khác nhau trong thế giới thực:* Các SVM rất hữu ích trong việc [phân loại văn bản và siêu văn bản](https://en.wikipedia.org/wiki/Text_categorization) , [phân loại hình ảnh](https://en.wikipedia.org/wiki/Image_classification) , các ký tự viết tay, sử dụng để nghiên cứu gen di truyền …

# **3 . Bài Toán của SVM :**

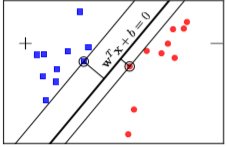
-Như ta đã biết về *bài toán phân lớp nhị phân* , ở đây SVM sẽ dựa trên tập m các cặp dữ liệu (,) tìm ra siêu phẳng tối ưu để phân lớp dữ liệu .

-*Để đơn giản ta tiếp cận với bài toán phân lớp nhị phân trên dữ liệu phân chia tuyến tính* :

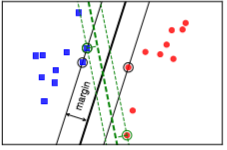
Cho một tập dữ liệu phân tách tuyến tính (linearly separable) :

Xây dựng siêu phẳng (hyperplane) :

Sao cho siêu phẳng tìm được có khoảng cách giữa nó và những điểm dữ liệu gần nhất với nó hai bên siêu phẳng là bằng nhau và lớn nhất khi so với các siêu phẳng còn lại . Khoảng cách này còn được gọi là *margin* .



Việc *margin* trở nên rộng hơn , sẽ mang lại hiệu ứng phân lớp tốt hơn vì sự phân chia lúc 2 lớp lúc này trở nên rạch ròi . Từ đó ta nhận thấy rằng bài toán cần giải trong SVM là bài toán đi tìm siêu phẳng sao cho *margin* giữa hai lớp là lớn nhất .



Nhận xét quan trọng:  
Với bất kỳ 1 cặp dữ liệu (,) nào thì khoảng cách từ điểm đó tới siêu phẳng là :

(1)

Và , ta có phương trình siêu phẳng : nhân thêm vào 2 vế : ta có siêu phẳng mới : có dạng giông như dạng siêu phẳng ban đầu vì vậy ta có thể chon w và b sao cho :

(2)

Với là điểm trong tập dữ liệu mà gần với siêu phẳng nhất , các điểm này được gọi là support vector , vì nhờ nó mà ta có thể tìm được siêu phẳng .

Vậy với mọi ta luôn có :

Từ (1) ta suy ra , công thức *margin* :

Bài toán tối ưu của SVM chính là việc tìm ra w và b sao cho margin đạt giá trị lớn nhất :

(3)

Từ (1) và (2) với mọi n , ta có :

Vậy bài toán số (3) ta có thể đưa về bài toán tối ưu có rang buộc dạng :

Thỏa mãn :n= 1,2,..,m

Bằng biến đổi đơn giản :

Thỏa mãn :n= 1,2,..,m

**Lưu ý** ở đây , ta nghịch đảo hàm mục tiêu , bình phương để phương trình trở thành dạng bài toán toàn phương (Quadratic Problem) , việc đưa về phương trình toàn phương không làm thay đổi nghiệm của bài toán tối ưu này , đồng thời khiến việc giải toán sau này trở nên dễ dàng hơn .

# **4 . Giải bài toán tối ưu:**

**Phương pháp nhân tử Lagrange:**

-*Với một bài toán ràng buộc chỉ là phương trình* :

Thỏa mãn : , i= 1,2,..,l

(4)

Xét hàm số Lagrange :

được gọi là nhân tử Lagrange .   
Người ta đã chứng minh rằng , điểm optimal value của bài toán (4) thỏa mãn điều kiện , điều này tương đương với :

i= 1,2,..,l

-*Mở rông với một bài toán ràng buộc có cả phương trình và bất phương trình* :

Thỏa mãn : , i= 1,2,..,l

, i= 1,2,..,k

Ta có hàm Lagrange :

được gọi là nhân tử Lagrange

-*Đối ngẫu trong Lagrange* :

Giả sử ta định nghĩa một hàm:

Từ các điều kiện ràng buộc ta nhận thấy :

Nên khi và thì :

Suy ra :

Ta định nghĩa giá trị tối ưu của :

Và bài toán này được gọi là bài toán gốc .

Ta tiếp tục xem xét thêm một bài toán :

Ta có :

Ta định nghĩa giá trị tối ưu của

Một nhân xét cực kì quan trọng : giá trị của lớn nhất của bài toán đối ngẫu luôn luôn nhỏ hơn hoặc bằng giá trị nhỏ nhất của bài toán gốc:

(5)

Điều đó có được là do tính chất của một hàm số bất kì max min min max

Ở (5) dấu ‘=’ thường xảy ra khi đảm bảo các điều kiện dưới đây :  
 Hàm và là hàm số lồi , và là hàm số tuyến tính .  
 Hàm , i= 1,2,..,k (**s**later)  
Đây là hiện tưởng Strong Duality xảy ra , khi đó tồn tại các giá trị là lời giải của bài toán gốc ,, là lời giải của bài toán đối ngẫu

Và các nghiệm này thỏa mãn điều kiện Karush-Kuhn-Tucker (KKT) được giải thích rõ hơn ở bên dưới .

**Bài toán đối ngẫu SVM:**

**-**Nhắc lại bài toán  **:**

Thỏa mãn :n= 1,2,..,m

Ta có :

Hàm Lagrange có dạng :

(6)

Đạo hàm hàm Lagrange theo từng biến ta được :

Suy ra :

Thay hai phương trình vào (6) , rút gọn ta thu được :

Ta phát biểu lại bài toán huấn luyện SVM trở thành bài toán tìm sao cho :

Thỏa mãn : i= 1,2,..,m

Đây được gọi là hàm đối ngẫu Wolfe**.**

Lợi ích chính cùa bài toán đối ngẫu là hàm mục tiêu W chỉ phụ thuộc vào các nhân tử Lagrange .

*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

Sau khi giải được bài toán đối ngẫu , ta thu được vector chứa tập các thừa số Lagrange :

Ta tính w theo công thức:

Ta tính b theo công thức:

**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

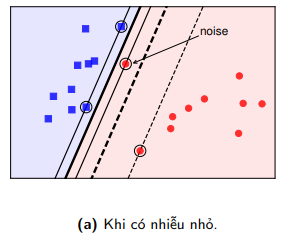
Tuy nhiên bài toán trên có cách tíếp cận nặng nề cũng như không thực tế khi làm việc với tập dữ liệu lớn bởi tiêu tốn khá nhiều thời gian và yêu cầu một lượng bộ nhớ đáng kể , vì vậy chung ta không đi sâu thêm vào cách giải này .

# **5 . Soft Margin :**

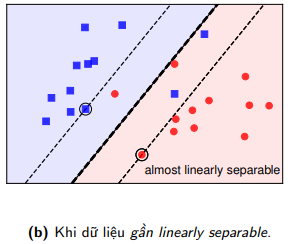
Support vector machine (SVM) thuần chỉ làm việc khi dữ liệu của hai lớp là linearly separable. Một cách tự nhiên, chúng ta cũng mong muốn rằng SVM có thể làm việc với dữ liệu gần linearly separable giống như logistic regression đã làm được .

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 nhưng có một điểm nhiễu của lớp đỏ ở quá gần so với lớp 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 với các điểm ở lớp xanh và quá xa các điểm thuộc lớp 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 . Trong trường hợp này, không tồn tại đường thẳng nào hoàn thoàn phân chia hai lớp dữ liệu, vì vậy bài toán tối ưu SVM trở nên vô nghiệm. Tuy nhiên, nếu chịu hy sinh một chút những điểm ở gần khu vực biên giới giữa hai lớp, 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 lấn sang phía bên kia của các đường suport tương ứng, ta gọi điểm đó rơi vào vùng không an toàn.



Để có một margin lớn hơn trong soft margin SVM, ta cần hy sinh một vài điểm dữ liệu bằng cách chấp nhận cho chúng rơi vào vùng không an toàn. Tất nhiên, việc hy sinh này cần được hạn chế, nếu không, 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 cũng như tối thiểu sự hy sinh

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:

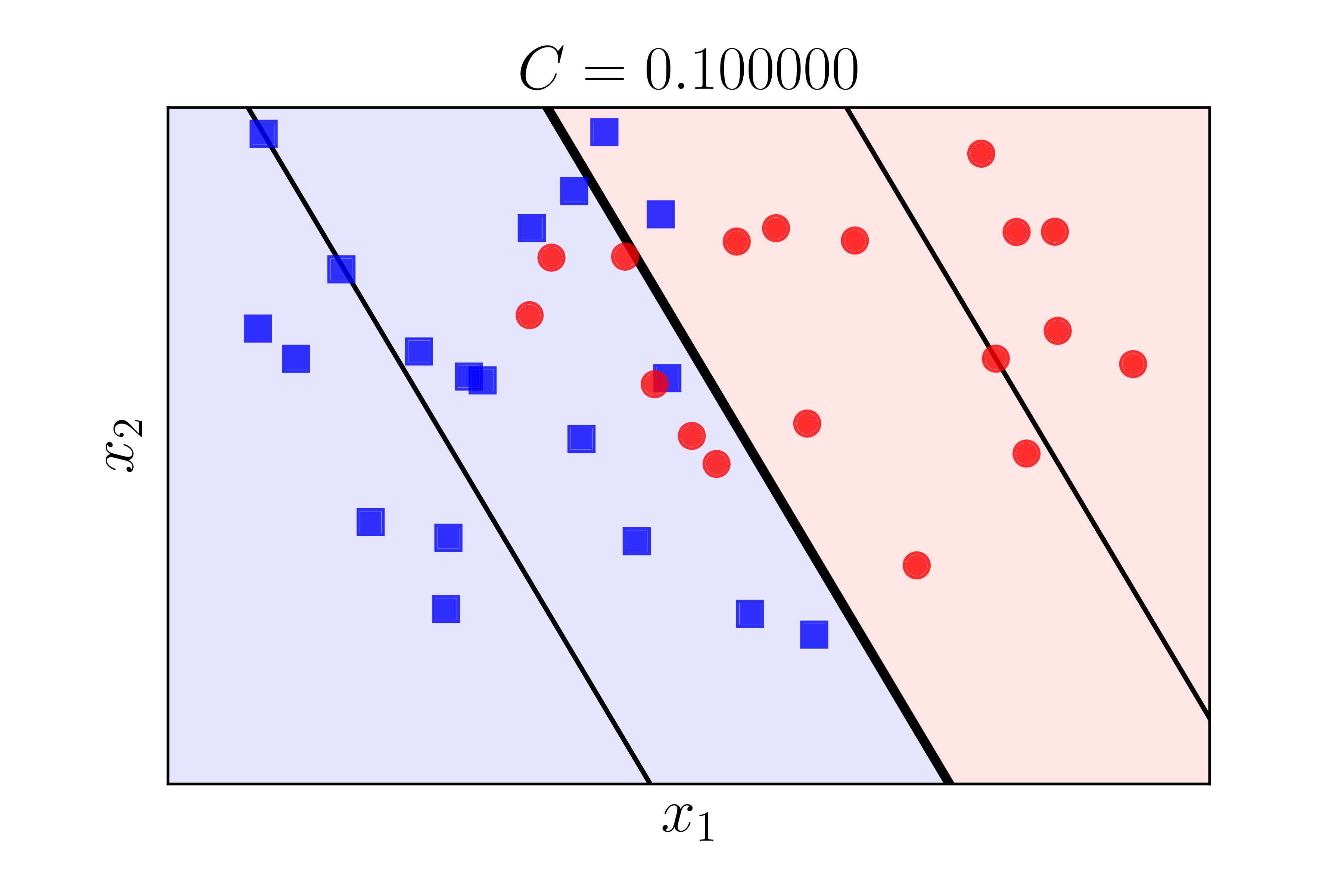
Để tránh việc tối thiểu hóa bằng giá trị âm, ta thêm điều kiện vào. Ngoài ra để kiểm soát soft margin, ta sử dụng tham số C, xác định tầm quan trọng của zeta

Soft margin function:

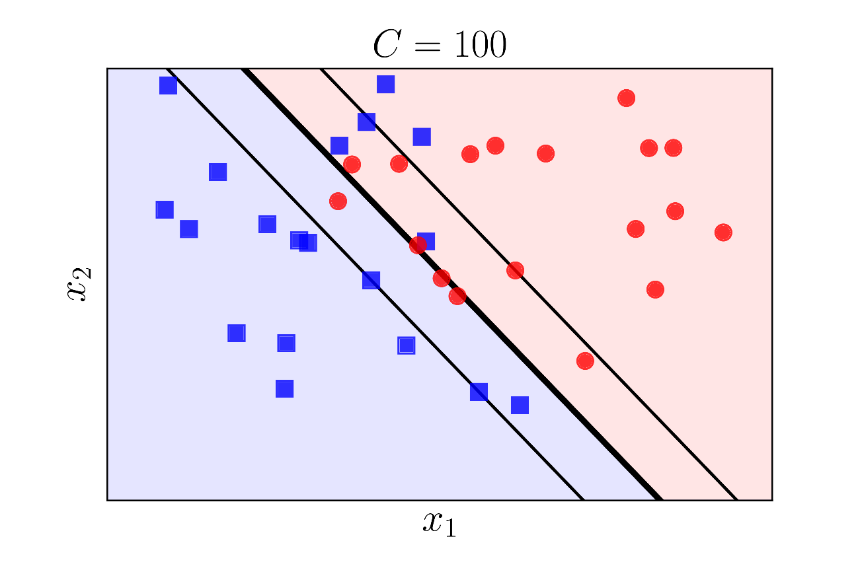
Tương tự, SVM nguyên bản (hard SVM) ta có được bài toán đối ngẫu với một chút khác biệt về ràng buộc:

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)*

# **6 . Kernel SVM**

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,…

# **7 . Mutil – SVM**

Từ đầu đến giờ ta chỉ việc phân lớp giữa hai lớp dữ liệu , vậy để phân chia dữ liệu có nhiều lớp ta cần phải làm thế nào ?

Ta cần mở rộng SVM với 2 kỹ thuật phổ biến , One Versert One (OVO), One Versert Rest (OVR):

OVO : giả sử số lớp cần phân chia là m lớp vậy ta cần xây dựng m\*(m-1)/2 bộ phân lớp dữ liệu , đôi một giữa các lớp với nhau , khi có một dữ liệu mới vào ta sử dụng từng lớp phân chia một phân loại rồi đưa ra kết luận .

OVR : giả sử số lớp cần phân chia là m lớp ta chỉ cần xây dựng đúng m bộ phân lớp , giữa một lớp và các lớp còn lại , khi có dữ liệu mới vào , ta xét khoảng cách mà dữ liệu đó cách phân lớp của lớp nào xa nhất thì ta chọn lớp đó .

Đánh giá về 2 kĩ thuật mở rộng SVM cho ta dễ dàng nhận thấy OVO có khả năng sẽ phân lớp tốt hơn OVR vì có nhiều số lớp xây dựng để xét dữ liệu mới , tuy nhiên về hiệu năng hoạt động khi train model cũng như xét dữ liệu mới OVR nhanh hơn OVO rất nhiều nếu số lớp cần phân lớn OVO sẽ càng chậm

# **8 . Thuật toán SMO**

SMO (**sequential minimal optimization**) : tối ưu hóa tuần tự

SMO giải bài toán tối ưu sau:

Sở dĩ gọi nó là tối ưu hóa tuần tự vì 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.

# **9 . Áp dụng SVM cho nhận dạng chữ số viết tay :**

Bộ dữ liệu huấn luyện : MNIST

Link : <https://www.kaggle.com/c/digit-recognizer>

Ngôn ngữ lập trình : python

MNIST bao gồm hai tập con: tập dữ liệu huấn luyện (training set) có tổng cộng 48k ví dụ khác nhau về chữ số viết tay từ 0 đên 9, tập dữ liệu kiểm tra (test set) có 12k ví dụ khác nhau. Tất cả đều đã được gán nhãn.

**Áp dụng thuật toán SMO**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Lần 1 | Lần 2 | Lần 3 |
| Số mẫu | 840 | 2100 | 4200 |
| Thời gian | 15p | 60p | 1h30 |
| Result trên bộ Test | 45% | 55% | 60% |

Nhận xét chủ quan : do phần cứng yếu nên tốc độ train chậm , nhận thấy sự đúng đắn tăng sau khi train nhiều bộ test dần , bản thân người nghiên cứu có khả năng nghiên cứu còn yếu nên chưa cải thiện , cũng như làm chủ được thuật toán .

Tài liệu tham khảo

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

[https://machinelearningcoban.com](https://machinelearningcoban.com/)/

<http://cs229.stanford.edu/materials/smo.pdf>/

<http://cs229.stanford.edu/notes/cs229-notes3.pdf>/