**Using SVM For Time Series Prediction**

1. Support vector regression (SVR)

Ý tưởng cơ bản là ánh xạ dữ liệu x vào không gian đặc trưng có số chiều lớn **Ғ** (high-dimensional feature space) bằng một *ánh xạ phi tuyến* **φ** rồi thực hiện hồi quy tuyến tính trong không gian này:

ƒ(x) = ( ω.φ(x) ) + b với φ : Rn 🡪 Ғ, ω є Ғ, [1.1]

và b là ngưỡng (threshold).

Vì vậy, hồi quy tuyến tính trên không gian có số chiều lớn tương ứng với hồi quy phi tuyến trên không gian đầu vào Rn (không gian có số chiều nhỏ). Nếu ta không thể dùng thủ thuật tính toán cho kernel thì tích vô hướng ω.φ(x) phải được tính trực tiếp trên không gian Ғ này. Khi ánh xạ φ đã được cố định, ta phải tìm ω từ dữ liệu bằng cách tối thiểu hóa tổng của giá trị rủi ro kinh nghiệm (empirical risk) Remp[ƒ] và ||ω2||. Điều này làm tăng tính phẳng(flatness) trong không gian đặc trưng.

Rreg[ƒ] = Remp[ƒ] + λ|| ω2|| = , [1.2]

Với l là kích thước tập mẫu (x1, …, xl), C(.) là hàm chi phí, và λ là hằng tiêu chuẩn (regularization constant). Đối với tập hợp hàm chi phí, đẳng thức [1.2] có thể được tối thiểu hóa bằng cách giải quyết vấn đề cực trị của hàm bậc hai (quadratic programming problem).

Ngoài ra, vector ω có thể được tính thông qua các điểm dữ liệu như sau:

ω = [1.3]

chính là giải pháp cho vấn đề cực trị hàm bậc 2 trình bày ở trên. Ngoài ra, có thể được hiểu một cách trực quan là nó giúp điều chỉnh giá trị tiến gần yi . Từ [1.1] và [1.3] ta có thể suy ra vấn đề trên không gian đầu vào Rn như sau:

ƒ(x) = [1.4]

Trong đó, k(xi, xj) = (φ(xi).φ(xj)) là hàm kernel. Và bất kỳ hàm kernel đối xứng nào cũng phải thỏa định lý Mercer tương ứng với một tích vô hướng trong một không gian đặc trưng nào đó. Thông thường đó là RBF kernel (còn gọi là Gaussian kernel):

k(x,y) = exp(-||x – y||2/ (2σ2))

1. Vapnik’s ε-insensitive Loss Function

Đối với hàm chi phí này, những số nhân Lagrange αi, αi\* thường “thưa thớt” tức là chúng khác 0 sau khi tối ưu hóa nếu và chỉ nếu chúng vượt khỏi cận (outside of the boundary). Điều đó có nghĩa là chúng thỏa mãn điều kiện Karush-Kuhn-Tucker. Hàm chi phí ε-insensitive được cho bởi

C(ƒ(x) – y) = [1.5]

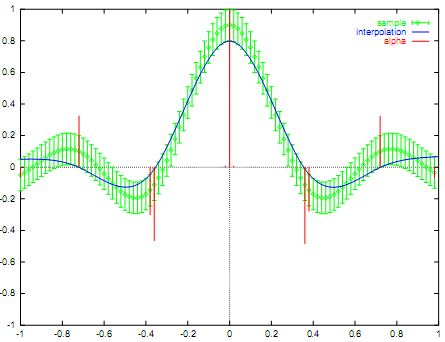
Vấn đề quadratic programming được định ra như sau

Tối thiểu hóa

Với

[1.6]

Càng ít nhiễu, αi, αi\* càng “thưa thớt”. Ngoài ra, nếu ε lớn, chúng sẽ có xu hướng không khớp khi hồi quy, đặc biệt nếu ε quá lớn thì kết quả hồi quy sẽ trở thành hằng.



*Đây là hồi quy trong trường hợp ε-insensitive (dùng kernel B-splines) của hàm sinc. Nó thể hiện tính phẳng tối đa với ε tube được áp xung quanh dữ liệu. Những điểm biên, nơi mà ƒ(xi) – yi = ε sign(αi – αi\*) sẽ được sử dụng để tính ngưỡng b.*

1. Huber’s Loss Function

Ưu điểm của hàm này là không đưa thêm bất kỳ độ lệch (bias) nào cũng giống như ε-insensitive. Tuy nhiên, αi và αi\* không còn “thưa thớt” nữa.

C(ƒ(x) – y) = [1.7]

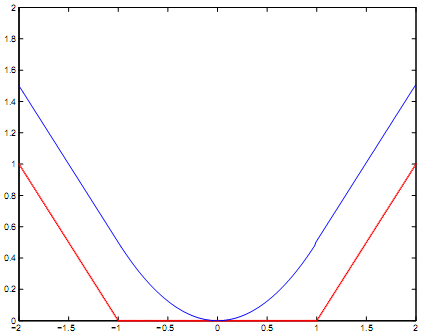
Vấn đề quadratic programming sẽ có dạng

Tối thiểu hóa

Với

[1.8]

Như vậy, về cơ bản, tất cả mẫu đều là support vector.



*ε-insensitive và Huber’s loss function với ε = 1*

1. Cách tính ngưỡng b

Phương trình [1.6] và [1.8] dùng để tính αi và αi\*. Để chọn b phù hợp, ta phải sử dụng một cách trực tiếp hơn điều kiện Karush-Kuhn-Tucker để đưa đến vấn đề quadratic programming đã trình bày ở trên. Điểm mấu chốt là chọn được αk, αk\* sao cho độ lỗi δk = ƒ(xk) – yk có thể được xác định một cách độc lập. Trong trường hợp *ε-insensitive,* điều này có nghĩa là chọn những điểm xk trên biên với điều kiện αk, αk\* nằm trong khoảng mở (0;1/𝛌). Trong trường hợp đó, ta được

δk *= ε sign(αi – αi\*)*

Về cơ bản, một giá trị xk đủ để giúp ta xác định b nhưng để vững chắc hơn, ta nên lấy trung bình tất cả những điểm biên với

b = averagek { δk + yk - }

Cho trường hợp của Huber, b được tính tương tự với

δk *=* 𝛌*(αi – αi\*)* với αk, αk\* thuộc về [0; ε/𝛌) nghĩa là những điểm mà phần toàn phương (bậc 2) của hàm chi phí xác định

Chú ý rằng khi ta giải quyết vấn đề quadratic programming với một *bộ tối ưu* để tính toán giá trị double dual ta có thể tìm ra giá trị *biến cơ bản* b.