« Bài 18: Duality (/2017/04/02/duality/)
Bài 20: Soft Margin Support Vector Machine » (/2017/04/13/softmarginsmv/)

Bài 19: Support Vector Machine

Linear-models (/tags#Linear-models) Classification (/tags#Classification)

Apr 9, 2017

Trong loạt bài tiếp theo, tôi sẽ trình bày về một trong những thuật toán classification phổ biến nhất (cùng với softmax regression (/2017/02/17/softmax/)). Có rất nhiều suy luận toán học trong phần này yêu cầu bạn cần có kiến thức về Duality (/2017/04/02/duality/) cũng như về tối ưu lồi. Bạn được khuyến khích đọc các Bài 16, 17, và 18 trước khi đọc bài này.

Nếu không muốn đi sâu vào phần toán, bạn có thể bỏ qua mục 3.

Trong trang này:

1. Giới thiệu

Trước khi đi vào phần ý tưởng chính của Support Vector Machine, tôi xin một lần nữa nhắc lại kiến thức về hình học giải tích mà chúng ta đã quá quen khi ôn thi đại học.

1.1. 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 (x_0, y_0) tới *đường thẳng* có phương trình $w_1x + w_2y + b = 0$ được xác định bởi:

$$\frac{|w_1x_0 + w_2y_0 + b|}{\sqrt{w_1^2 + w_2^2}}$$

Trong không gian ba chiều, khoảng cách từ một điểm có toạ độ (x_0, y_0, z_0) tới một *mặt phẳng* có phương trình $w_1x + w_2y + w_3z + b = 0$ được xác định bởi:

$$\frac{|w_1x_0 + w_2y_0 + w_3z_0 + b|}{\sqrt{w_1^2 + w_2^2 + w_3^2}}$$

Hơn nữa, nếu ta 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 (tôi tạm gọi đây là *phía dương* của đường thẳng), những điểm làm cho biểu thức trong dấu giá trị tuyệt đối mang dấu âm nằm về phía còn lại (tôi gọ 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.

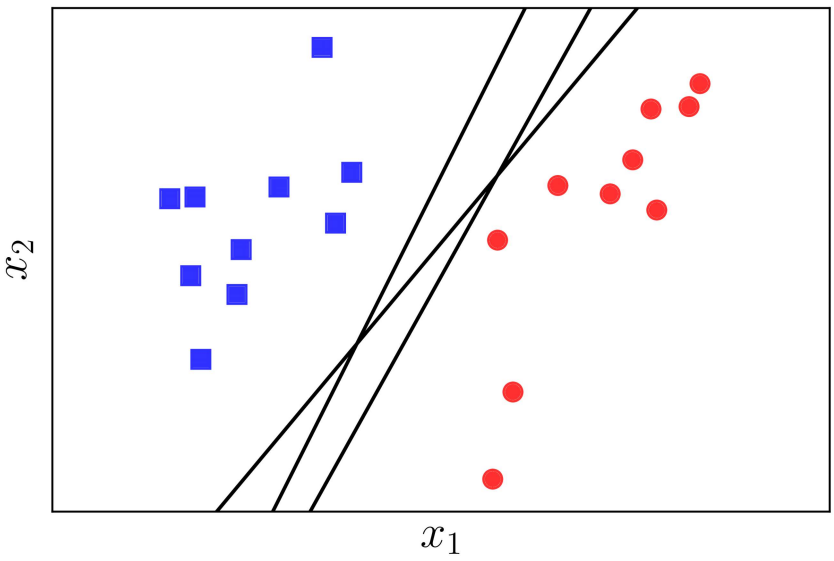
Việc này có thể được tổng quát lên không gian nhiều chiều: Khoảng cách từ một điểm (vector) có toạ độ $\mathbf{x}_0$ tới *siêu mặt phẳng* (*hyperplane*) có phương trình $\mathbf{w}^T\mathbf{x} + b = 0$ được xác định bởi:

$$\frac{|\mathbf{w}^T\mathbf{x}_0 + b|}{||\mathbf{w}||_2}$$

Với $||\mathbf{w}||_2 = \sqrt{\sum_{i=1}^{d} w_i^2}$ với d là số chiều của không gian.

1.2. Nhắc lại bài toán phân chia hai classes

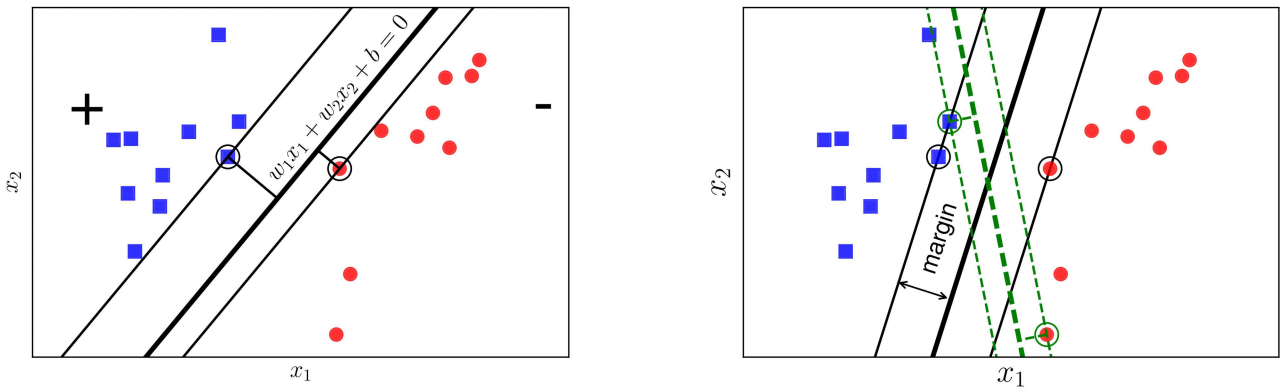
Chúng ta cùng quay lại với bài toán trong Perceptron Learning Algorithm (PLA) (/2017/01/21/perceptron/). Giả sử rằng có hai class khác nhau được mô tả bởi các điểm trong không gian nhiều chiều, hai classes này *linearly separable*, tức tồn tại một siêu phẳng phân chia chính xác hai classes đó. Hãy tìm một siêu mặt phẳng phân chia hai classes đó, tức tất cả các điểm thuộc một class nằm về cùng một phía của siêu mặt phẳng đó và ngược phía với toàn bộ các điểm thuộc class còn lại. Chúng ta đã biết rằng, thuật toán 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 *theo một tiêu chuẩn nào đó*? Trong ba đường thẳng minh họa trong Hình 1 phía trên, có hai đường thẳng khá *lệch* về phía class hình tròn đỏ. Điều này có thể khiến cho lớp màu đỏ *không vui vì lãnh thổ xem ra bị lấn nhiều quá*. Liệu có cách nào để tìm được đường phân chia mà cả hai classes đều cảm thấy *công bằng* và *hạnh phúc* nhất hay không?

Chúng ta cần tìm một tiêu chuẩn để đo sự *hạnh phúc* của mỗi class. Hãy xem Hình 2 dưới đây:



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

Nếu ta định nghĩa *mức độ hạnh phúc* của một class tỉ lệ thuận với khoảng cách gần nhất từ một điểm của class đó tới đường/mặt phân chia, thì ở Hình 2 trái, class tròn đỏ sẽ *không được hạnh phúc cho lắm* vì đường phân chia gần nó hơn class vuông xanh rất nhiều. 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, như thế thì mới *công bằng*. Khoảng cách như nhau này được gọi là *margin* (*lề*).

Đã có *công bằng* rồi, chúng ta cần *văn minh* nữa. *Công bằng* mà cả hai đều *kém hạnh phúc như nhau* thì chưa phải là *văn mình* cho lắm.

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

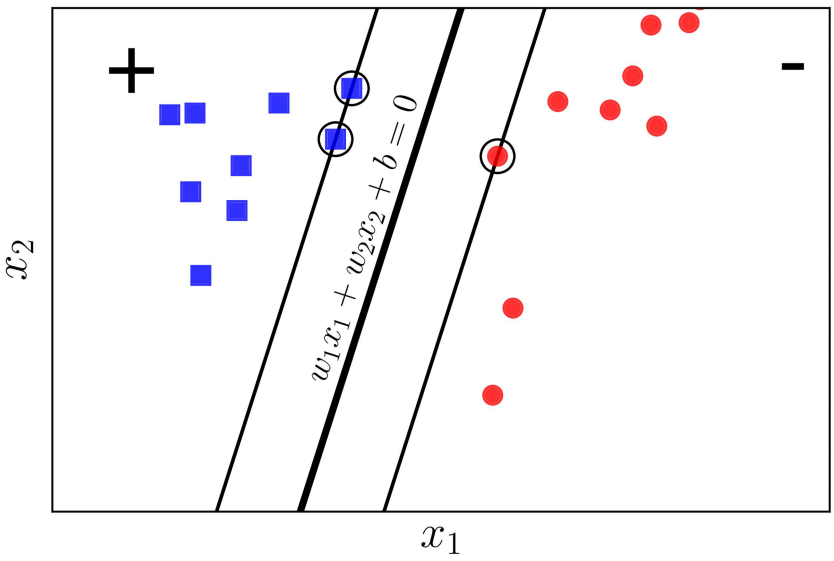
Việc *margin* rộng hơn sẽ mang lại hiệu ứng phân lớp tốt hơn vì *sự phân chia giữa hai classes là rạch ròi hơn*. Việc này, sau này các bạn sẽ thấy, là một điểm khá quan trọng giúp *Support Vector Machine* mang lại kết quả phân loại tốt hơn so với *Neural Network với 1 layer*, tức Perceptron Learning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

Bài toán tối ưu trong *Support Vector Machine* (SVM) chính là bài toán đi tìm đường phân chia sao cho *margin* là lớn nhất. Đây cũng là lý do vì sao SVM còn được gọi là *Maximum Margin Classifier*. Nguồn gốc của tên gọi Support Vector Machine sẽ sớm được làm sáng tỏ.

2. Xây dựng bài toán tối ưu cho SVM

Giả sử rằng các cặp dữ liệu của *training set* là $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$ với vector $\mathbf{x}_i \in \mathbb{R}^d$ thể hiện *đầu vào* của một điểm dữ liệu và y_i là *nhãn* của điểm dữ liệu đó. d là số chiều của dữ liệu và N là số điểm dữ liệu. Giả sử rằng *nhãn* của mỗi điểm dữ liệu được xác định bởi $y_i = 1$ (class 1) hoặc $y_i = -1$ (class 2) giống như trong PLA.

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



Hình 3: Phân tích bài toán SVM.

Giả sử rằng các điểm vuông xanh thuộc class 1, các điểm tròn đỏ thuộc class -1 và mặt $\mathbf{w}^T\mathbf{x} + b = w_1x_1 + w_2x_2 + b = 0$ 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 $\mathbf{w}$ và b. Chú ý rằng chúng ta cần đi tìm các hệ số $\mathbf{w}$ và b.

Ta quan sát thấy một điểm quan trọng sau đây: với cặp dữ liệu $(\mathbf{x}_n, y_n)$ bất kỳ, khoảng cách từ điểm đó tới mặt phân chia là:

$$\frac{y_n(\mathbf{w}^T\mathbf{x}_n + b)}{||\mathbf{w}||_2}$$

Điều này có thể dễ nhận thấy vì theo giả sử ở trên, y_n luôn cùng dấu với *phía* của $\mathbf{x}_n$. Từ đó suy ra y_n cùng dấu với $(\mathbf{w}^T\mathbf{x}_n + b)$, 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):

$$\text{margin} = \min_n \frac{y_n(\mathbf{w}^T\mathbf{x}_n + b)}{||\mathbf{w}||_2}$$

Bài toán tối ưu trong SVM chính là bài toán tìm $\mathbf{w}$ và b sao cho *margin* này đạt giá trị lớn nhất:

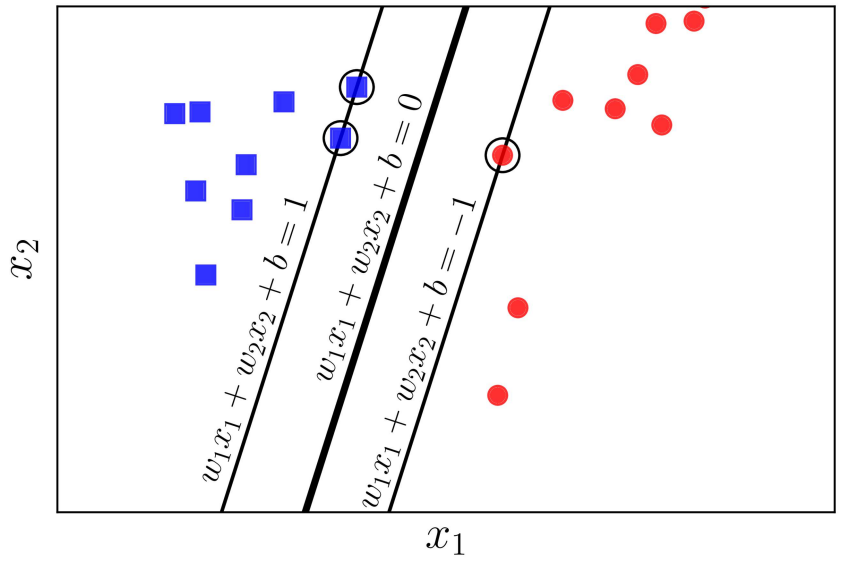
$$(\mathbf{w}, b) = \arg\max_{\mathbf{w},b} \left\{ \min_n \frac{y_n(\mathbf{w}^T\mathbf{x}_n + b)}{||\mathbf{w}||_2} \right\} = \arg\max_{\mathbf{w},b} \left\{ \frac{1}{||\mathbf{w}||_2} \min_n y_n(\mathbf{w}^T\mathbf{x}_n + b) \right\} \quad (1)$$

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 $\mathbf{w}$ bởi $k\mathbf{w}$ và b bởi kb trong đó k là một hằng số dương thì mặt phân chia không thay đổi, tức khoảng cách từ từng điểm đến mặt phân chia không đổi, tức *margin* không đổi. Dựa trên tính chất này, ta có thể giả sử:

$$y_n(\mathbf{w}^T\mathbf{x}_n + b) = 1$$

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:

$$y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$$

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

$$(\mathbf{w}, b) = \arg\max_{\mathbf{w},b} \frac{1}{||\mathbf{w}||_2}$$
$$\text{subject to:} \quad y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1, \forall n = 1, 2, \ldots, N \quad (2)$$

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

$$(\mathbf{w}, b) = \arg\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||_2^2$$
$$\text{subject to:} \quad 1 - y_n(\mathbf{w}^T\mathbf{x}_n + b) \leq 0, \forall n = 1, 2, \ldots, N \quad (3)$$

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

Quan sát quan trọng: Trong bài toán (3), hàm mục tiêu là một norm, nên là một hàm lồi (/2017/03/12/convexity/#-norms). Các hàm bất đẳng thức ràng buộc là các hàm tuyến tính theo $\mathbf{w}$ 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 (/2017/03/19/convexopt/#-quadratic-programming). Thậm chí, hàm mục tiêu là *strictly convex* vì $||\mathbf{w}||_2^2 = \mathbf{w}^T\mathbf{I}\mathbf{w}$ và $\mathbf{I}$ 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 (/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 (/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.

Đến đây, bạn đọc có thể bắt đầu hiểu tại sao tôi cần viết 3 bài 16-18 trước khi viết bài này. Nếu bạn muốn hiểu sâu hơn về SVM, tôi khuyến khích đọc Mục 3 dưới đây. Nếu không, bạn có thể sang Mục 4 để xem ví dụ về cách sử dụng SVM khi lập trình.

Xác định class cho một điểm dữ liệu mới: Sau khi tìm được mặt phân cách $\mathbf{w}^T\mathbf{x} + b = 0$, class của bất kỳ một điểm nào sẽ được xác định đơn giản bằng cách:

$$\text{class}(\mathbf{x}) = \text{sgn}(\mathbf{w}^T\mathbf{x} + b)$$

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

3. 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 (/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 (/2017/04/02/duality/#-kkt-optimality-conditions).

3.1. 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 $\mathbf{w}, b$ thoả mãn:

$$1 - y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0, \;\; \forall n = 1, 2, \ldots, N$$

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 $(\mathbf{w}_0, b_0)$ sao cho:

$$\begin{aligned} & 1 - y_n(\mathbf{w}_0^T\mathbf{x}_n + b_0) \leq 0, \;\; \forall n = 1, 2, \ldots, N \\ \Leftrightarrow \; & 2 - y_n(2\mathbf{w}_0^T\mathbf{x}_n + 2b_0) \leq 0, \;\; \forall n = 1, 2, \ldots, N \end{aligned}$$

Vậy chỉ cần chọn $\mathbf{w}_1 = 2\mathbf{w}_0$ và $b_1 = 2b_0$, ta sẽ có:

$$1 - y_n(\mathbf{w}_1^T\mathbf{x}_n + b_1) \leq -1 < 0, \;\; \forall n = 1, 2, \ldots, N$$

Từ đó suy ra điều kiện Slater thoả mãn.

3.2. Lagrangian của bài toán SVM

Lagrangian (/2017/04/02/duality/#-lagrangian) của bài toán (3) là:

$$\mathcal{L}(\mathbf{w}, b, \lambda) = \frac{1}{2}||\mathbf{w}||_2^2 + \sum_{n=1}^{N}\lambda_n(1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)) \qquad (4)$$

với $\lambda = [\lambda_1, \lambda_2, \ldots, \lambda_N]^T$ và $\lambda_n \geq 0, \; \forall n = 1, 2, \ldots, N$.

3.3. Hàm đối ngẫu Lagrange

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

$$g(\lambda) = \min_{\mathbf{w}, b} \mathcal{L}(\mathbf{w}, b, \lambda)$$

với $\lambda \succeq 0$.

Việc tìm giá trị nhỏ nhất của hàm này theo $\mathbf{w}$ và b có thể được thực hiện bằng cách giải hệ phương trình đạo hàm của $\mathcal{L}(\mathbf{w}, b, \lambda)$ theo $\mathbf{w}$ và b bằng 0:

$$\frac{\partial \mathcal{L}(\mathbf{w}, b, \lambda)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{n=1}^{N} \lambda_n y_n \mathbf{x}_n = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \lambda_n y_n \mathbf{x}_n \quad (5)$$

$$\frac{\partial \mathcal{L}(\mathbf{w}, b, \lambda)}{\partial b} = -\sum_{n=1}^{N} \lambda_n y_n = 0 \quad (6)$$

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

$$g(\lambda) = \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \mathbf{x}_n^T \mathbf{x}_m \quad (7)$$

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

Xét ma trận:

$$\mathbf{V} = [y_1 \mathbf{x}_1, y_2 \mathbf{x}_2, \ldots, y_N \mathbf{x}_N]$$

và vector $\mathbf{1} = [1, 1, \ldots, 1]^T$, ta có thể viết lại $g(\lambda)$ dưới dạng:

$$g(\lambda) = -\frac{1}{2} \lambda^T \mathbf{V}^T \mathbf{V} \lambda + \mathbf{1}^T \lambda. \quad (8)$$

(*Nếu khó tin, bạn có thể viết ra để quen dần với các biểu thức đại số tuyến tính.*)

Đặt $\mathbf{K} = \mathbf{V}^T \mathbf{V}$, ta có một quan sát quan trọng: $\mathbf{K}$ là một ma trận nửa xác định dương (/2017/03/12/convexity/#positive-semidefinite). Thật vậy, với mọi vector λ, ta có:

$$\lambda^T \mathbf{K} \lambda = \lambda^T \mathbf{V}^T \mathbf{V} \lambda = ||\mathbf{V}\lambda||_2^2 \geq 0.$$

(*Đây chính là định nghĩa của ma trận nửa xác định dương.*)

Vậy $g(\lambda) = -\frac{1}{2} \lambda^T \mathbf{K} \lambda + \mathbf{1}^T \lambda$ là một hàm *concave* (/2017/03/12/convexity/#concave-function).

3.4. 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 (/2017/04/02/duality/#-bai-toan-doi-ngau-lagrange-the-lagrange-dual-problem):

$$\begin{aligned} \lambda = \arg\max_{\lambda} \; & g(\lambda) \\ \text{subject to:} \quad & \lambda \succeq 0 \quad (9) \\ & \sum_{n=1}^{N} \lambda_n y_n = 0 \end{aligned}$$

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* (/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 $\mathbf{w}$ 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. Tuy nhiên, điều hấp dẫn của bài toán đối ngẫu này đến từ phần *Kernel Support Vector Machine (Kernel SVM)*, tức cho các bài toán mà dữ liệu không phải là *linearly separable* hoặc *gần linearly separable*. Phần *Kernel SVM* sẽ được tôi trình bày sau 1 hoặc 2 bài nữa. Ngoài ra, dựa vào tính chất đặc biệt của hệ điều kiện KKT mà SVM có thể được giải bằng nhiều phương pháp hiệu quả hơn.

3.5. 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 (/2017/04/02/duality/#-kkt-optimality-conditions) sau đây với biến số là $\mathbf{w}, b$ và λ:

$$1 - y_n(\mathbf{w}^T\mathbf{x}_n + b) \leq 0, \ \forall n = 1, 2, \ldots, N \quad (10)$$

$$\lambda_n \geq 0, \ \forall n = 1, 2, \ldots, N$$

$$\lambda_n(1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)) = 0, \ \forall n = 1, 2, \ldots, N \quad (11)$$

$$\mathbf{w} = \sum_{n=1}^{N} \lambda_n y_n \mathbf{x}_n \quad (12)$$

$$\sum_{n=1}^{N} \lambda_n y_n = 0 \quad (13)$$

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 $\lambda_n = 0$ hoặc $1 - y_n(\mathbf{w}^T\mathbf{x}_n + b) = 0$. Trường hợp thứ hai chính là:

$$\mathbf{w}^T\mathbf{x}_n + b = y_n \quad (14)$$

với chú ý rằng $y_n^2 = 1, \ \forall n$.

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 $\mathbf{w}^T\mathbf{x}_n + b = \pm 1$ *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.

Một quan sát khác, 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. Nhìn theo một cách khác, hầu hết các λ_n bằng 0. Vậy là mặc dù vector $\lambda \in \mathbb{R}^N$ 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.

Tiếp tục phân tích, 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 $\lambda_n = 0$ hoặc $\lambda_n \neq 0$. Tổng số trường hợp phải xét là 2^N. Với $N > 50$ (thường là như thế), đây là một con số rất lớn, giải bằng cách này sẽ không khả thi. Tôi sẽ không đi sâu tiếp vào việc giải hệ KKT như thế nào, trong phần tiếp theo chúng ta sẽ giải bài toán tối ưu (9) bằng CVXOPT và bằng thư viện `sklearn`.

Sau khi tìm được λ từ bài toán (9), ta có thể suy ra được $\mathbf{w}$ dựa vào (12) và b dựa vào (11) và (13). Rõ ràng ta chỉ cần quan tâm tới $\lambda_n \neq 0$.

Gọi tập hợp $\mathcal{S} = \{n : \lambda_n \neq 0\}$ và $N_\mathcal{S}$ là số phần tử của tập $\mathcal{S}$. Với mỗi $n \in \mathcal{S}$, ta có:

$$1 = y_n(\mathbf{w}^T\mathbf{x}_n + b) \Leftrightarrow b + \mathbf{w}^T\mathbf{x}_n = y_n$$

Mặc dù từ chỉ một cặp $(\mathbf{x}_n, y_n)$, ta có thể suy ra ngay được b nếu đã biết $\mathbf{w}$, một phiên bản khác để tính b thường được sử dụng và được cho là *ổn định hơn trong tính toán* (*numerically more stable*) là:

$$b = \frac{1}{N_\mathcal{S}}\sum_{n \in \mathcal{S}}(y_n - \mathbf{w}^T\mathbf{x}_n) = \frac{1}{N_\mathcal{S}}\sum_{n \in \mathcal{S}}\left(y_n - \sum_{m \in \mathcal{S}}\lambda_m y_m \mathbf{x}_m^T\mathbf{x}_n\right) \quad (15)$$

tức trung bình cộng của mọi cách tính b.

Trước đó, $\mathbf{w}$ đã được tính bằng:

$$\mathbf{w} = \sum_{m \in \mathcal{S}}\lambda_m y_m \mathbf{x}_m \quad (16)$$

theo (12).

Quan sát quan trọng: Để xác định một điểm $\mathbf{x}$ mới thuộc vào class nào, ta cần xác định dấu của biểu thức:

$$\mathbf{w}^T\mathbf{x} + b = \sum_{m \in \mathcal{S}}\lambda_m y_m \mathbf{x}_m^T\mathbf{x} + \frac{1}{N_\mathcal{S}}\sum_{n \in \mathcal{S}}\left(y_n - \sum_{m \in \mathcal{S}}\lambda_m y_m \mathbf{x}_m^T\mathbf{x}_n\right)$$

Biểu thức này phụ thuộc vào cách tính tích vô hướng giữa các cặp vector $\mathbf{x}$ và từng $\mathbf{x}_n \in \mathcal{S}$. Nhận xét quan trọng này sẽ giúp ích cho chúng ta trong bài Kernal SVM.

4. Lập trình tìm nghiệm cho SVM

Trong mục này, tôi sẽ trình bày hai cách tính nghiệm cho SVM. Cách thứ nhất dựa theo bài toán (9) và các công thức (15) và (16). Cách thứ hai sử dụng trực tiếp thư viện `sklearn`. Cách thứ nhất chỉ là để chứng minh nãy giờ tôi không *viết nhảm*, bằng cách minh hoạ kết quả tìm được và so sánh với nghiệm tìm được bằng cách thứ hai.

4.1. 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*):

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
np.random.seed(22)

means = [[2, 2], [4, 2]]
cov = [[.3, .2], [.2, .3]]
N = 10
X0 = np.random.multivariate_normal(means[0], cov, N) # class 1
X1 = np.random.multivariate_normal(means[1], cov, N) # class -1
X = np.concatenate((X0.T, X1.T), axis = 1) # all data
y = np.concatenate((np.ones((1, N)), -1*np.ones((1, N))), axis = 1) # labels
```

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

```
from cvxopt import matrix, solvers
# build K
V = np.concatenate((X0.T, -X1.T), axis = 1)
K = matrix(V.T.dot(V)) # see definition of V, K near eq (8)

p = matrix(-np.ones((2*N, 1))) # all-one vector
# build A, b, G, h
G = matrix(-np.eye(2*N)) # for all lambda_n >= 0
h = matrix(np.zeros((2*N, 1)))
A = matrix(y) # the equality constrain is actually y^T lambda = 0
b = matrix(np.zeros((1, 1)))
solvers.options['show_progress'] = False
sol = solvers.qp(K, p, G, h, A, b)

l = np.array(sol['x'])
print('lambda = ')
print(l.T)
```

Kết quả:

```
lambda =
 [[  8.54018321e-01   2.89132533e-10   1.37095535e+00   6.36030818e-10
    4.04317408e-10   8.82390106e-10   6.35001881e-10   5.49567576e-10
    8.33359230e-10   1.20982928e-10   6.86678649e-10   1.25039745e-10
    2.22497367e+00   4.05417905e-09   1.26763684e-10   1.99008949e-10
    2.13742578e-10   1.51537487e-10   3.75329509e-10   3.56161975e-10]]
```

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* $\mathcal{S}$ rồi tìm nghiệm của bài toán:

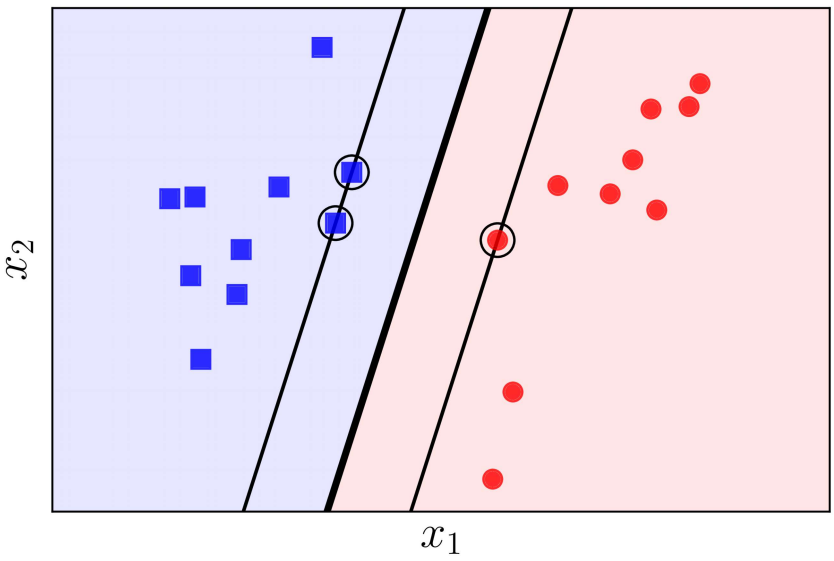
```
epsilon = 1e-6 # just a small number, greater than 1e-9
S = np.where(l > epsilon)[0]

VS = V[:, S]
XS = X[:, S]
yS = y[:, S]
lS = l[S]
# calculate w and b
w = VS.dot(lS)
b = np.mean(yS.T - w.T.dot(XS))

print('w = ', w.T)
print('b = ', b)
```

```
w =  [[-2.00984381  0.64068336]]
b =  4.66856063387
```

Minh hoạ kết quả:



Hình 5: Minh hoạ nghiệm tìm được bởi SVM.

Đường màu đen đậm ở giữa chính là mặt phân cách tìm được bằng SVM. Từ đây có thể thấy *nhiều khả năng là các tính toán của ta là chính xác*. Để kiểm tra xem các tính toán phía trên có chính xác không, ta cần tìm nghiệm bằng các công cụ có sẵn, ví dụ như `sklearn`.

Source code cho phần này có thể được tìm thấy ở đây (https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/19_svm/plt/SVM-example.ipynb).

4.2. 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.

Nếu dùng thư viện thì sẽ như sau:

```
from sklearn.svm import SVC

y1 = y.reshape((2*N,))
X1 = X.T # each sample is one row
clf = SVC(kernel = 'linear', C = 1e5) # just a big number

clf.fit(X1, y1)

w = clf.coef_
b = clf.intercept_
print('w = ', w)
print('b = ', b)
```

```
w =  [[-2.00971102  0.64194082]]
b =  [ 4.66595309]
```

Kết quả này khá giống với kết quả chúng ta tìm được ở phần trên. Có rất nhiều tuỳ chọn cho SVM, các bạn sẽ dần thấy trong các bài sau.

5. Tóm tắt và thảo luận

- Với bài toán binary classification mà 2 classes là *linearly separable*, có vô số các siêu mặt phẳng giúp phân biệt hai classes, tức mặt phân cách. Với mỗi mặt phân cách, ta có một *classifier*. Khoảng cách gần nhất từ 1 điểm dữ liệu tới mặt phân cách ấy được gọi là *margin* của classifier đó.
- Support Vector Machine là bài toán đi tìm mặt phân cách sao cho *margin* tìm được là lớn nhất, đồng nghĩa với việc các điểm dữ liệu *an toàn nhất* so với mặt phân cách.
- Bài toán tối ưu trong SVM là một bài toán lồi với hàm mục tiêu là *stricly convex*, nghiệm của bài toán này là duy nhất. Hơn nữa, bài toán tối ưu đó là một Quadratic Programming (QP).
- Mặc dù có thể trực tiếp giải SVM qua bài toán tối ưu gốc này, thông thường người ta thường giải bài toán đối ngẫu. Bài toán đối ngẫu cũng là một QP nhưng nghiệm là *sparse* nên có những phương pháp giải hiệu quả hơn.
- Với các bài toán mà dữ liệu *gần linearly separable* hoặc *nonlinear separable*, có những cải tiền khác của SVM để thích nghi với dữ liệu đó. Mời bạn đón đọc bài tiếp theo.
- Source code (https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/19_svm/plt/SVM-example.ipynb).

6. Tài liệu tham khảo

[1] Bishop, Christopher M. “Pattern recognition and Machine Learning.”, Springer (2006). (book (http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf))

[2] Duda, Richard O., Peter E. Hart, and David G. Stork. Pattern classification. John Wiley & Sons, 2012.

[3] `sklearn.svm.SVC` (http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)

[4] LIBSVM – A Library for Support Vector Machines (https://www.csie.ntu.edu.tw/~cjlin/libsvm/)

Nếu có câu hỏi, Bạn có thể để lại comment bên dưới hoặc trên Forum (https://www.facebook.com/groups/257768141347267/) để nhận được câu trả lời sớm hơn.
Bạn đọc có thể ủng hộ blog qua 'Buy me a cofee' (/buymeacoffee/) ở góc trên bên trái của blog.
Tôi vừa hoàn thành cuốn ebook 'Machine Learning cơ bản', bạn có thể đặt sách tại đây (/ebook/). Cảm ơn bạn.

« Bài 18: Duality (/2017/04/02/duality/)
Bài 20: Soft Margin Support Vector Machine » (/2017/04/13/softmarginsmv/)