Bài tập 03

Classification

Thông tin cá nhân

Họ và tên	Vũ Lê Thế Anh	Nguyễn Lê Hồng Hạnh
MSSV	1612838	1612849
Email	{1612838, 1612849}@student.hcmus.edu.vn	
SĐT	0961565087	0902719551

Yêu cầu bài tập

STT	Yêu cầu	Hoàn thành
1	Thực hành: báo cáo kết quả chạy tập dữ liệu mushroom.arff trên weka	100%
2	Thực hành: báo cáo kết quả chạy tập dữ liệu zoo.arff trên weka	100%
3	Thực hành: báo cáo kết quả chạy tập dữ liệu letter.arff trên weka	100%
4	Lý thuyết: tìm hiểu một giải thuật phân lớp	100%

Mục lục

1 Báo cáo với ứng dụng WEKA

1.1 Tập dữ liệu mushroom

1. Bảng 1 thống kê độ chính xác phân lớp của mỗi giải thuật trên dữ liệu được cho.

Giải thuật	Accuracy	Detailed Accuracy By Class							
		Class edible				Class poisonous			
		TPR	FP	Pre	Rec	TPR	FPR	Prec	Rec
LR	99.7537%	0.998	0.003	0.998	0.998	0.997	0.002	0.997	0.997
J48	99.8768%	1.000	0.003	0.998	1.000	0.997	0.000	1.000	0.997
IBk (KNN = 1)	99.8768%	0.998	0.000	1.000	0.998	1.000	0.002	0.997	1.000
IBk (KNN = 4)	99.1379%	0.988	0.005	0.995	0.988	0.995	0.012	0.988	0.995

Bảng 1: Thống kê độ chính xác phân lớp của mỗi giải thuật (TPR: True Positive Rate, FPR: False Positive Rate, Pre: Precision, Rec: Recall)

2. Đối với bài toán nhận diện nấm độc, trên thực tế, việc cho nhầm nấm độc vào nhóm nấm ăn được nguy hiểm nghiêm trọng hơn so với việc xác định một nấm ăn được là nấm độc. Do đó, đối với bài toán này, ta cần chú trọng tỉ lệ FP (False Positive) của lớp "edible" hơn là tỉ lệ TP (True Positive). Hay nói cách khác, thuật giải nào có tỉ lệ FP cho "edible" càng thấp thì thuật giải đó càng nên ưu tiên hơn. Trong 2 thuật giải A và B, thì thuật giải B tốt hơn, vì có tỉ lệ FP là 0.

1.2 Tập dữ liệu zoo

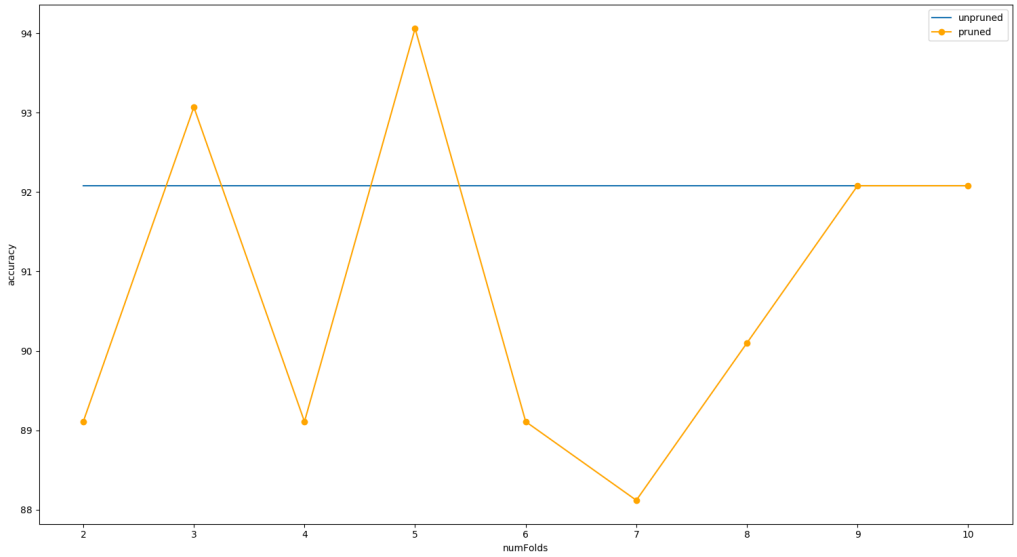
3. Đồ thị biểu diễn độ chính xác phân lớp theo sự biến thiên của numFolds được cho ở hình 1.

4. **numFolds** cho biết tỉ lệ lượng dữ liệu được sử dụng để cắt tỉa. Ví dụ, con số 3 mặc định có nghĩa là thuật toán sẽ chia dữ liệu thành 3 phần và sử dụng 1 phần để tỉa nhánh, 2 phần còn lại để sinh cây tiếp tục.

   Nếu con số này quá lớn, phần để tỉa nhánh nhỏ, mô hình sẽ phức tạp và khó hiểu, dễ bị quá khớp (overfit) với tập dữ liệu huấn luyện.

   Nếu con số này quá nhỏ, phần để tỉa nhánh lớn, mô hình sẽ trở nên quá đơn giản (sử dụng ít thông tin để sinh cây, tỉa bớt nhiều,...) và đồng thời và không đủ khớp (underfit) với tập dữ liệu huấn luyện.

5. Nhìn chung, độ chính xác không ổn định, phần lớn không vượt qua trường hợp không cắt tỉa và thay đổi thất thường khi numFolds tăng. Độ chính xác chỉ lên cao hơn so với không cắt tỉa với một số giá trị nhỏ (cụ thể là 3 và 5).



Hình 1: Biểu đồ thể hiện độ chính xác theo sự biến thiên numFolds

Việc tỉa nhánh ở một chừng mực nào đó sẽ giúp giảm độ phức tạp của mô hình và nhờ đó giúp mô hình ít bị "overfit" với dữ liệu huấn luyện hơn nên độ chính xác trên tập kiểm tra sẽ tăng (nói cách khác, lỗi tổng quát hóa của mô hình sẽ giảm, hay mô hình có tính tổng quát cao hơn).

Như nhận xét thì các giá trị numFolds nhỏ (tương ứng cắt tỉa nhiều) có xu hướng cho độ chính xác cao hơn, nhưng nếu nhỏ quá thì dễ không đủ khớp (như trường hợp 2 có lẽ là đã cắt quá nhiều) còn nếu lớn quá thì dễ bị quá khớp (như trường hợp từ 6 trở đi). Việc chia dữ liệu thành các bộ cũng dễ bị ảnh hưởng bởi việc chia ngẫu nhiên, giải thích sự thay đổi thất thường với các numFolds khác nhau.

1.3 Tập dữ liệu letter

6. Thuật toán được chọn là **LibSVM** với độ chính xác: 97.6438%. Bảng 2 thể hiện thống kê độ chính xác của mô hình.

7. Trong xử lý hình ảnh có một kỹ thuật sử dụng đặc trưng toàn cục để biểu diễn một bức ảnh đó là đặc trưng moment, tức là các giá trị trung bình (moment bậc 1), phương sai (moment bậc 2),... Các thuộc tính trong tập dữ liệu này có vẻ cũng sử dụng ý tưởng tương tự, do đó có thể xem vector với các phần tử là giá trị từng thuộc tính như một điểm trong không gian và sử dụng các thuật toán trên điểm (vận dụng các độ đo khoảng cách,...). Một trong những thuật toán đó là SVM (Support Vector Machine).

   Trước khi Neural Network đạt hiệu quả như hiện tại (mà trong đó phần lớn là nhờ sự nâng cấp phần cứng có thể tạo ra các kiến trúc "sâu" hơn) thì SVM có thể được xem là một trong những thuật toán hiệu quả nhất. Thuật toán cụ thể sẽ được trình bày ở

Class	TP Rate	FP Rate	Precision	Recall
A	0.998	0.000	0.997	0.998
B	0.967	0.002	0.947	0.957
C	0.983	0.001	0.986	0.985
D	0.969	0.002	0.949	0.959
E	0.972	0.002	0.961	0.967
F	0.966	0.001	0.971	0.968
G	0.964	0.001	0.976	0.970
H	0.930	0.002	0.943	0.937
I	0.974	0.001	0.977	0.975
J	0.961	0.001	0.975	0.968
K	0.962	0.002	0.956	0.959
L	0.977	0.000	0.990	0.983
M	0.987	0.001	0.981	0.984
N	0.984	0.001	0.978	0.981
O	0.974	0.001	0.977	0.975
P	0.964	0.001	0.979	0.971
Q	0.989	0.001	0.980	0.984
R	0.954	0.002	0.958	0.956
S	0.992	0.000	0.995	0.993
T	0.983	0.000	0.994	0.988
U	0.992	0.000	0.989	0.991
V	0.984	0.001	0.976	0.980
W	0.989	0.000	0.993	0.991
X	0.987	0.001	0.982	0.985
Y	0.992	0.001	0.985	0.988
Z	0.988	0.000	0.990	0.989

Bảng 2: Các thống kê kết quả khi sử dụng LibSVM

phần sau của báo cáo, ở đây xin sơ lược ý tưởng của nó.

SVM là thuật toán phân lớp nhị phân, tức phân chia các đối tượng vào 1 trong 2 lớp định sẵn. SVM thực hiện việc phân lớp này bằng cách tìm ra một siêu phẳng phân tách, hình dung như một biên giới mà tập hợp các điểm thuộc 1 lớp nằm ở một nửa còn các điểm thuộc lớp kia nằm ở nửa còn lại. Siêu phẳng này được đánh giá trên tiêu chí cách xa nhất có thể với các điểm từ cả hai tập, tạo thành một khu vực trống (không có điểm nào từ cả hai tập) lớn nhất ở lân cận siêu phẳng.

Loại SVM cụ thể sử dụng ở đây là C-SVC, là một bộ phân lớp sử dụng một tham số C là mức độ cho phép một điểm được vượt biên (đi vào trong khu vực trống). Ngoài ra

Tên	Giá trị
SVMType	C-SVC
cost	100
kernelType	radial basis function
probabilityEstimates	True

Bảng 3: Các tham số cần chú ý

nó còn sử dụng một hàm nhân để tạo thành các biên giới phi tuyến tính.

Bộ phân lớp nhị phân cũng có thể giải quyết bài toán phân lớp nhiều lớp, ở đây sử dụng chiến thuật "một chọi một" tức xây dựng một bộ phân lớp cho từng cặp lớp ($n(n-1)/2$ bộ với n lớp).

Sau khi thử nghiệm nhiều giải thuật khác nhau trên Weka thì nhận thấy SVM đã chứng tỏ được vị thế của mình với kết quả vượt trội.

8. Các tham số quan trọng có thể thử thay đổi so với cấu hình mặc định là `SVMType`, `cost`, `kernelType` và `probabilityEstimates`.

   Thay đổi `SVMType` tức thay đổi loại thuật toán SVM sử dụng. Như trên trình bày, loại được chọn là `C-SVC` là một trong những thuật cơ bản, dễ hiểu và hiệu quả.

   Thay đổi `kernelType` sẽ thay đổi hàm nhân sử dụng. Ở đây khuyến khích sử dụng hàm `RBF`. Hàm này giúp ánh xạ dữ liệu vào không gian vector vô hạn chiều giúp cho việc tìm ra siêu phẳng phân tách được mở rộng và kết quả tốt hơn. Một lý do khác là hàm (bản chất là hàm Gaussian) này có tính chất làm trơn nên sẽ ít bị ảnh hưởng bởi nhiễu.

   Thay đổi `cost` sẽ thay đổi tham số C. Khi C tăng, thuật toán sẽ nghiêm ngặt hơn trong việc ngăn cản các điểm dữ liệu vượt biên, tuy nhiên sẽ không tốt nếu tăng quá nhiều vì sẽ làm cho mô hình bị cứng và dễ ảnh hưởng bởi nhiễu. Tham số này được chọn là 100 theo cách thử và sai dựa vào kết quả của 10-fold cross validation (kết quả giảm dần khi thấp hơn và đứng yên khi tăng lên).

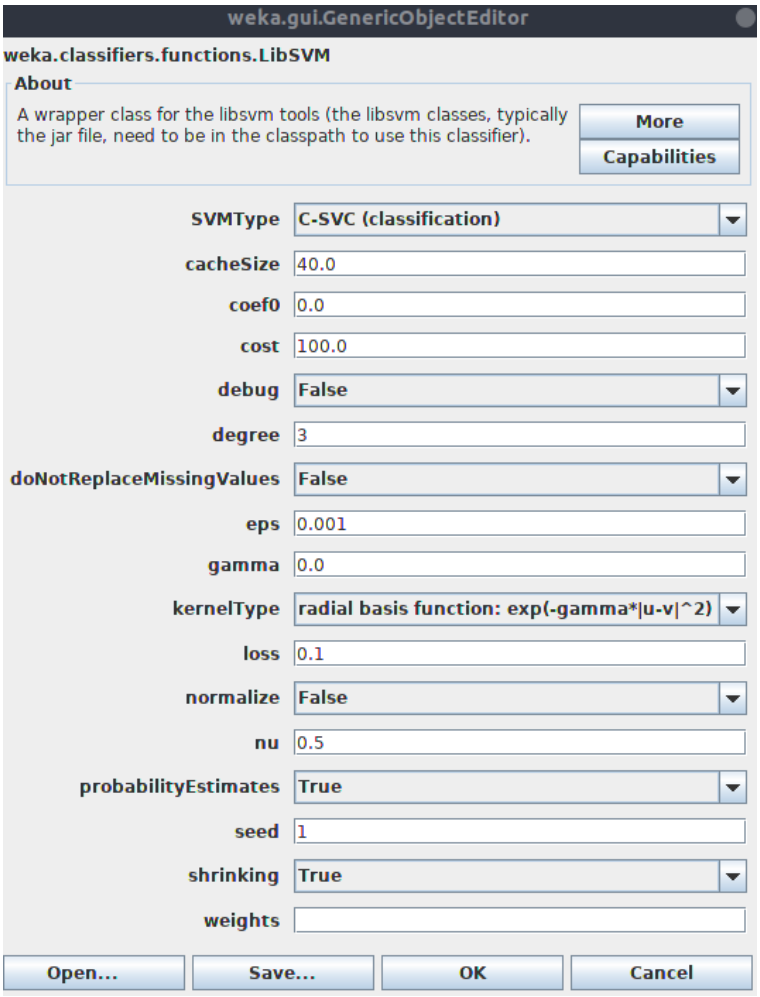
   Ngoài ra còn có tham số `probabilityEstimates` thể hiện rằng đầu ra sẽ là phân lớp (nhị phân) hay xác suất thuộc về một lớp. Việc tính xác suất có lợi hơn (kết quả trơn hơn), nên đặt về `True`.

   Như vậy so với cấu hình mặc định, một số tham số cần lưu ý được cho ở bảng 3.

   Cụ thể bộ tham số sử dụng được cho ở ở hình 2.

2 Tìm hiểu thuật toán

Support vector machine là một **bộ phân lớp nhị phân tuyến tính phi xác suất**, tức nó là một giải thuật máy học nhằm giải quyết **bài toán phân lớp nhị phân** bằng cách tìm một **siêu phẳng phân tách** mà **cơ sở không dựa trên xác suất**. Tuy nhiên, các cải tiến hay biến thể của thuật toán có thể bỏ đi một hoặc nhiều trong các tính chất kể trên, ví dụ như Kernel



Hình 2: Tham số chạy thuật toán LibSVM trên bộ dữ liệu letter.

SVM cũng tìm ra siêu phẳng nhưng trong một không gian đặc trưng khác mà khi ánh xạ ngược lại không gian đặc trưng ban đầu thì mô hình kết quả có thể là một hàm phi tuyến với các đặc trưng đầu vào.

Ở đây trình bày đầu tiên mô hình SVM cơ bản là **Hard-margin SVM** (SVM lề cứng), sau đó là **Soft-margin SVM** (SVM lề mềm) và cuối cùng là **Kernel SVM** (SVM nhân).

2.1 Khái niệm

Ta nhắc lại một số khái niệm. Đầu tiên là về bài toán **phân lớp nhị phân**. Đây là một bài toán thuộc lớp bài toán học có giám sát, tức hệ thống sẽ nhận vào một tập dữ liệu huấn luyện đã có gán nhãn sẵn $\mathcal{D} = \{(\vec{x}, y)/\vec{x} \in \mathbb{R}^n, y \in \{-1, 1\}\}$ kích thước $|\mathcal{D}| = N$. Trong đó, $\vec{x}$ là vector đặc trưng và y là nhãn. Do đây là bài toán phân lớp nhị phân nên y chỉ có thể nhận giá trị là -1 hoặc 1, ứng với hai lớp khác nhau, từ đây gọi là lớp Pos và lớp Neg.

Một **siêu phẳng** trong không gian n chiều là không gian con $n-1$ chiều của không gian đó. Ví dụ trong không gian 2 chiều, một siêu phẳng là một đường thẳng, còn trong không gian 3 chiều, một siêu phẳng là một mặt phẳng. Phương trình siêu phẳng là $(H) : \vec{w}^T\vec{x} + b = 0$

với $\vec{w}$ là vector pháp tuyến của siêu phẳng và b mang ý nghĩa độ dời theo phương của $\vec{w}$. Khoảng cách từ một điểm $\vec{x}$ đến siêu phẳng là

$$d(\vec{x}, H) = \frac{|\vec{w}^T\vec{x} + b|}{\|\vec{w}\|}.$$

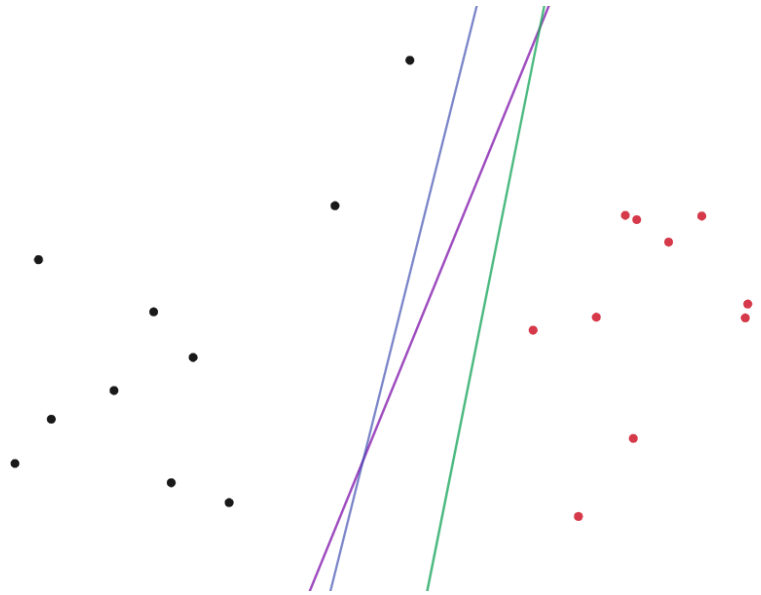
Một khái niệm cần lưu ý ở đây là việc một tập dữ liệu huấn luyện có tính **phân tách tuyến tính** (linearly seperable), hiểu đơn giản là có tồn tại một siêu phẳng chia không gian thành hai nửa không gian sao cho tất cả điểm thuộc một lớp nằm về một phía còn tất cả điểm thuộc một lớp khác nằm về phía còn lại. Về mặt toán học, hai tập hợp Pos và Neg được gọi là phân tách tuyến tính khi tồn tại bộ $\vec{w} \in \mathbb{R}^n$ và $b \in \mathbb{R}$ sao cho

$$\begin{cases} \vec{w}^T\vec{x} + b > 0 & \vec{x} \in Pos \\ \vec{w}^T\vec{x} + b < 0 & \vec{x} \in Neg \end{cases}.$$

Khi đó, $\vec{w}^T\vec{x} + b = 0$ chính là phương trình siêu phẳng phân tách.

2.2 SVM lề cứng

Đối với một bộ dữ liệu phân tách tuyến tính, có thể tìm được vô số siêu phẳng phân tách (xem Hình 3), tạo thành một **tập khả thi** (feasible set). Như vậy, cần phải có một tiêu chuẩn để quyết định xem siêu phẳng nào trong tập là ta mong muốn nhất.

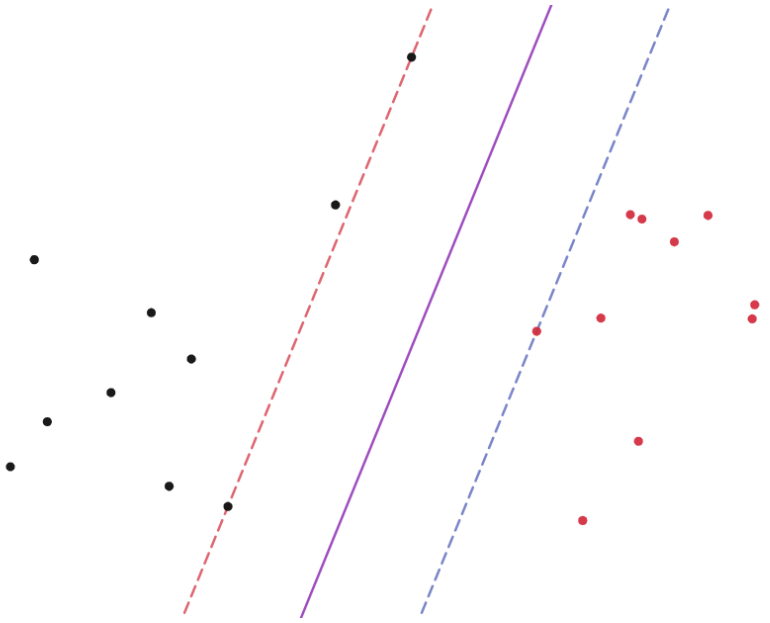


Hình 3: Nhiều đường thẳng làm tốt việc "phân tách".

2.2.1 Bài toán gốc

Đối với giải thuật SVM, siêu phẳng được chọn là siêu phẳng có kích thước **lề** (margin) lớn nhất. Kích thước lề ở đây là khoảng cách đến siêu phẳng của điểm dữ liệu gần siêu phẳng

nhất. (xem Hình 4)



Hình 4: Siêu phẳng tìm thấy là siêu phẳng có kích thước lề lớn nhất.

Như vậy, giải thuật SVM có mục tiêu tìm ra $(\vec{w}, b)$ để giải bài toán tối ưu

$$(\vec{w}, b) = \arg\max_{(\vec{w},b)} \min_{(x_i,y_i)\in\mathcal{D}} \frac{y_i(\vec{w}^T\vec{x_i} + b)}{\|\vec{w}\|}.$$

Ở đây đã gán nhãn $y_i = 1$ cho $\vec{x_i} \in Pos$ và $y = -1$ cho các điểm thuộc $\vec{x_i} \in Neg$. Do đó, $|\vec{w}^T\vec{x_i} + b| = y_i(\vec{w}^T\vec{x_i} + b)$.

Một nhận xét lúc này là nếu $(\vec{w}, b)$ là một nghiệm của bài toán tối ưu trên thì $(k\vec{w}, kb)$ với $k \in \mathbb{R}$ cũng là một nghiệm bởi chúng mô tả cùng một siêu phẳng. Do đó, không mất tính tổng quát ta có thể xét điều kiện $\min_{(x_i,y_i)\in\mathcal{D}} = y_i(\vec{w}^T\vec{x_i} + b) = 1, \forall i \in [1, N]$ và bài toán trở thành

$$(\vec{w}, b) = \arg\max_{(\vec{w},b)} \frac{1}{\|\vec{w}\|} = \arg\min_{(\vec{w},b)} \frac{1}{2}\|\vec{w}\|^2$$

$$y_i(\vec{w}^T\vec{x_i} + b) \geq 1, \forall i \in [1, N]$$

Việc giải bài toán tối ưu trên sẽ cho ta siêu phẳng cần tìm. Khi đó, với mỗi điểm dữ liệu mới $\vec{x}$, ta có thể phân lớp cho điểm dữ liệu này bằng cách tính $\vec{w}^T\vec{x} + b$ và kiểm tra dấu của nó. Nếu dương, điểm mới này thuộc lớp Pos và ngược lại thuộc lớp Neg nếu âm.

2.2.2 Bài toán đối ngẫu

Bài toán trên thuộc về lớp bài toán quy hoạch toàn phương (quadratic programming), hoàn toàn có thể giải được sử dụng các công cụ tối ưu, ví dụ như thư viện cvxopt của Python. Tuy nhiên, ta sẽ khảo sát một hướng giải khác được ưa chuộng hơn, vì hai lý do. Thứ nhất, nó cho ta một bài tối ưu có thể giải hiệu quả hơn việc giải bài toán tối ưu trên. Thứ hai, nó cho

ta khả năng sử dụng hàm nhân (kernel, sẽ nói rõ ở phần SVM nhân) hiệu quả. Cách tiếp cận này như sau.

Xét hàm Lagrangian với nhân tử Lagrange $\vec{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_N), \lambda_i \geq 0$.

$$\mathcal{L}(\vec{w}, b, \vec{\lambda}) = \frac{1}{2}\|\vec{w}\|^2 + \sum_{i=1}^{N} \lambda_i [1 - y_i(\vec{w}^T \vec{x_i} + b)]$$

Điểm tối ưu của Lagrangian với các λ_i cố định thỏa $\nabla_{w_j}\mathcal{L} = \nabla_b \mathcal{L} = 0$, tức

$$\vec{w} = \sum_{i=1}^{N} \lambda_i y_i \vec{x_i}; \sum_{i=1}^{N} \lambda_i y_i = 0$$

Thế vào phương trình ban đầu ta có

$$g(\vec{\lambda}) = \min_{(\vec{w},b)} \mathcal{L}(\vec{w}, b, \vec{\lambda}) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \lambda_i \lambda_j \langle \vec{x_i}, \vec{x_j} \rangle$$

với $\langle x, y \rangle = \vec{x}^T \vec{y}$ là tích vô hướng giữa hai vector.

Ta có bài toán đối ngẫu Lagrange

$$\max_{\vec{\lambda}} \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \lambda_i \lambda_j \langle \vec{x_i}, \vec{x_j} \rangle$$

$$\lambda_i \geq 0, \forall i \in [1, N]$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0$$

Thực tế, đây cũng chính là một bài toán quy hoạch toàn phương, nếu xét ma trận $V = [y_1\vec{x_1}, y_2\vec{x_2}, \ldots, y_N\vec{x_N}]$ và $\vec{1}_N = [1, 1, ..., 1]^T \in \mathbb{R}^N$, ta có thể viết nó lại thành như sau.

$$\max_{\vec{\lambda}} -\frac{1}{2}\vec{\lambda}^T V^T V \vec{\lambda} + 1^T \vec{\lambda}$$

$$-\lambda_i \leq 0, \forall i \in [1, N]$$

$$\sum_{i=1}^{N} y_i \lambda_i = 0$$

Giải quyết bài toán này bằng công cụ sẽ cho ta các giá trị λ_i cần thiết.

Từ $\vec{\lambda}$, ta có thể tính được $\vec{w}$ dựa vào phương trình

$$\vec{w} = \sum_{i=1}^{N} \lambda_i y_i \vec{x_i}$$

Một điều bỏ qua chưa nói ở đây là việc bài toán có tính chất thỏa mãn hệ điều kiện KKT,

trong đó có một đẳng thức quan trọng

$$\lambda_i(1 - y_i(\vec{w}^T\vec{x_i} + b)) = 0, \forall i \in [1, N]$$

Như vậy, với mọi điểm, hoặc là $\lambda_i = 0$, hoặc là $\vec{w}^T\vec{x_i} + b = y_i$ (do $y_i \in \{-1, 1\}$), hoặc cả hai. Một nhận xét quan trọng là nếu $\lambda_i > 0$ thì $y_i = \vec{w}^T\vec{x_i} + b$ tức điểm này nằm ngay trên lề, ta gọi những điểm này là **vector hỗ trợ**, và nó cũng khởi nguồn cho cái tên **máy vector hỗ trợ** (support vector machine). Về mặt hình ảnh, có thể thấy bộ dữ liệu trên thực tế của chúng ta thường có số lượng vector hỗ trợ khá ít, do đó $\lambda_i = 0$ sẽ khá nhiều tương ứng.

Từ một vector hỗ trợ bất kì, tức $y_i = \vec{w}^T\vec{x_i} + b$, ta cũng tính được b. Tuy nhiên, để kết quả ổn định, người ta thường tính b bằng cách lấy trung bình. Gọi S là tập các vector hỗ trợ.

$$b = \frac{1}{N_S}\sum_{i \in S}(y_i - \vec{w}^T\vec{x_i})$$

Một nhận xét quan trọng khác như sau.

$$\vec{w}^T\vec{x} = \left(\sum_{i=1}^{N}\lambda_i y_i \vec{x_i}\right)^T \vec{x} = \sum_{i=1}^{N}\lambda_i y_i \langle\vec{x_i}, \vec{x}\rangle = \sum_{i \in S}\lambda_i y_i \langle\vec{x_i}, \vec{x}\rangle$$

Như vậy, khi cần tính toán $\vec{w}^T\vec{x}$ chỉ cần tính với các vector hỗ trợ chứ không cần tính lại với toàn bộ điểm trong tập dữ liệu.

2.2.3 Tóm tắt thuật toán

Cho tập dữ liệu huấn luyện $\mathcal{D} = \{(\vec{x}, y)/\vec{x} \in \mathbb{R}^n, y \in \{-1, 1\}\}$ kích thước $|\mathcal{D}| = N$.

1. Đặt $V = [y_1\vec{x_1}, y_2\vec{x_2}, \ldots, y_N\vec{x_N}]$, $\vec{1}_N = [1, 1, ..., 1]^T \in \mathbb{R}^N$
2. Giải bài toán quy hoạch toàn phương để tìm $\vec{\lambda}$

$$\vec{\lambda} = \arg\max_{\vec{\lambda}} -\frac{1}{2}\vec{\lambda}^T V^T V\vec{\lambda} + \vec{1}^T\vec{\lambda}$$

$$-\lambda_i \leq 0, \forall i \in [1, N]$$

$$\sum_{i=1}^{N} y_i\lambda_i = 0$$

3. Xác định tập các vector hỗ trợ $S = \{i : \lambda_i > 0\}$.
4. Tìm $\vec{w}$ (có thể bỏ qua nếu không cần thiết)

$$\vec{w} = \sum_{i \in S}\lambda_i y_i \vec{x_i}$$

5. Tìm b

$$b = \frac{1}{N}\sum_{i \in S}(y_i - \sum_{j \in S}\lambda_j y_j \langle \vec{x_i}, \vec{x_j} \rangle)$$

Khi đó, ứng với mỗi điểm dữ liệu mới $\vec{x}$, kiểm tra dấu của

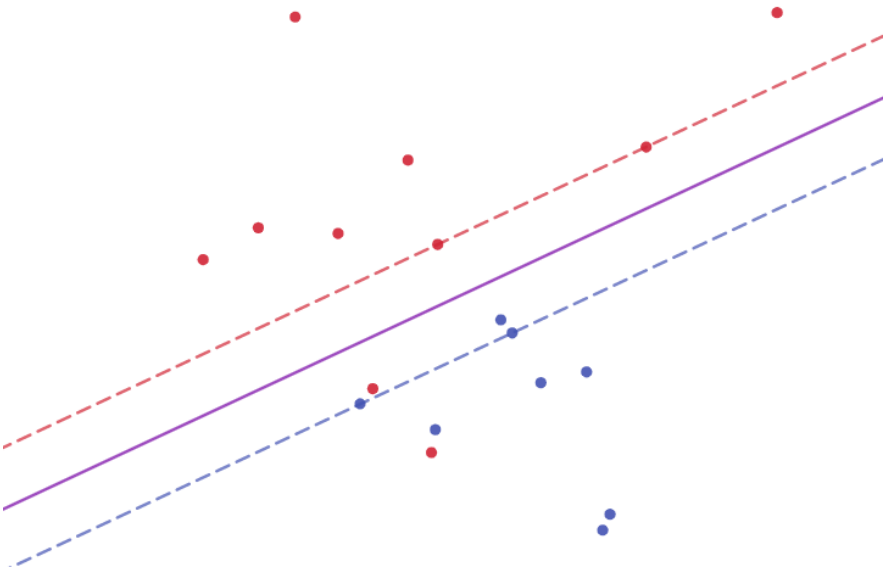
$$\hat{y} = \sum_{i \in S}\lambda_i y_i \langle \vec{x_i}, \vec{x} \rangle + b$$

Nếu $\hat{y} > 0$, gán nhãn $\vec{x} \in Pos$, ngược lại gán nhãn $\vec{x} \in Neg$.

2.3 SVM lề mềm

Phần trên nói về giải thuật SVM lề cứng là giải thuật nền tảng của SVM. Tuy nhiên, có hai khuyết điểm dễ nhận thấy của giải thuật này. Đầu tiên, nó chỉ có thể giải cho trường hợp hai lớp có tập dữ liệu phân tách tuyến tính với nhau. Thứ hai, giống như các giải thuật máy học "cứng" khác, SVM cũng gặp vấn đề với những dữ liệu nhiễu. Để giải quyết điều này, người ta đề xuất mô hình SVM lề mềm như sau.

Một cách hình ảnh, SVM lề cứng sẽ cho một vùng không gian trống giữa hai lề, trống ở đây theo nghĩa không có điểm dữ liệu nào nằm trong vùng này. Như vậy, nếu ta nới lỏng điều kiện và cho phép một số điểm dữ liệu được phép vượt qua lề và đánh lỗi cho siêu phẳng tìm được dựa trên mức độ "vượt rào" của các điểm. (xem Hình 5)



Hình 5: Siêu phẳng tìm thấy "du di" cho một số điểm.

Khi đó, hàm mục tiêu cần tối thiểu được thêm vào phần lỗi nói trên như sau

$$\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N}\alpha_i$$

với $C > 0$ là trọng số đánh lỗi, C càng lớn thể hiện mức độ nghiêm ngặt càng cao của lề, ít cho các điểm vượt qua. Giá trị α_i mang ý nghĩa như mức độ vượt lề của điểm dữ liệu i.

Bài toán tối ưu lúc này là

$$\arg\min_{\vec{w},b,\vec{\alpha}} \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N}\alpha_i$$

$$y_i(\vec{w}^T\vec{x_i} + b) \geq 1 - \alpha_i, \forall i \in [1, N]$$

$$\alpha_i \geq 0, \forall i \in [1, N]$$

Bài toán đối ngẫu Lagrange trở thành

$$\max_{\vec{\lambda}} \sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \lambda_i \lambda_j \langle \vec{x_i}, \vec{x_j} \rangle$$

$$0 \leq \lambda_i \leq C, \forall i \in [1, N]$$

$$\sum_{i=1}^{N}\lambda_i y_i = 0$$

Bài toán này chỉ khác so với phiên bản SVM lề cứng ở việc thêm điều điện chặn $\lambda_i \leq C$. Như vậy, tóm tắt thuật toán nêu trong phần SVM lề cứng cũng sẽ chỉ thay đổi ở bước giải quy hoạch toàn phương.

2.4 SVM nhân

Trong thực tế, dữ liệu của chúng ta khả năng cao là sẽ không thể phân tách tuyến tính. Chúng ta có thể giải quyết vấn đề này bằng SVM lề mềm. Tuy nhiên, mô hình tìm được có thể không chính xác về bản chất, tức dữ liệu cần một hàm phi tuyến để phân tách. Để giải quyết vấn đề này, ngoài việc sử dụng vector đặc trưng $\vec{x}$, ta có thể thêm các đặc trưng khác là hàm phi tuyến của các phần tử trong $\vec{x}$.

Xét ví dụ bài toán XOR ($\oplus$). ta có

$$1 \oplus 1 = 0 \oplus 0 = 0 \Rightarrow N = \{(1, 1), (0, 0)\}$$

$$0 \oplus 1 = 1 \oplus 0 = 1 \Rightarrow P = \{(1, 0), (0, 1)\}$$

Có thể thấy trong mặt phẳng 2 chiều, không thể tìm ra đường thẳng phân tách hai tập này.

Tuy nhiên, nếu ta kết hợp thêm đặc trưng, ví dụ $x_3 = (x_1 - x_2)^2$, ta có bộ dữ liệu mới

$$N = \{(1, 1, 0), (0, 0, 0)\}, P = \{(1, 0, 1), (0, 1, 1)\}$$

Dữ liệu lúc này hoàn toàn có thể được phân tách bởi siêu phẳng $x_3 = 0.5$ hay $(x_1 - x_2)^2 - 0.5 = 0$.

Về mặt toán học, cách tiếp cận này được giải thích cụ thể như sau. Các điểm dữ liệu ban đầu thuộc về không gian dữ liệu, hay $\vec{x_i} \in \mathcal{X}$. Bài toán phân lớp nhị phân có thể xem như việc,

với một điểm đầu vào $\vec{x}$ mới, tìm y sao cho $(\vec{x}, y)$ tương đồng với các điểm nằm trong tập huấn luyện. Để làm được điều này, ta cần phải có một độ đo tương đồng trong $\mathcal{X}$ và trong $\{-1, 1\}$. Đối với tập $\{-1, 1\}$, do chỉ có 2 phần tử nên độ đo tương đồng chỉ cần đơn giản là giống hoặc khác. Đối với $\mathcal{X}$, ta cần xét một hàm

$$K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, (\vec{x}, \vec{x'}) \mapsto k(\vec{x}, \vec{x'})$$

sao cho $\forall x, x' \in \mathcal{X}$

$$K(\vec{x}, \vec{x'}) = \langle \Phi(\vec{x}), \Phi(\vec{y}) \rangle$$

với Φ là ánh xạ vào trong không gian tích vô hướng, gọi là không gian đặc trưng. K được gọi là là hàm nhân, còn Φ gọi là ánh xạ đặc trưng.

Ở ví dụ nêu trên, ánh xạ đặc trưng chính là

$$\vec{\Phi}(\vec{x}) = \begin{bmatrix} x_1 \\ x_2 \\ (x_1 - x_2)^2 \end{bmatrix}$$

Khi đó, đối với bài toán SVM, tất cả những $\vec{x}$ đều sẽ được thay bằng $\vec{\Phi}(\vec{x})$. Cách tiếp cận này có thể hiểu là ta sẽ ánh xạ dữ liệu sang một không gian khác mà trong đó tồn tại siêu phẳng để phân tách nó, tìm ra siêu phẳng đó và ánh xạ ngược trở về không gian dữ liệu. Thực tế, việc làm này rất phức tạp và tốn kém về mặt tính toán. Tuy nhiên, nhờ việc giải bài toán đối ngẫu (thay vì bài toán gốc) trong bài toán SVM lề cứng, ta tìm được một cách có thể dễ dàng chen giữa việc này vào trong quá trình giải bài toán, thậm chí đưa được dữ liệu vào một không gian vô hạn chiều (ví dụ sử dụng kernel Gaussian) mà không quá tốn kém.

Lợi thế này có được là do mặc dù $\vec{\Phi}(\vec{x})$ có thể vô cùng khó tính toán (tưởng tượng ánh xạ sang một không gian có số chiều rất lớn, thậm chí vô hạn), nhưng K thì lại rất dễ để tính. Nhìn lại thuật toán SVM khi giải bài toán đối ngẫu ở trong không gian đặc trưng, ta không thật sự cần phải tìm $\vec{\Phi}(\vec{x})$ bao giờ mà chỉ cần tìm $K(\vec{x}, \vec{z}) = \langle \vec{\Phi}(\vec{x}), \vec{\Phi}(\vec{x}) \rangle$. Như vậy, ta hoàn toàn có thể sử dụng nhân mà không cần phải xác định chính xác $\vec{\Phi}(\vec{x})$.

Xét ví dụ $K(\vec{x}, \vec{z}) = (\vec{x}^T \vec{z})^2, \vec{x}, \vec{z} \in \mathbb{R}^n$, biến đổi một chút ta có

$$K(\vec{x}, \vec{z}) = \left(\sum_{i=1}^{n} \vec{x_i} \vec{z_i} \right) \left(\sum_{i=1}^{n} \vec{x_i} \vec{z_i} \right) = \sum_{i=1}^{n} \sum_{j=1}^{n} (x_i x_j)(z_i z_j)$$

Trong trường hợp $n = 2$, hàm nhân này tương ứng với ánh xạ đặc trưng

$$\vec{\Phi}(\vec{x}) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_2 x_1 \\ x_2 x_2 \end{bmatrix}$$

Chú ý việc tìm $\vec{\Phi}$ có độ phức tạp $\mathcal{O}(n^2)$ do phải tìm tích của từng cặp phần tử. Trong khi đó độ phức tạp của việc tìm $K(\vec{x}, \vec{z})$ chỉ là $\mathcal{O}(n)$ bằng cách bình phương tích vô hướng $\vec{x}^T \vec{z}$, vốn tính trong $\mathcal{O}(n)$.

Như đã nói ở trên, hàm nhân có ý nghĩa phản ánh độ tương đồng giữa hai vector truyền vào, tức càng lớn khi nó càng tương đồng và càng bé khi ngược lại. Gọi G kích thước $N \times N$ là ma trận nhân ứng với không gian dữ liệu $\mathcal{X}$ khi các phần tử của nó bằng giá trị hàm nhân tương ứng, hay $G_{ij} = K(\vec{x_i}, \vec{x_j}), \vec{x_i}, \vec{x_j} \in \mathcal{X}$. Khi đó, một hàm nhân hợp lệ khi và chỉ khi G là ma trận đối xứng, nửa xác định dương.

Một số hàm nhân thông thường

- Tuyến tính: $\vec{x}^T \vec{z}$
- RBF (thường dùng trong các thư viện): $\exp(-\gamma \|\vec{x} - \vec{x}\|^2)$
- Đa thức: $(1 + \vec{x}^T \vec{z})^p$

Việc áp dụng hàm nhân vào thuật toán lúc này chỉ là chọn hàm K phù hợp và thay đổi tích vô hướng $\langle \vec{x}, \vec{z} \rangle$ thành $K(\vec{x}, \vec{z})$.

2.5 Phân lớp nhiều lớp

Như đã trình bày, SVM thường được sử dụng trong bài toán phân lớp nhị phân. Tuy nhiên, có thể mở rộng nó thành phân lớp n lớp ($n > 2$) với 2 hướng tiếp cận chính.

- One-vs-One (một chọi một): với mỗi cặp lớp ta sẽ tạo ra một bộ phân lớp nhị phân và có thể xác định lớp kết quả theo thể thức đấu vòng tròn. Hướng tiếp cận này tạo ra $\binom{n}{2}$ bộ phân lớp.
- One-vs-Rest (một chọi nhiều): với mỗi lớp, ta xét các phần tử thuộc lớp đó là tập Pos và các phần tử thuộc các lớp còn lại là tập Neg. Hướng tiếp cận này tạo ra n bô phân lớp.

Mỗi cách tiếp cận sẽ cho các đường phân tách riêng, tuy nhiên điểm chung là sẽ luôn tồn tại một vùng không rõ ràng (không biết thuộc về lớp nào). Vấn đề này nằm ngoài mục tiêu bài báo cáo nên xin phép chỉ nói sơ lược.

Tài liệu

[1] Machine Learning cơ bản - SVM

[2] Kernel Method in Machine Learning

[3] Stanford University, CS229 - Machine

[4] Bishop, Christopher M. "Pattern recognition and Machine Learning.", Springer (2006)

[5] Kernel and Feature map, Xavier Bourretsicotte