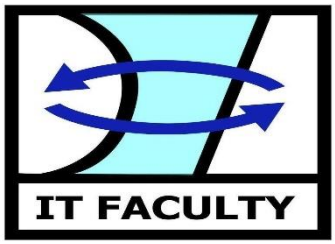
ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA CÔNG NGHỆ THÔNG TIN
Tel. (+84.0236) 3736949, Fax. (84-511) 3842771
Website: http://dut.udn.vn/khoacntt, E-mail: cntt@dut.udn.vn



BÁO CÁO MÔN HỌC
DỰ ÁN LẬP TRÌNH TÍNH TOÁN

ĐỀ TÀI :

SUPPORT VECTOR MACHINE

HỌ TÊN SINH VIÊN	MÃ SINH VIÊN	NHÓM
NGUYỄN DOÃN THANH HOÀNG	102230150	2
PHAN VĂN HIẾU	102230120	2

CBHD : **PGS.TS. NGUYỄN TẤN KHÔI**

Đà nẵng, dd/2023

MỤC LỤC

DANH MỤC HÌNH ẢNH

LỜI CẢM ƠN

Trong quá trình thực hiện bài báo cáo đồ án này, nhóm chúng em đã nhận được nhiều sự giúp đỡ từ giáo viên phụ trách **PGS.TS. NGUYỄN TẤN KHÔI**. Nhóm chúng em xin được gửi lời cảm ơn chân thành và sâu sắc đến thầy, cảm ơn thầy đã tạo điều kiện và hỗ trợ để chúng em có thể hoàn thành tốt bài báo cáo này. Một lần nữa nhóm em xin chân thành cảm ơn. Chúc thầy sức khỏe và thành đạt.

MỞ ĐẦU

Trong thời đại số hóa ngày nay, việc xử lý và phân hóa dữ liệu đang trở thành một phần không thể thiếu trong nhiều lĩnh vực, từ kinh doanh cho đến khoa học và công nghệ. Trong tương lai, việc hiểu và áp dụng các thuật toán phân hóa dữ liệu sẽ trở thành yếu tố quyết định giữa thành công và thất bại của một dự án.

Một trong những thuật toán quan trọng trong học máy là Máy Vector Hỗ Trợ (Support Vector Machine - SVM). Tuy nhiên, việc tối ưu hóa SVM để xử lý các tập dữ liệu lớn thường gặp phải nhiều thách thức do tính toán phức tạp. Đây là lý do tại sao thuật toán tối ưu hóa tuần tự (Sequential Minimal Optimization - SMO) được phát triển. SMO được giới thiệu bởi John Platt vào năm 1998, nhằm giải quyết bài toán tối ưu hóa cho SVM một cách hiệu quả hơn.

1. Mục đích và mục tiêu thực hiện đề tài:

1.1 Mục đích:

- Nhằm để giải quyết được về vấn đề phân lớp dữ liệu trong không gian giúp cho việc quản lí dữ liệu một cách chính xác.
- Dần hoàn thiện kĩ năng và tư duy lập trình tính toán

1.2 Mục tiêu:

Tìm ra được Hyperlane một cách tối ưu nhất để phân loại dữ liệu chính xác và hợp lí nhất có thể.

3. Tổng quan về đề tài

3.1 Khái niệm: Support vector machine (SVM) là một trong những thuật toán máy học giám sát, được sử dụng trong phân lớp tốt như là phân tích hồi quy. Tuy nhiên, SVM được sử dụng chủ yếu ở trong machine learning. Tạo ra chương trình tìm ra được phương trình siêu phẳng dùng để phân lớp dữ liệu trong không gian.

3.2 Sơ lượt về chương trình:

Giới thiệu và mô tả:

- Mã bắt đầu bằng một phần giới thiệu ngắn về đề tài, cung cấp thông tin nhóm thực hiện và hướng dẫn.
- Sử dụng các hằng số và thư viện để hiển thị màu sắc trong console.

Đọc dữ liệu:

- Sử dụng hàm createRecordVector để đọc dữ liệu từ tệp văn bản và lưu trữ vào một vector các bản ghi.

Thực hiện thuật toán SVM:

- Sử dụng thuật toán SMO (Sequential Minimal Optimization) để tối ưu hóa các hệ số alpha và bias, từ đó tìm ra hyperplane tối ưu để phân loại dữ liệu.
- Mỗi bản ghi trong tệp dữ liệu được biểu diễn bằng một vector các thuộc tính và thuật toán SVM được áp dụng để phân loại dữ liệu thành các lớp tương ứng.

Hiển thị kết quả:

- Sau khi hoàn thành thuật toán, kết quả được hiển thị thông qua hàm display bao gồm thông tin về các Support Vector, vector pháp tuyến và hệ số Bias.

Kết luận:

- Mã kết thúc bằng thông báo kết thúc chương trình và cảm ơn người dùng.

CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

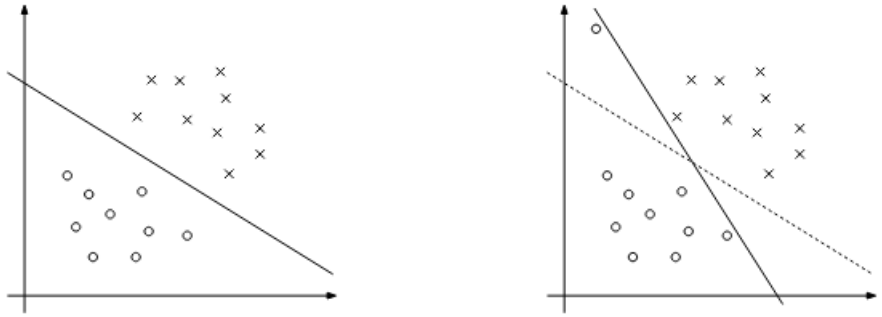
1.1. Ý tưởng

- Tạo cấu trúc dữ liệu dùng để chứa các tọa độ điểm của hai lớp dữ liệu.

- Sử dụng thuật toán SMO tối ưu hóa tuần tự hai trong số các biến đối ngẫu cho đến khi chúng hội tụ.

2.2. Phân tích toán học

Trong trường hợp dữ liệu đầu vào không tuyến tính hoàn hảo, trong một số trường hợp, việc tìm một siêu phẳng phân lớp có thể không phải là điều chúng ta mong muốn, vì nó có thể dễ bị ảnh hưởng bởi các điểm ngoại lệ. Ví dụ, hình bên trái dưới đây cho thấy một bộ phân loại biên tối ưu. Khi thêm một điểm ngoại lệ duy nhất vào vùng phía trên bên trái (hình bên phải), nó khiến biên quyết định thay đổi đáng kể, và bộ phân loại kết quả có một biên nhỏ hơn nhiều.



Hình 1: Tồn tại điểm ngoại lệ.

Để làm cho thuật toán hoạt động hiệu quả với các tập dữ liệu không thể phân tách tuyến tính cũng như ít nhạy cảm hơn với các điểm ngoại lệ, chúng ta sẽ cải tiến lại bài toán tối ưu hóa sau:

$$\boldsymbol{min_{w,b,\varepsilon}} \frac{\mathbf{1}}{\mathbf{2}} \|\boldsymbol{w}\|^2{}_2 + \boldsymbol{C} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{\varepsilon_i}$$

$$\textit{Subject to: } 1 - \boldsymbol{y_i}(\boldsymbol{w^T x_i} + \boldsymbol{b}) \geq \mathbf{1} - \boldsymbol{\varepsilon_i},\ \boldsymbol{i} = \mathbf{1}, \dots, \boldsymbol{n},$$

$$\textit{với } \boldsymbol{\varepsilon_i} \geq \mathbf{0},\ \boldsymbol{i} = \mathbf{1}, \dots, \boldsymbol{n}. \quad (1)$$

Nếu như các ví dụ có biên chức năng ($\boldsymbol{y_i}(\boldsymbol{wx_i} + \boldsymbol{b})$) nhỏ hơn 1, có nghĩa là các điểm dữ liệu không ảnh hưởng đáng kể đến siêu phẳng tối ưu. Các điểm này có thể được

loại bỏ mà không làm thay đổi vị trí của siêu phẳng với các điểm dữ liệu tuyến tính khác. Kế tiếp, nếu một ví dụ có biên chức năng chính xác là 1 − ξi (với ξ > 0), có nghĩa là nó gần với siêu phẳng tối ưu hơn một chút và đóng góp vào hàm mục tiêu với một chi phí Cξi. Tham số C được sử dụng để điều chỉnh trọng số giữa hai mục tiêu: Làm cho $||w||^2$ nhỏ hơn: Điều này có nghĩa là làm cho vectơ trọng số w nhỏ hơn, điều này có thể tăng cường biên giữa các lớp và giảm.

Tiếp theo, ta cần tiến hành Largange cho bài toán **(1)**. Khi đó:

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i \left[y^{(i)} (x^T w + b) - 1 + \xi_i \right] - \sum_{i=1}^{n} r_i \xi_i.$$

Dễ thấy, các $\boldsymbol{\alpha}_i$ và $\boldsymbol{r}_i$ là các hệ số nhân Lagrange (bị ràng buộc phải ≥ 0). Ta cần tìm dạng đối ngẫu của bài toán. Để làm điều này, trước tiên chúng ta cần tối thiểu hóa L(w, b, α) bằng cách cho đạo hàm riêng của (w, b, α) bằng 0.

$$\frac{\partial L}{\partial w} = 0 \Leftrightarrow w - \sum_{i=1}^{n} \alpha_i \boldsymbol{y^i x^i} = 0 \Leftrightarrow w = \sum_{i=1}^{n} \alpha_i \boldsymbol{y^i x^i} \quad (2)$$

$$\frac{\partial L}{\partial b} = 0 \Leftrightarrow \sum_{i=1}^{n} \alpha_i y_i = 0 \quad (3)$$

$$\frac{\partial L}{\partial \varepsilon} = 0 \Leftrightarrow \alpha_i = c - r_i \quad (4)$$

Thay các biểu thức trên vào Lagrange ta sẽ thu được hàm đối ngẫu sau:

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & 0 \le \alpha_i \le C, \quad i = 1, \ldots, n \\ & \sum_{i=1}^{n} \alpha_i y^{(i)} = 0, \end{aligned}$$

Ngoài ra các điều kiện bổ sung KKT(sẽ hữu ích trong phần kiểm tra sự hội tụ):

$$\begin{aligned} \alpha_i = 0 \quad &\Rightarrow \quad y^{(i)} (w^T x^{(i)} + b) \ge 1 \\ \alpha_i = C \quad &\Rightarrow \quad y^{(i)} (w^T x^{(i)} + b) \le 1 \\ 0 < \alpha_i < C \quad &\Rightarrow \quad y^{(i)} (w^T x^{(i)} + b) = 1. \end{aligned}$$

Tiếp theo, hãy đến với thuật toán SMO.

3.3. Lý thuyết về thuật toán SMO:

Thuật toán SMO do John Platt đề xuất, cung cấp một cách hiệu quả để giải quyết vấn đề đối ngẫu phát sinh từ việc dẫn xuất của SVM. Một phần để làm rõ nguồn gốc của thuật toán SMO và một phần vì bản thân của nó cũng rất thú vị, chúng ta hãy cùng tìm hiểu thêm về thuật toán ***tăng tọa độ***(nguồn gốc của thuật toán SMO).

Phương pháp tăng tọa độ (Coordinate Ascent/Descent) là một thuật toán tối ưu hóa lặp lại, trong đó ta tối ưu hóa từng biến trong số các biến của hàm mục tiêu một cách tuần tự, trong khi giữ các biến còn lại cố định.

Quy trình của phương pháp tăng tọa độ:

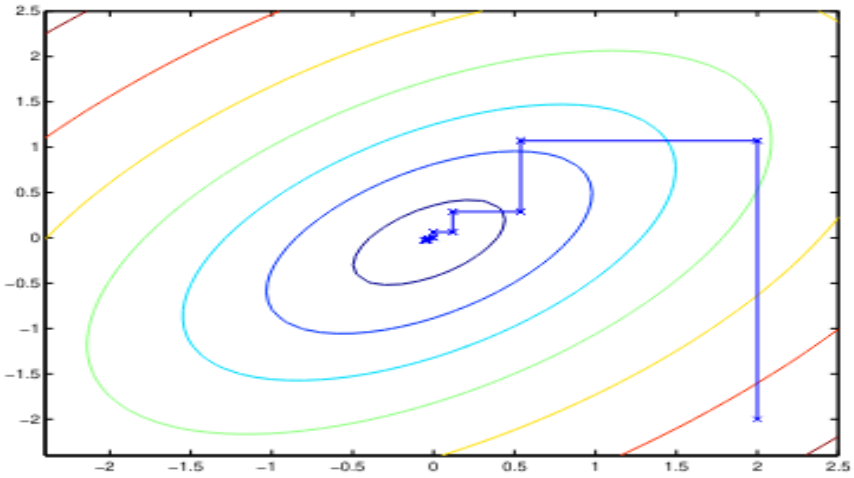
1. Khởi tạo: chọn giá trị ban đầu cho tất cả các biến $\alpha_1,\ \alpha_2, \alpha_3, \dots,\ \alpha_n$
2. Vòng lặp cho đến khi hội tụ

   Với từng biến α_i(i chạy từ 1 đến n):

   - Giữ cố định các biến $\alpha_1,\ \alpha_2, \dots, \alpha_{i-1},\ \alpha_{i+1}, \alpha_n$.
   - Tối ưu hóa hàm mục tiêu chỉ với biến α_i

     $$\alpha_i \coloneqq argmax\ w(\alpha_1,\ \alpha_2, \dots, \hat{a}_i\ \alpha_{i-1},\ \alpha_{i+1}, \alpha_n).$$

Dưới đây là hình ảnh của phương pháp tăng tọa độ đang hoạt động:



Hình 2: phương pháp tăng tọa độ

Các đường elip trong hình là các đường đồng mức của một hàm bậc hai mà chúng ta muốn tối ưu hóa. Tăng tọa độ được khởi tạo tại (2, -2), và cũng được vẽ trong hình là

đường đi mà nó đã thực hiện trên đường đến cực đại toàn cục. Lưu ý rằng ở mỗi bước, tăng tọa độ thực hiện một bước song song với một trong các trục, vì chỉ có một biến được tối ưu hóa tại một thời điểm.

Phương pháp tăng tọa độ được đề cập trong bối cảnh của SMO vì nó cung cấp cơ sở lý thuyết cho việc tối ưu hóa. SMO áp dụng nguyên lý này bằng cách tối ưu hóa từng cặp $\alpha_i v$à α_j trong khi các biến khác cố định. Sau khi hoàn tất cơ sở lí thuyết trên, hãy cùng tôi tìm hiểu về bản chất của thuật toán SMO(Sequential Minimal Optimization).

Thuật toán SMO là thuật toán tối ưu hóa tuần tự, một thuật toán đơn giản có thể nhanh chóng giải quyết vấn đề tối ưu hóa bậc hai (QP) của SVM. SMO phân chia vấn đề QP tổng thể thành các vấn đề QP con, sử dụng định lý của ***Osuna*** để đảm bảo hội tụ. Không giống như các phương pháp trước đây, SMO chọn giải quyết vấn đề tối ưu hóa nhỏ nhất có thể trong mỗi bước. Ưu điểm của SMO nằm ở chỗ giải quyết hai hệ số Lagrange có thể được thực hiện một cách phân tích. Do đó, việc tối ưu hóa QP số được loại bỏ hoàn toàn. Vòng lặp bên trong của thuật toán có thể được biểu diễn trong một lượng nhỏ mã, thay vì gọi toàn bộ thư viện QP. Nhiều vấn đề tối ưu hóa phụ được giải quyết trong quá trình của thuật toán, mỗi vấn đề phụ đều nhanh đến mức vấn đề QP tổng thể được giải quyết nhanh chóng. Ngoài ra, SMO không yêu cầu bất kỳ lưu trữ ma trận bổ sung nào. Vì vậy, các vấn đề đào tạo SVM rất lớn có thể phù hợp với bộ nhớ của một máy tính cá nhân hoặc máy trạm thông thường. Bởi vì không sử dụng các thuật toán ma trận trong SMO, nó ít bị ảnh hưởng bởi các vấn đề về độ chính xác số.

Có hai bước chính trong SMO:

- Phương pháp Heuristics chọn những hệ số nào để tối ưu hóa.
- Phân tích để giải quyết *hai* hệ số Lagrange.

Vì sao chọn 2 hệ số Lagrange để tối ưu mà không phải là 1...?. Một thắc mắc mà chắc có lẽ khá nhiều người đặt nghi vấn…, hãy cùng đến phần giải thích dưới đây để có cái nhìn rõ ràng hơn:

Theo điều kiện ràng buộc:

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \Leftrightarrow y_1 + \alpha_2 y_2 + \cdots + \alpha_n y_n = 0$$

$$\Leftrightarrow \alpha_1 y_1 = -[\alpha_2 y_2 + \cdots + \alpha_n y_n]$$

Nhân 2 vế với y_1:

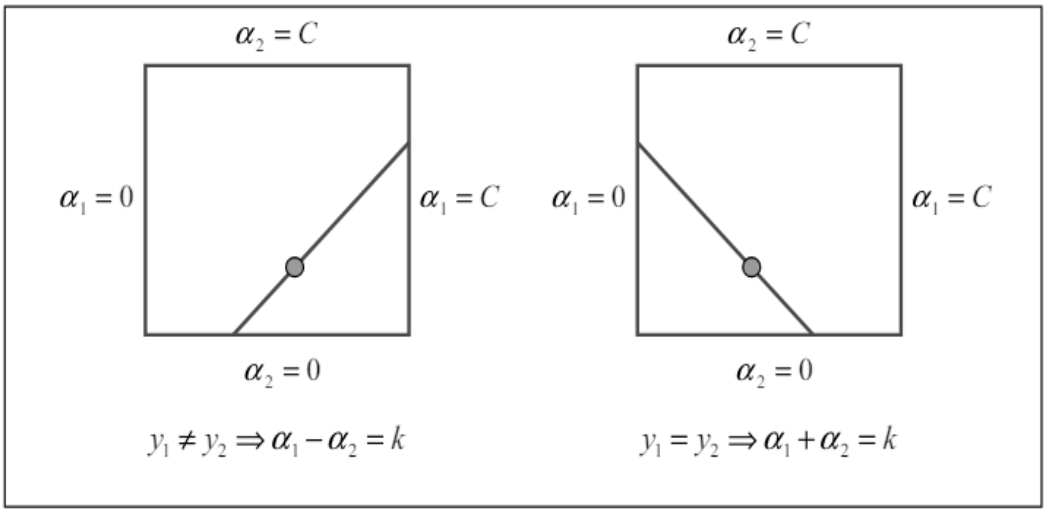
$$\alpha_1 = -y_1[\alpha_2 y_2 + \cdots + \alpha_n y_n] = \sum_{i=2}^{n} \alpha_i y_i$$

Nhận xét: Khi chỉ tối ưu hóa α_1 thì tất cả các biến đối ngẫu còn lại sẽ không đổi. Vậy α_1 cũng không đổi. Vì vậy cần phải tối ưu hóa hai hệ số Lagrange!!!.

Tương tự với chứng minh trên, đối với tối ưu hóa hai hệ số Larange thì công thức ràng buột là:

$$\alpha_1 y_1 + \alpha_2 y_2 = k, \text{ với k là hằng số} \qquad (\mathbf{5})$$

Trước hết, chúng ta sẽ tìm hiểu cách tối ưu hai hệ số Lagrange trước.



*Hình **3**: hình ảnh minh họa giới hạn giá trị của biến đối ngẫu*

Nhìn vào hình (2) và (5), ta thấy rằng:

Khi $y1 \neq y2 \Rightarrow \alpha_1 - \alpha_2 = k$

Khi $y1 = y2 \Rightarrow \alpha_1 + \alpha_2 = k$

Các đầu đoạn thẳng chéo có thể biểu diễn một cách đơn giản. Không làm mất tính tổng quát, thuật toán trước tiên tính toán hệ số Lagrange thứ hai α_2 và xác định các đầu của đoạn thẳng chéo theo α_2.

Nếu y_1 khác y2 thì các giới hạn sau đây sẽ được áp dụng cho α_2:

$$L = \max(\boldsymbol{\alpha_2 - \alpha_1}, 0), \quad H = \min(C, C + \boldsymbol{\alpha_2 - \alpha_1}) \qquad \mathbf{(6)}$$

Ngược lại, khi nhãn y_1 bằng y_2, thì các giới hạn sau đây được áp dụng cho α_2:

$$L = \max(\mathrm{C} - \boldsymbol{\alpha}_2 + \boldsymbol{\alpha}_1), \quad \mathrm{H} = \min(\mathrm{C}, \boldsymbol{\alpha}_2 + \boldsymbol{\alpha}_1) \qquad (7)$$

Ví dụ : khi $\alpha_1 = 0.5$, $\alpha_2 = 1$, $y_1 = y_2$, C = 1 => cận dưới (L) và cận trên (H) của α_2 lần lượt là: 0.5 và 1.

Tiếp theo, đạo hàm của hàm mục tiêu dọc theo đường chéo được biểu diễn như sau:

$$\eta = k(\overrightarrow{x_1}, \overrightarrow{x_1}) + k(\overrightarrow{x_2}, \overrightarrow{x_2}) - 2 * k(\overrightarrow{x_1}, \overrightarrow{x_2}) \qquad \mathbf{(8)}$$

K là hàm kernel là sản phẩm tích vô hướng của các điểm dữ liệu

Khi hàm mục tiêu dương thì $\eta > 0$ khi đó thì hàm mục tiêu sẽ có tính lồi, lúc đó sẽ có những tính chất sau đây:

- Điểm cực tiểu toàn cục duy nhất. Điều này có nghĩa là không có vùng nào trong hàm mục tiêu mà giá trị giảm xuống và sau đó tăng lên.
- Quá trình tối ưu hóa sẽ hội tụ về điểm cực tiểu toàn cục, điều này đảm bảo rằng việc tối ưu hóa sẽ đưa ra kết quả ổn định.

Kết luận: Trong những trường hợp bình thường, hàm mục tiêu sẽ có tính dương, sẽ có một điểm cực tiểu theo hướng của ràng buộc tuyến tính, và khi đó $\eta > 0$ nên kết quả sẽ ổn định.

Trong trường hợp này, SMO tính toán điểm cực tiểu theo hướng của ràng buột:

$$\alpha_2^{\text{new}} = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta} \qquad \mathbf{(9)}$$

Trong đó $E_i = u - y_i$ là lỗi trên ví dụ đào tạo thứ i, với u là vị trí điểm dữ liệu thứ i thuộc hyperplane nào đó.

Nếu sai số E này quá lớn, điều đó cho thấy mô hình cần được diều chỉnh để cải thiện độ chính xác của dự đoán.

Điểm cực tiểu có ràng buột được tìm bằng cách cắt ngắn cực tiểu không ràng buột về các điểm cuối của đoạn thẳng trong ***hình (2)***, cập nhật giá trị biến đối ngẫu thứ thứ hai như sau:

$$\alpha_2^{\text{new,clipped}} = \begin{cases} H & \text{if} \quad \alpha_2^{\text{new}} \geq H; \\ \alpha_2^{\text{new}} & \text{if} \quad L < \alpha_2^{\text{new}} < H; \\ \text{L} & \text{if} \quad \alpha_2^{\text{new}} \leq L. \end{cases}$$

Sau khi tính gá trị α_2, ta cập nhật lại giá trị α_1:

$$\alpha^{new} = \alpha_1 + s(\alpha_2 - \alpha_2^{new}), \quad \text{s} = y_1 y_2.$$

Nhận xét:

Trong trường hợp bình thường khi η dương thì thuật toán chạy rất tốt và ổn định, còn khi η không dương thì hãy cùng đến với phần nhân xét tiếp theo.

1. Trường hợp giá trị η âm: điều này có thể làm cho hàm mục tiêu trở nên lõm. Trong ngữ cảnh này, hàm mục tiêu không còn được xác định một cách rõ ràng và có thể gây ra các vấn đề không mong muốn trong quá trình tối ưu hóa.

2. Giá trị η bằng 0**:** Điều này có thể xảy ra khi có nhiều hơn một ví dụ huấn luyện có cùng một vector đầu vào. Trong SVM, đây là trường hợp mà các điểm dữ liệu trong không gian đặc trưng không thể được phân biệt hoàn toàn, do đó giá trị η bằng 0 sẽ phù hợp với các điều kiện ràng buộc của bài toán tối ưu hóa.

3. SMO và giá trị η không dương: Thuật toán SMO vẫn có thể hoạt động hiệu quả trong các trường hợp khi giá trị η không dương. Trong trường hợp này, hàm mục tiêu sẽ được đánh giá ở hai đầu của một đoạn thẳng, điều này thường xảy ra khi các điểm dữ liệu gần nhau trong không gian đặc trưng.

Tiếp theo cùng đến với **Nguyên lí Heuristics và Second-Heuristics**

Nguyên lí trên sẽ chọn hai hệ số Lagrange để tối ưu hóa, chúng hoạt động như sau:

1. Kiểm tra điều kiện KKT:

- Đoạn mã bắt đầu bằng việc tính giá trị lỗi E2 và giá trị r2 tại biến đối ngẫu thứ hai.
- Nếu r2 không thỏa mãn điều kiện KKT (tức là ví dụ hiện tại cần tối ưu hóa), đoạn mã tiếp tục.

2. Sử dụng Second-Heuristics để chọn biên đối ngẫu đầu tiên:

- Nếu có nhiều hơn một ví dụ trong tập huấn luyện, đoạn mã sử dụng Second-Heuristics để chọn biến đối ngẫu thứ nhất.
- Bắt đầu quá trình tối ưu hóa cho 2 biến đối ngẫu, nếu chúng được tối ưu hóa thì kết thúc thuật toán, còn nếu Sec-Heuristics không tìm được cặp phù hợp, đoạn mã tiếp tục duyệt qua tất cả các biến đối ngẫu khác giá trị 0 và C và bắt đầu tối ưu hóa .

Nếu vẫn không thành công (tức là vẫn chưa có quá trình tối ưu nào cho biến đối ngẫu), đoạn mã sẽ duyệt qua tất cả các ví dụ trong tập huấn luyện, cho đến khi có sự tối ưu thì dừng thuật toán.

Mục tiêu của cả hai heuristics

Heuristic thông thường: Tìm các điểm dữ liệu vi phạm điều kiện KKT và cố gắng tối ưu hóa chúng bằng cách duyệt qua các ví dụ không ràng buộc và toàn bộ tập huấn luyện.

Heuristic thứ hai (sec_heuristics): Trong các trường hợp khó khăn, sec_heuristics cố gắng tìm các cặp điểm dữ liệu có thể tạo ra sự thay đổi đáng kể trong quá trình tối ưu hóa. Điều này đặc biệt hữu ích khi các heuristics thông thường không tạo ra tiến trình hữu ích.

Sau khi Tối ưu hóa được mỗi cặp hệ số đối ngẫu Lagrange, tiếp tục tiến hành tối ưu hóa hệ số **bias**. Ngưỡng **bias** được tính lại sau mỗi bước, để đảm bảo các điều kiện KKT được thỏa mãn cho cả hai ví dụ đã được tối ưu hóa.

Ngưỡng b_1 sau đây có giá trị khi hệ số Lagrange mới là α_1 không ở ranh giới, vì nó buộc đầu ra của SVM là y_1 khi đầu vào là x_1:

$$b_1 = - E_1 - y_1 \cdot K(x_1, x_2) \,.\, (\alpha_2^{new} - \alpha_2) - y_1 \cdot K(x_1, x_2) \,.\, (\alpha_1^{new} - \alpha_1) + \text{bias}$$

Ngưỡng b_2 sau đây có giá trị khi hệ số Lagrange mới là α_2 không ở ranh giới, vì nó buộc đầu ra của SVM là y_2 khi vector đầu vào là x_2:

$$b_2 = - E_2 - y_2 \cdot K(x_1, x_1) \,.\, (\alpha_2^{new} - \alpha_2) - y_1 \cdot K(x_1, x_2) \,.\, (\alpha_1^{new} - \alpha_1) + \text{bias}$$

Tiếp theo, nếu α_1^{new} hoặc α_2^{new} hoặc cả hai $\in \{0, C\}$ và $L \neq H$ thì bias sẽ được cập nhật như sau:

Bias =

- Quá trình cứ lặp đi lặp lại cho đến khi các hệ số đối ngẫu được tối ưu tốt nhất.

Khi đó ***hyperplane*** sẽ được tính theo công thức đã được nhắc ở **(2)** là:

$$w = \sum_{i=1}^{n} \alpha_i y^i x^i$$

CHƯƠNG 2: TỔ CHỨC CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

2.1. Phát biểu bài toán

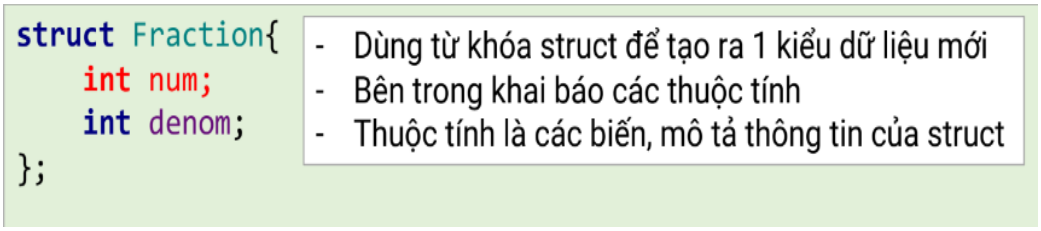
Mô tả đầu vào(*input*) và đầu ra(*output*)

- Đầu vào: Danh sách tọa độ vector của các điểm dữ liệu.
- Đầu ra: Tọa độ của vector pháp tuyến w, hệ số bias và các biến đối ngẫu của Support Vector hoặc là điểm nhiễu (nếu có).

2.2. Cấu trúc dữ liệu

2.2.1) Struct (viết tắt của từ "Structure")

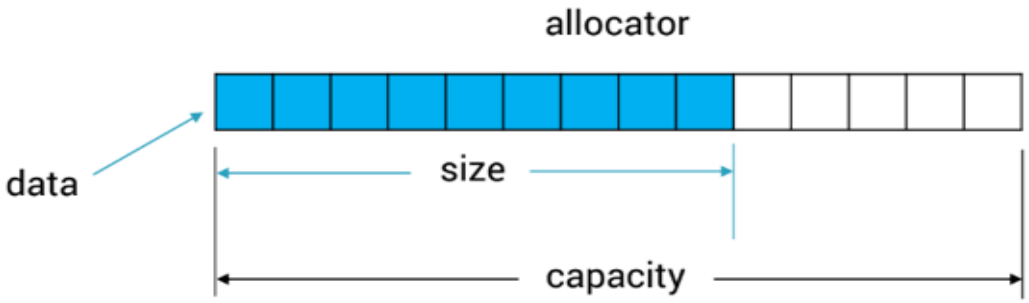
Là một cấu trúc dữ liệu cho phép bạn nhóm các biến lại với nhau dưới một tên chung. Cấu trúc này giúp tổ chức dữ liệu liên quan lại với nhau, thường là các biến thuộc những kiểu dữ liệu khác nhau. Struct trong C++ có thể được sử dụng giống như một kiểu dữ liệu tùy chỉnh, cho phép bạn định nghĩa các thuộc tính (biến thành viên) và các hàm (phương thức thành viên).



Hình 3: hình ảnh minh họa về struct

2.2.2) Dynamic Array(vector):

Là cấu trúc dữ liệu dung để lưu trữ các đối tượng có kích thước không xác định. Khác với mảng tĩnh, người dung không cần phải khai báo trước số lượng phần tử khi sử dụng.



Hình 4: hình ảnh minh họa về vector

Một vài hàm trong thư viện chuẩn STL được sử dụng:

- push_back() thêm 1 phần tử vào vector
- erase() : xóa phần tử đó ra khỏi vector
- size() : đếm số phần tử hiện có trong vector
- Clear(): xóa tất cả phần tử bên trong Vector(dung lượng không bị giải phóng mà vẫn còn trong bộ nhớ)

…

Khai báo kiểu Vector có kích thước là mảng động 1 chiều và 2 chiều lần lượt là:

- vector<Type> VectorName (1D)
- vector<vector<Type>> MatrixName (2D)

2.3. Thuật toán

2.3.1) Thuật toán **Kernel** tính tích vô hướng của 2 điểm dữ liệu

```
double kernel(vector<double>& x1, vector<double>& x2)
    double sum = 0.0;
    for(int i = 0; i < x1.size(); ++i)
        sum += x1[i] * x2[i];
    return sum;
End kernel
```

2.3.2) Thuật toán **SvmOutput** có chức năng tính toán đầu ra của mô hình SVM cho một điểm dữ liệu mới dựa trên các nhân tử Lagrange α, nhãn tickets, và các điểm hỗ trợ (support vectors) đã được xác định trong quá trình huấn luyện.

```
double svmOutput(vector<double> x)
    double result = 0.0;
    for(int i = 0; i < alpha.size(); ++i)
    {
        result += alpha[i] * tickets[i] * kernel(points[i], x);
     }
     return resule + bias;
```

End svmOutput

2.3.3) Thuật toán **TakeStep** trong thuật toán có chức năng chính là thực hiện một bước cập nhật các nhân tử Lagrange (α) cho hai điểm dữ liệu trong quá trình tối ưu hóa hàm mục tiêu của SVM. Đây là một bước quan trọng trong việc tìm kiếm lời giải tối ưu cho bài toán SVM.

```
int takeStep(int i1, int i2)
    Nếu chỉ có 1 điểm dữ liệu thì dừng việc tối ưu
  if(i1 == i2) return 0;

  double alph1 = alpha[i1];
  double alph2 = alpha[i2];
  int y1 = tickets[i1];
  int y2 = tickets[i2];
  double s = y1 * y2;

 double E1 = svmOutput(points[i1]) - y1;
 double E2 = svmOutput(points[i2]) - y2;

 double L, H;
 if(y1 != y2)
 {
      L = max(0.0, alph2 - alph1);
```

```
        H = min(C, C + alph2 - alph1);
  }
  else
  {
        L = max(0.0, alph2 + alph1 - C);
        H = min(C, alph2 + alph1);
  }

  if(L == H)
  {
        return 0;
  }
  double k11 = kernel(points[i1], points[i1]);
  double k12 = kernel(points[i1], points[i2]);
double k22 = kernel(points[i2], points[i2]);

double eta = k11 + k22 - 2 * k12;

double a2;
if(eta > 0)
{
        a2 = alph2 + y2 * (E1 - E2) / eta;
        if(a2 < L) a2 = L;
        else if(a2 > H) a2 = H;
}
else
{
        Tính toán giá trị khách quan tại L và H
        double Lobj = svmOutput(points[i2]) * y2 * L;
        double Hobj = svmOutput(points[i2]) * y2 * H;
```

```
            if(Lobj < Hobj - eps)       a2 = L;
            else if(Lobj > Hobj + eps)  a2 = H;
            else  a2 = alph2;
        }
        if(fabs(a2 - alph2) < eps * (a2 + alph2 + eps)) return 0;
        double a1 = alph1 + s * (alph2 - a2);
        alpha[i1] = a1;
        alpha[i2] = a2;

        Cập nhật bias (nếu cần)
        double b1 = -E1 - y1 * k11 * (a1 - alph1) - y2 * k12 * (a2 - alph2) +  bias;
        double b2 = -E2 - y1 * k12 * (a1 - alph1) - y2 * k22 * (a2 - alph2) + bias;

        if(0 < a1 && a1 < C)        bias = b1;
        else if(0 < a2 && a2 < C)   bias = b2;
        else                        bias = (b1 + b2) / 2;

    return 1;
```

End takeStep

2.3.4) Thuật toán **examineExample** trong thuật toán SMO có chức năng chính là kiểm tra xem nhân tử Lagrange (α) của điểm dữ liệu thứ hai (i2) có vi phạm các điều kiện KKT (Karush-Kuhn-Tucker) hay không. Nếu vi phạm, hàm sẽ cố gắng tìm và cập nhật một cặp nhân tử Lagrange (α_1 và α_2) để cải thiện hàm mục tiêu của SVM.

```
int examineExample(int i2)
        int y2 = tickets[i2];
        double alph2 = alpha[i2];
        double E2 = svmOutput(points[i2]) - y2;
        double r2 = E2 * y2;
```

```
 if((r2 < -tol && alph2 < C) || (r2 > tol && alph2 > 0))
{
      int numAlpha = alpha.size();
      int i1 = -1;
      if(numAlpha > 1)
     {
            random_device rd;
            mt19937 gen(rd());
            uniform_int_distribution<> dis(0, numAlpha - 1);
            i1 = dis(gen); // Chọn một số ngẫu nhiên

            if(takeStep(i1, i2) == 1)
           {
                 return 1;
            }
     }

    Lặp qua tất cả các alpha non-zero và non-C
    for(int i = 0; i < numAlpha; ++i)
    {
          if(alpha[i] != 0 && alpha[i] != C)
          if(takeStep(i, i2) == 1)
          return 1;
    }
    Lặp qua tất cả các ví dụ
    for(int i = 0; i < numAlpha; ++i)
          if(takeStep(i, i2) == 1)
               return 1;
          return 0;
```

End examineExample

2.3.5) Thuật toán **smoAlgorithm** là phần chính của SMO (Sequential Minimal Optimization) và có chức năng tối ưu hóa các nhân tử Lagrange α để giải bài toán SVM. Đây là quy trình lặp lại liên tục cho đến khi không có sự thay đổi nào hoặc tất cả các điểm dữ liệu đã được xem xét và không còn cần thiết phải xem xét thêm.

```
void smoAlgorithm()
    int numChanged = 0;
    bool examineAll = true;
    int numPoints = points.size();

    while(numChanged > 0 || examineAll)
    {
        numChanged = 0;

        if(examineAll)
        {
        for(int i = 0; i < numPoints; ++i)
            numChanged += examineExample(i);
        examineAll = false;
         }
         else
         {
         for(int i = 0; i < numPoints; ++i)
            if(alpha[i] > 0 && alpha[i] < C)
                numChanged += examineExample(i);
        if(numChanged == 0)   examineAll = true;
         }
    }
End smoAlgorithm
```

END!!!.

CHƯƠNG 3: CHƯƠNG TRÌNH VÀ KẾT QUẢ

3.1. Tổ chức chương trình

STT	Tên Hàm	Chức Năng chính
1	createRecordVector	Đọc file lưu dữ liệu vào vector
2	SvmOutput	Tính toán đầu ra của dữ liệu trong mô hình SVM
3	Kernel	Sản phẩm là tích vô hướng của 2 điểm dữ liệu
4	takestep	Tối ưu và cập nhật nhân tử lagrange
5	ExamineExample	Kiểm tra điều kiện KKT
6	SmoAlgorithm	Xem xét độ tối ưu của hệ số lagrange để tối ưu hóa nếu cần thiết
7	Introduction	Gioi Thiệu thông tin về giáo viên, thành viên và project
8	Display	In ra sản phẩm của dự án
9	COLOR_NEXT	In màu

3.2. Ngôn ngữ cài đặt

Ngôn ngữ c++.

3.3. Kết quả

3.3.1) Giao diện chính của chương trình

```
                    DE TAI: SUPPORT VECTOR MACHINE
                Sequential Minimal Optimization Algorithm
                            MACHINE LEARNING

                    GIAO VIEN HUONG DAN: NGUYEN TAN KHOI
          SINH VIEN THUC HIEN:
          NGUYEN DOAN THANH HOANG     LOP 23T_KHDL2     NHOM 1
          PHAN VAN HIEU               LOP 23T_KHDL1     NHOM 1

                                K23 DUT

NHAP SO CHIEU: 3

NHAP TEN FILE CAN DOC: iris1.tab
A.  TIEN HANH DOC FILE TU TEP DU LIEU
    DA DOC FILE THANH CONG!!!

B.  TIEN HANH CHAY THUAT TOAN SMO

    BAT DAU QUA TRINH TOI UU HOA HAI HE SO LAGRANGE VA CAP NHAT HE SO BIAS:

    KET THUC QUA TRINH TOI UU!!!
    KET THUC THUAT TOAN SMO!!!
```

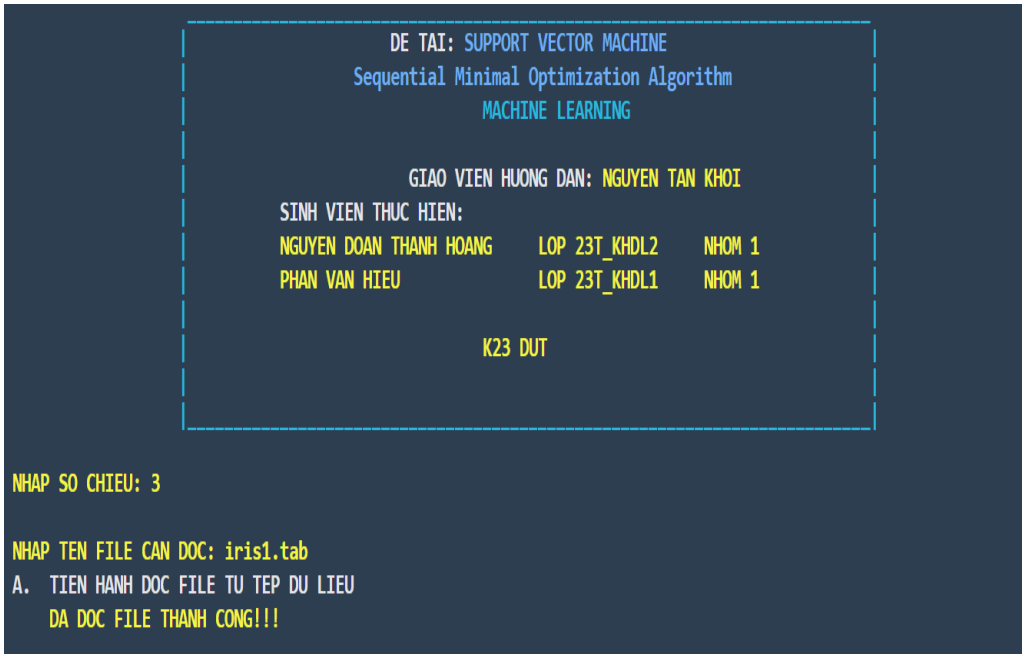
```
C.  TIEN HANH XUAT KET QUA CUA THUAT TOAN SMO

    THONG TIN VE SUPPORT VECTOR
| STT | DUAL COEFFICIENT | SUPPORT VECTOR |
|  3  |      0.062       | 2.00  3.00  0.00 |
|  4  |      0.062       | 6.00  -1.00 0.00 |
    THONG TIN VE HYPERPLANE
| VECTOR TRONG SO w | HE SO BIAS |
|  0.25  -0.25 0.00 |    -0.75   |

D.   TIEN HANH UNG DUNG KIEM TRA TINH PHAN LOP CUA CAC DIEM DU LIEU
E.   TIEN HANH KIEM TRA THUAT TOAN SMO
F.  KET THUC CHUONG TRINH!!!

                ================================== THANK YOU ==================================
```

3.3.b) Kết quả thực thi của chương trình

TEST 1:

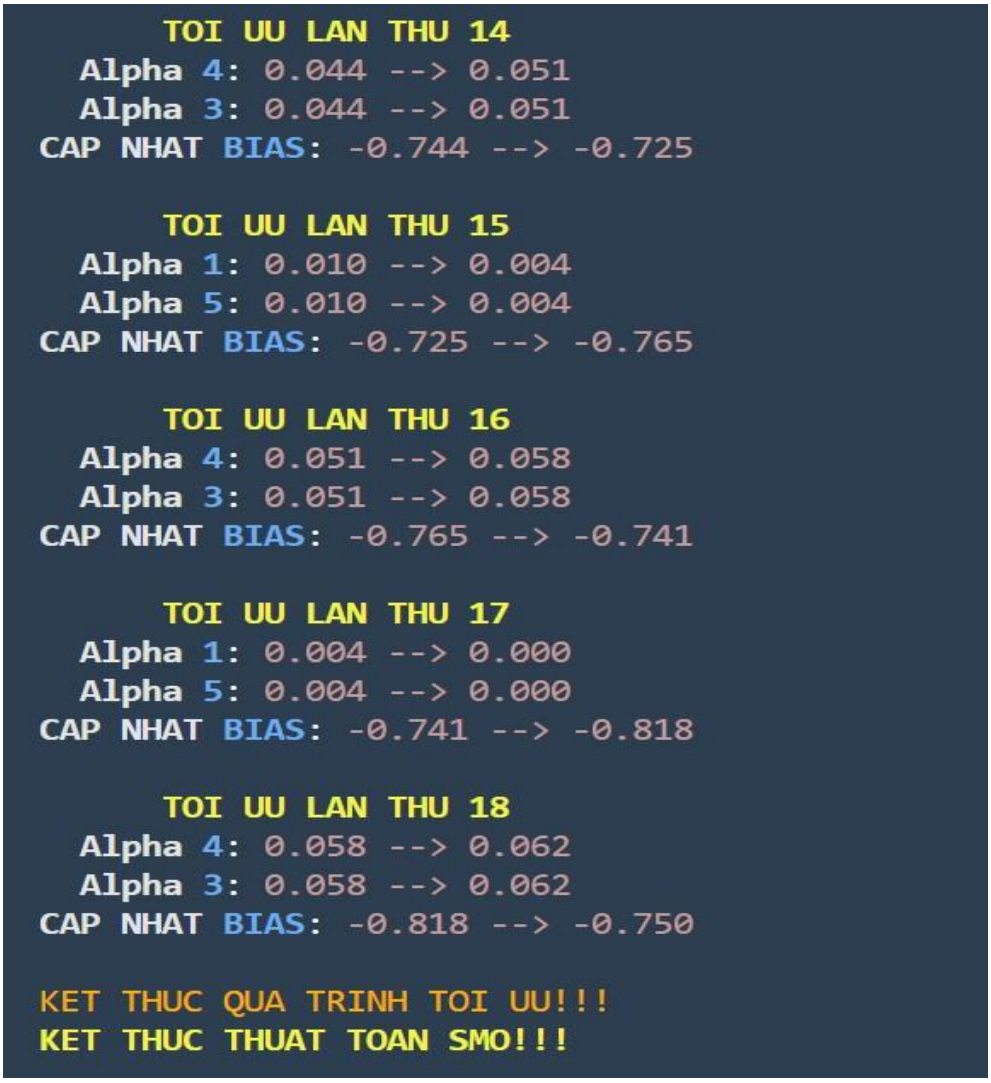
Hình 3.3.2.1: Test 1.

Giao diện sau khi nhập dữ liệu:



Hình 3.3.2.2: giao diện sau khi nhập dữ liệu

Tiếp theo, bắt đầu tiến hành thực hiện thuật toán smo, tối ưu hóa 2 biến đối ngẫu và cập nhật bias.



Hình 3.3.2.3: Hình ảnh minh họa về quá trình tối ưu hóa.

Sau khi biến đối ngẫu và Bias được tối ưu, tiếp tục tiến hành tính w và in siêu phẳng cùng vector hỗ trợ.

C. TIEN HANH XUAT KET QUA CUA THUAT TOAN SMO

THONG TIN VE SUPPORT VECTOR

STT	DUAL COEFFICIENT	SUPPORT VECTOR
3	0.062	2.00 3.00 0.00
4	0.062	6.00 -1.00 0.00

THONG TIN VE HYPERPLANE

VECTOR TRONG SO w	HE SO BIAS
0.25 -0.25 0.00	-0.75

Hình 3.3.2.4: Bảng thông tin kết quả.

Sau khi có siêu phẳng, tiến hành kiểm tra lại kết quả bằng cách tính đầu ra của các điểm dữ liệu thông qua siêu phẳng, từ đó đối chiếu dữ liệu ban đầu để kiểm tra thuật toán.

```
E.   TIEN HANH KIEM TRA THUAT TOAN SMO

     TOA DO DIEM DU LIEU THU 1:      DAU RA CUA DIEM DU LIEU LA: -1.00 KHONG PHAI LA SUPPORT VECTOR!!, THUOC LOP -1
     TOA DO DIEM DU LIEU THU 2:      DAU RA CUA DIEM DU LIEU LA: -1.50 Thuoc lop -1

     TOA DO DIEM DU LIEU THU 3:      DAU RA CUA DIEM DU LIEU LA: -1.00 LA MOT VECTOR HO TRO!

     TOA DO DIEM DU LIEU THU 4:      DAU RA CUA DIEM DU LIEU LA: 1.00 LA MOT VECTOR HO TRO!

     TOA DO DIEM DU LIEU THU 5:      DAU RA CUA DIEM DU LIEU LA: 1.25 THUOC LOP +1

     TOA DO DIEM DU LIEU THU 6:      DAU RA CUA DIEM DU LIEU LA: 1.25 THUOC LOP +1

F.  KET THUC CHUONG TRINH!!!
```

Hình 3.3.2.5: Kiểm tra đối chiếu kết quả của thuật toán.

3.3.c) Mô tả kết quả thực hiện chương trình.

Nhận xét đánh giá:

- Chương trình tìm Hyperplane một cách nhanh chóng và chính xác.
- Chương trình chạy ổn định.
- Làm đơn giản vấn đề.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

1. Kết luận

Hiệu quả: SMO cải thiện tốc độ huấn luyện SVM bằng cách giải quyết các bài toán con một cách nhanh chóng và trực tiếp, thay vì phải sử dụng các phương pháp tối ưu hóa toàn cục phức tạp hơn.

Đơn giản: SMO không yêu cầu các bước tính toán ma trận lớn, làm giảm bớt yêu cầu về bộ nhớ và tính toán.

Khả năng xử lý phi tuyến: SMO có thể được sử dụng với các hàm kernel khác nhau, giúp SVM có khả năng xử lý các vấn đề phân loại phi tuyến.

Tiến bộ thực nghiệm: SMO được chứng minh là một trong những thuật toán thực nghiệm hiệu quả nhất cho việc huấn luyện SVM, đặc biệt là với các tập dữ liệu lớn.

2. Hướng phát triển

Mở rộng cho SVM đa lớp: Mở rộng SMO để xử lý các bài toán phân loại đa lớp.

TÀI LIỆU THAM KHẢO

Tiếng Anh:

1. Andrew Ng(1976), CSS29 FALL 2023-2024 Stanford University
2. John C. Platt(1963), Microsoft.com
3. Bishop, Christopher M. “Pattern recognition and Machine Learning.”, Springer (2006). (book)

Tiếng Việt

4. AI Viet Nam
5. Machine Learning cơ bản

PHỤ LỤC

```
#include<iostream>
#include<vector>
#include<random>
#include<cmath>
#include <fstream>
#include <string>
#include<iomanip>
#define CLEAR_TEXT "\33[0m"
#define NORMAL_TEXT "\33[1;37m"
#define RED_TEXT "\33[1;31m"
#define GREEN_TEXT "\33[1;32m"
#define YELLOW_TEXT "\33[1;33m"
#define BLUE_TEXT "\33[1;34m"
#define PINK_TEXT "\33[1;35m"
#define CYAN_TEXT "\33[1;36m"
#define WHITE_TEXT "\33[1;37m"
#define GREY_TEXT "\033[38;5;243m"
using namespace std;
const double tol = 1e-3;
```

```
const double eps = 1e-5;
const double C = 1.0; // Thông số điều chỉnh
struct Record
{
    float sepal_width, sepal_length, petal_width, petal_length;
    std::string classname;

    void reset_Record()
    {
        sepal_width = 0;
        sepal_length = 0;
        petal_width = 0;
        petal_length = 0;
        classname.clear();
    }

    void cout_Record()
    {
        std::cout << sepal_length << " "
        << sepal_width << " " << petal_length << " " << petal_width << " "
        << classname << "\n";
    }
};

void createRecordVector(std::vector<Record> &records, const char *filename)
{
    // open file
    std::ifstream f(filename);
    // error checking
    if (!f)
    {
```

```
            std::cout << "Error\n" << endl;
            return;
        }
        cout << "    DA DOC FILE THANH CONG!!!\n";
       // cout << "\nDU LIEU DUOC NHAP TU FILE LA: \n";
        std::string buffer;
        getline(f, buffer);
        getline(f, buffer);
        getline(f, buffer);

        Record buffer_record;
        records.clear();
        buffer_record.reset_Record();
        char buffer_char;
        //int STT = 0;
        while (true)
        {
            f >> buffer_record.sepal_length >> buffer_record.sepal_width
            >>  buffer_record.petal_length  >>  buffer_record.petal_width  >>
buffer_record.classname;
            if (!buffer_record.sepal_length)
                break;

            records.push_back(buffer_record);
            //cout << ++STT << ": ";
            //records[records.size() - 1].cout_Record();
            buffer_record.reset_Record();
        }

        // close file
        f.close();
```

```
}

struct SVM
{
    vector<double> alpha;
    vector<int> tickets;
    vector<vector<double>> points;
    double bias = 0.0;

    // Hàm kernel
    double kernel(vector<double>& x1, vector<double>& x2)
    {
        double sum = 0.0;
        for(int i = 0; i < x1.size(); ++i)
            sum += x1[i] * x2[i];
        return sum;
    }

    // Hàm SVM output
    double svmOutput(vector<double> x)
    {
        double result = 0.0;
        for(int i = 0; i < alpha.size(); ++i){
            result += alpha[i] * tickets[i] * kernel(points[i], x);
        }
        return result + bias;
    }

    // Hàm kiểm tra và thay đổi
    int takeStep(int i1, int i2)
    {
```

```
if(i1 == i2) return 0;

double alph1 = alpha[i1];
double alph2 = alpha[i2];
int y1 = tickets[i1];
int y2 = tickets[i2];
double s = y1 * y2;

double E1 = svmOutput(points[i1]) - y1;
double E2 = svmOutput(points[i2]) - y2;

double L, H;
if(y1 != y2)
{
   L = max(0.0, alph2 - alph1);
   H = min(C, C + alph2 - alph1);
}
else
{
   L = max(0.0, alph2 + alph1 - C);
   H = min(C, alph2 + alph1);
}

if(L == H)
{
   return 0;
}
double k11 = kernel(points[i1], points[i1]);
double k12 = kernel(points[i1], points[i2]);
double k22 = kernel(points[i2], points[i2]);
```

```
double eta = k11 + k22 - 2 * k12;

double a2;
if(eta > 0)
{
   a2 = alph2 + y2 * (E1 - E2) / eta;
   if(a2 < L) a2 = L;
   else if(a2 > H) a2 = H;
}
else
{
   // Tính toán giá trị khách quan tại L và H
   double Lobj = svmOutput(points[i2]) * y2 * L;
   double Hobj = svmOutput(points[i2]) * y2 * H;

   if(Lobj < Hobj - eps)       a2 = L;
   else if(Lobj > Hobj + eps)  a2 = H;
   else  a2 = alph2;
}
if(fabs(a2 - alph2) < eps * (a2 + alph2 + eps)) return 0;
double a1 = alph1 + s * (alph2 - a2);
alpha[i1] = a1;
alpha[i2] = a2;

// Cập nhật bias (nếu cần)
double b1 = -E1 - y1 * k11 * (a1 - alph1) - y2 * k12 * (a2 - alph2) + bias;
double b2 = -E2 - y1 * k12 * (a1 - alph1) - y2 * k22 * (a2 - alph2) + bias;

if(0 < a1 && a1 < C)        bias = b1;
else if(0 < a2 && a2 < C)   bias = b2;
else                        bias = (b1 + b2) / 2;
```

```
return 1;
}

// Hàm kiểm tra một ví dụ
int examineExample(int i2)
{
    int y2 = tickets[i2];
    double alph2 = alpha[i2];
    double E2 = svmOutput(points[i2]) - y2;
    double r2 = E2 * y2;

    if((r2 < -tol && alph2 < C) || (r2 > tol && alph2 > 0))
    {
        int numAlpha = alpha.size();

        // Lựa chọn i1 bằng heuristic (tùy chỉnh nếu cần)
        int i1 = -1;
        if(numAlpha > 1)
        {
            random_device rd;
            mt19937 gen(rd());
            uniform_int_distribution<> dis(0, numAlpha - 1);
            i1 = dis(gen); // Chọn một số ngẫu nhiên

            if(takeStep(i1, i2) == 1)
            {
                return 1;
            }
        }
```

```
        // Lặp qua tất cả các alpha non-zero và non-C
        for(int i = 0; i < numAlpha; ++i)
        {
            if(alpha[i] != 0 && alpha[i] != C)
            {
                if(takeStep(i, i2) == 1)
                {
                    return 1;
                }
            }
        }

        // Lặp qua tất cả các ví dụ
        for(int i = 0; i < numAlpha; ++i)
        {
            if(takeStep(i, i2) == 1)
            {
                return 1;
            }
        }
    }

    return 0;
}

void smoAlgorithm()
{
    int numChanged = 0;
    bool examineAll = true;
    int numPoints = points.size();
```

```
        while(numChanged > 0 || examineAll)
        {
           numChanged = 0;

           if(examineAll)
           {
              for(int i = 0; i < numPoints; ++i)
                 numChanged += examineExample(i);

              examineAll = false;
           }
           else
           {
              for(int i = 0; i < numPoints; ++i)
                 if(alpha[i] > 0 && alpha[i] < C)
                    numChanged += examineExample(i);


              if(numChanged == 0)   examineAll = true;

           }
        }
     }
  };
  void introduction()
  {
     cout << GREEN_TEXT << string(19, ' ') << "|" << string(22, ' ') <<
WHITE_TEXT << "DE TAI: "<< BLUE_TEXT << "SUPPORT VECTOR MACHINE"
     << string(22, ' ') << GREEN_TEXT << "|" << endl;
     cout << string(19, ' ') << GREEN_TEXT << "|" << string(18, ' ') <<
BLUE_TEXT << "Sequential Minimal Optimization Algorithm"
```

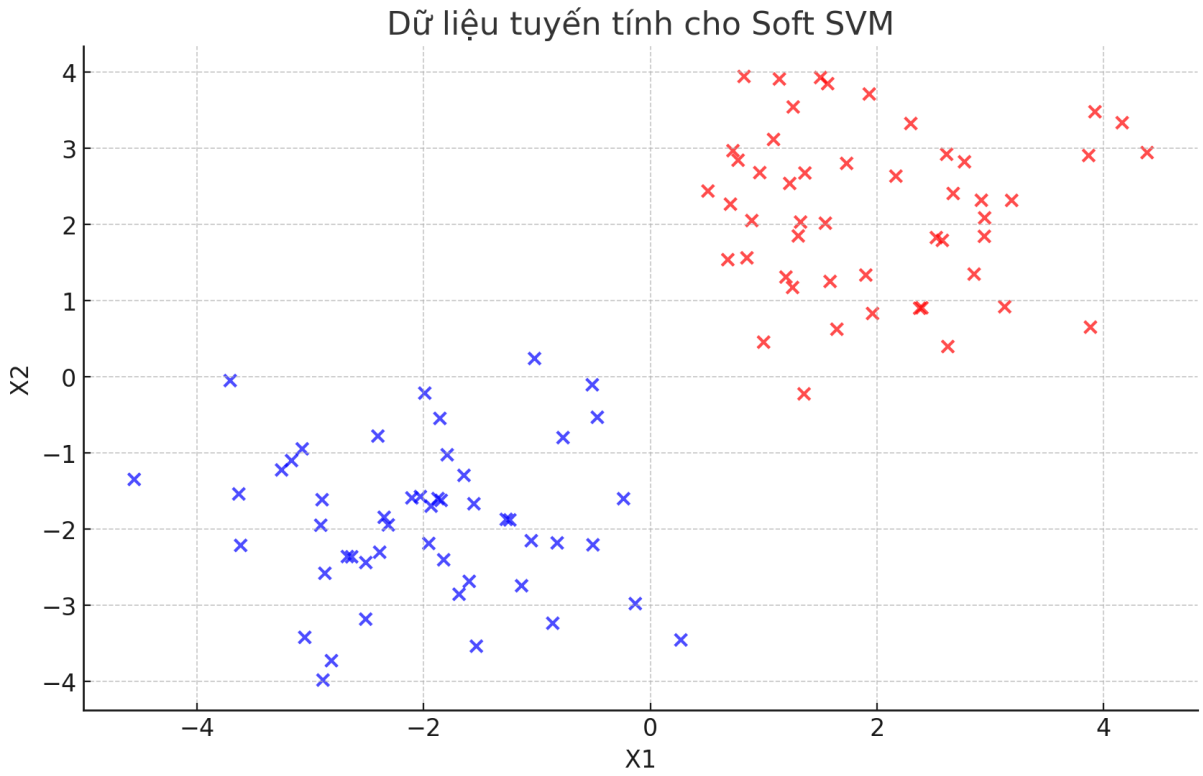
```
        << string(15, ' ')  << GREEN_TEXT << "|" << endl;
        cout << string(19, ' ') << "|" << string(32, ' ') << CYAN_TEXT << "MACHINE LEARNING " << string(25, ' ') << GREEN_TEXT << "|" << endl;
        cout << string(19, ' ') << "|" << string(74, ' ') << "|" << endl;
        cout << string(19, ' ') << "|" << string(24, ' ') << WHITE_TEXT << "GIAO VIEN HUONG DAN: " << YELLOW_TEXT << "NGUYEN TAN KHOI"
        << string(14, ' ') << GREEN_TEXT << "|" << endl;
        cout << string(19, ' ') << "|" << string(10, ' ') << WHITE_TEXT << "SINH VIEN THUC HIEN: "
        << string(43, ' ') << GREEN_TEXT << "|" << endl;
        cout << string(19, ' ') << "|" << string(10, ' ') << YELLOW_TEXT << "NGUYEN DOAN THANH HOANG"
        << string(5, ' ') << "LOP 23T_KHDL2" << string(5, ' ') << "NHOM 1" << string(12, ' ') << GREEN_TEXT << "|" << endl;
        cout << string(19, ' ') << "|" << string(10, ' ') << YELLOW_TEXT << "PHAN VAN HIEU"
         << string(15, ' ') << "LOP 23T_KHDL1" << string(5, ' ') << "NHOM 1" << string(12, ' ') <<GREEN_TEXT << "|" << endl;
        cout << string(19, ' ') << "|" << string(74, ' ') << "|" << endl;
        cout << string(19, ' ') << "|" << string(32, ' ') << YELLOW_TEXT << "K23 DUT" << string(35, ' ') << GREEN_TEXT << "|\n";
        cout << string(19, ' ') << "|" << string(74, ' ') << "|" << endl;
        cout <<  string(19, ' ') << "|";
        for(int i = 0; i < 74; i++) cout << "_";
        cout << "|" << endl;
        cout << CLEAR_TEXT << endl;
    }
    void display(int Data, SVM svm, vector<double> w)
    {
        cout << WHITE_TEXT << string(4, ' ');
        for(int i = 0; i <= 55; i++) cout << "_";
        cout << endl;
```

```
cout << string(4, ' ') << "|" << string(10, ' ') << RED_TEXT << "KET QUA SAU KHI CHAY THUAT TOAN SMO" << string(10, ' ') << CLEAR_TEXT << "|\n";
//cout << string(4, ' ') << "|" << string(55, ' ') << "|";
cout << string(4, ' ') << "|";
for(int i = 0; i < 55; i++) cout << "_";
cout << "|\n";
cout << string(4, ' ') << "|" << string(15, ' ') << RED_TEXT "THONG TIN VE SUPPORT VECTOR " << CLEAR_TEXT << string(12, ' ') << "|\n";
cout << string(4, ' ') << "|";
for(int i = 0; i < 55; i++) cout << "_";
cout << "|\n";
cout << string(4, ' ') << "|" << RED_TEXT << "STT" << CLEAR_TEXT << "|"  << string(5, ' ') << CYAN_TEXT << "dual coefficient" << CLEAR_TEXT
                                                << string(5, ' ') << "|" << string(5, ' ') << CYAN_TEXT << "support vector" << CLEAR_TEXT << string(5, ' ') << "|\n";
cout << string(4, ' ') << "|___|_____________________________" << "|" << "_____________________________|\n";
for(int i = 0; i < Data; i++)
{
   if(svm.alpha[i] > eps)
   {
                                                    cout << string(4, ' ') <<  "|" << CYAN_TEXT << i + 1 << CLEAR_TEXT << " |"
         << string(9, ' ') << fixed << setprecision(3) << svm.alpha[i] << string(12, ' ') << "|";
         cout << fixed << setprecision(2) << string(3, ' ') << svm.points[i][0] << " " <<  svm.points[i][0] << " " <<  svm.points[i][0] << " " <<  svm.points[i][0] << string(2, ' ') << "|\n";
   }
                                                }
cout << string(4, ' ') << "|___|";
```

```
        cout << "_____|" << "_____|\n";
        cout << string(4, ' ') << "|" << string(18, ' ') << RED_TEXT << "THONG TIN VE HYPERPLANE" << CLEAR_TEXT << string(14, ' ') << "|\n";
        cout << string(4, ' ') << "|_____|\n";
        cout << string(4, ' ') << "|" << string(5, ' ') << WHITE_TEXT << CYAN_TEXT << " vector trong so w  " << CLEAR_TEXT << string(5, ' ')
                                                << "|" << string(5, ' ') << CYAN_TEXT << "  HE SO BIAS  " << CLEAR_TEXT << string(5, ' ') << "|\n";
        cout << string(4, ' ') << "|_____|_____|\n";
        cout << string(4, ' ') << "|" << string(4, ' ');
                                                for(auto x : w)
                                                {
           cout << x << " ";
        }
        cout << " " << string(2, ' ') << "|" << string(10, ' ') << svm.bias << string(10, ' ') << "|\n";
        cout << string(4, ' ') << "|_____|_____|\n";
        cout << endl;
    }
    int main()
    {
        introduction();
        vector<Record> records;
        cout <<  YELLOW_TEXT << "a.  TIEN HANH DOC FILE TU TEP DU LIEU \n";
        createRecordVector(records, "iris.tab");
        //int Data; cin >> Data;
```

```
    vector<vector<double>> trainingPoints(records.size(), vector<double>(4)); // Tạo 2D vector
    vector<int> tickets(records.size());

    for(int i = 0; i < records.size(); ++i)
    {
       trainingPoints[i][0] = records[i].sepal_width;
       trainingPoints[i][1] = records[i].sepal_length;
       trainingPoints[i][2] = records[i].petal_width;
       trainingPoints[i][3] = records[i].petal_length;
    }
    string check = records[0].classname;
    for(int i = 0; i < records.size(); i++)
    {
       if(records[i].classname == check) tickets[i] = 1;
       else tickets[i] = -1;
    }

    SVM svm;
    svm.points = trainingPoints; // Khởi tạo `points`
    svm.tickets = tickets;
    svm.alpha = vector<double>(trainingPoints.size(), 0.0);

   // Chạy thuật toán SMO
    cout << "\nb.  TIEN HANH CHAY THUAT TOAN SMO\n";
    svm.smoAlgorithm();
    cout << "    DA CHAY XONG THUAT TOAN SMO!!!\n" << endl;

  // Tính toán và hiển thị vector trọng số w
    vector<double> w(svm.points[0].size(), 0.0);
    for(int i = 0; i < svm.alpha.size(); ++i)
```

```
        {
            for(int j = 0; j < w.size(); ++j)
                w[j] += svm.alpha[i] * svm.tickets[i] * svm.points[i][j];
        }
        cout <<"c.  TIEN HANH XUAT KET QUA CUA THUAT TOAN SMO:\n" <<
endl;
            display(records.size(), svm, w);
        cout <<  YELLOW_TEXT << "d.  KET THUC CHUONG TRINH!!!\n\n";
        cout << string(15, ' ') << CYAN_TEXT <<
            "==================================          THANK
YOU!!! ==================================\n";
                                                cout << CLEAR_TEXT;
    }
```


Dữ liệu tuyến tính cho Soft SVM
X1
X2
4
3
2
1
0
−1
−2
−3
−4
−4
−2
0
2
4