**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO ĐỒ ÁN THỰC TẬP CHUYÊN MÔN**

**ĐỀ TÀI: NGHIÊNG CỨU THUẬT TOÁN SVM VÀ ỨNG DỤNG VÀO BÀI TOÁN NHẬN DIỆN CHỮ SỐ VIẾT TAY**

Giảng viên hướng dẫn: TRẦN THỊ DUNG

Sinh viên thực hiện: NGUYỄN PHÚC HOÀI LINH

Lớp : CNTT

Khoá :58

Tp. Hồ Chí Minh, năm 2017

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO ĐỒ ÁN THỰC TẬP CHUYÊN MÔN**

**ĐỀ TÀI: NGHIÊNG CỨU THUẬT TOÁN SVM VÀ ỨNG DỤNG VÀO BÀI TOÁN NHẬN DIỆN CHỮ SỐ VIẾT TAY**

Giảng viên hướng dẫn: TRẦN THỊ DUNG

Sinh viên thực hiện: NGUYỄN PHÚC HOÀI LINH

Lớp : CNTT

Khoá :58

Tp. Hồ Chí Minh, năm 2017

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI **CỘNG HÒA XÃ HỘI CHỦ NGHIÃ VIỆT NAM**

**PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH** Độc lập – Tự do – Hạnh phúc

# NHIỆM VỤ THIẾT KẾ TỐT NGHIỆP

BỘ MÔN: **CÔNG NGHỆ THÔNG TIN**

-------\*\*\*-------

**Mã sinh viên:**5851071042  **Họ tên SV:** Nguyễn Phúc Hoài Linh

**Khóa:** 58 **Lớp:** CNTT

1. **Tên đề tài**
2. **Mục đích, yêu cầu**
3. **Nội dung và phạm vi đề tài**
4. **Công nghệ, công cụ và ngôn ngữ lập trình**
5. **Các kết quả chính dự kiến sẽ đạt được và ứng dụng**
6. **Giáo viên và cán bộ hướng dẫn**

Họ tên:

Đơn vị công tác:

Điện thoại: Email:

|  |  |
| --- | --- |
| **Ngày tháng 07 năm 2020**  **Trưởng BM Công nghệ Thông tin** | **Đã giao nhiệm vụ TKTN**  **Giáo viên hướng dẫn** |
| **ThS. Trần Thị Dung** |  |

Đã nhận nhiệm vụ TKTN

Sinh viên: Ký tên:

Điện thoại: Email:

**LỜI CẢM ƠN**

Lời đầu tiên, em xin chân thành cảm ơn giảng viên hướng dẫn, cô Trần Thị Dung, , người đã tận tình chỉ bảo và hướng dẫn em trong suốt quá trình thực hiện bài báo cáo bày

Em cũng xin gửi lời cảm ơn đên các thầy, cô bộ môn, các bạn sinh viên lớp CNTTK58, những người đã tạo điều kiện thuận lợi trong quá trình thực hiện bài báo cáo này.

Chắc chắn rằng bài báo cáo sẽ có nhiều điểm sai xót và khuyết điểm. Do đó, em mong quý thầy, cô có thể đóng góp giúp bài báo cáo có thể hoàn thiện.

Em xin chân thành cảm ơn!

**NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

|  |
| --- |
| ***Tp. Hồ Chí Minh, ngày ….… tháng ….… năm ….…***  **Giáo viên hướng dẫn**  **Trần Thị Dung** |

**MỤC LỤC**

[NHIỆM VỤ THIẾT KẾ TỐT NGHIỆP 3](#_Toc46932505)

[CHƯƠNG 1. MỞ ĐẦU 8](#_Toc46932506)

[1.1 Đặt vấn đề 8](#_Toc46932507)

[1.2 Mục tiêu nghiên cứu 8](#_Toc46932508)

[1.3 Phạm vi 8](#_Toc46932509)

[1.4 Cấu trúc báo cáo thực tập chuyên môn 8](#_Toc46932510)

[CHƯƠNG 2. GIỚI THIỆU CHUNG VỀ MACHINE LEARNING 1](#_Toc46932511)

[2.1 Giới thiệu về AI 1](#_Toc46932512)

[2.2 Machine learning (máy học) 1](#_Toc46932513)

[CHƯƠNG 3. THUẬT TOÁN SVM 4](#_Toc46932514)

[3.1 Bài toán tối ưu 4](#_Toc46932515)

[3.2 Thuật toán SVM 7](#_Toc46932516)

[3.3 Xây dựng bài toán tối ưu 10](#_Toc46932517)

[3.4 Xử lý nhiễu bằng phương pháp biên mềm (soft margin) 13](#_Toc46932518)

[3.6 Xử lý phi tuyến tính 17](#_Toc46932519)

[3.7 Ví dụ minh họa thuật toán SVM với thư viện Scikit-learn 19](#_Toc46932520)

[CHƯƠNG 4. ỨNG DỤNG SVM VÀO BÀI TOÁN NHẬN DIỆN CHỮ VIẾT TAY 26](#_Toc46932521)

[4.1 Bài toán nhận diện chữ số viết tay 26](#_Toc46932522)

[4.2 Trích xuất đặc trưng ảnh bằng thuật toán HOG 29](#_Toc46932523)

[4.3 Hạn chế demo 36](#_Toc46932524)

[TỔNG KẾT 38](#_Toc46932525)

**MỤC LỤC HÌNH VẼ**

*Hình 1.1 Miêu tả AI, ML, và DL* 2

Bảng 1.1 Miêu tả dữ liệu huấn luyện của bài toán ước tính giá bán nhà 4

*Hình 1.2 Overfitting và underfitting* 6

*Hình 1.3 Bài toán ước lượng chiều cao qua cân nặng* 7

*Hình 2.1 Giới thiệu về hyperlane* 8

*Hình 2.2 Các đường chia cắt hai lớp* 9

*Hình 2.3 Đường biên phù hợp nhất* 10

*Hình 2.4 Phương trình đường biên* 11

*Hình 2.5 Hai support vector* 13

*Hình 3.1 Điểm nhiễu* 14

*Hình 3.2 Hai class không phân biệt tuyến tính* 15

*Hình 3.2 Hai class không thể tìm đường biên* 17

*Hình 3.3 Biểu diễn các điểm dữ liệu lên không gain ban đầu* 18

*Bảng 2.1 Tọa độ các điểm* 18

*Hình 3,4 Ánh xạ lên không gian mới* 19

*Hình 4.1 Giao diện demo ban đầu* 20

*Hình 4.2 Các điểm dữ liệu thuộc hai lớp được tạo trên demo* 21

*Hình 4.3 Kết quả đường biên được tìm thấy* 21

*Hình 4.4 Biểu diễn các điểm nhiễu* 22

*Hình 4.5 Kết quả đường biên với các điểm nhiễu* 22

*Hình 4.6 Đường biên có điểm nhiễu với kernel là rbf* 23

*Hình 4.7 Đường biên có nhiễu bằng rbf, đã được thay đổi hằng số C* 24

*Hình 4.8 Dữ liệu không phân biệt tuyến tính* 24

*Hình 4.9 Kết quả đường biên với dữ liệu không phân biệt tuyến tính* 25

*Hình 5.1 Sơ đồ huấn luyện model cho bài toán nhận diện chữ số* 26

*Hình 5.2 Sơ đồ quy trình nhận dạng một bức ảnh có chữ viết tay* 27

*Hình 5.3 Minh họa về ảnh xám* 27

*Hình 5.4 Kết quả kiểm thử model* 28

*Hình 5.5 Sai sót trong chương trình* 29

*Hình 5.6 Minh họa về đường biên trong một bức ảnh* 29

*Hình 5.7 Chi tiết bức ảnh với Gradient* 30

*Hình 5.8 Cách tạo một vector đặc trưng trên một cell* 32

*Hình 5.9 Minh họa vector bin* 32

*Hình 5.10 HOG biểu diễn trên ảnh* 34

*Hình 5.11 Sơ đồ huấn luyện model đã thêm bước trích xuất đặc trưng* 35

*Hình 5.12 Sơ đồ nhận diện bức ảnh thêm bước trích xuất đặc trưng* 35

*Hình 5.13 Kết quả kiểm thử model sau khi chỉnh sửa* 35

*Hình 5.14 Chữ số được vẽ lệch phía phải, nên kết quả không đúng* 36

*Hình 5.16 Thay đổi bước tiền xử lý* 37

*Hình 5.17 Ví dụ về tập dữ liệu huấn luyện mới sau khi xử lý* 37

# CHƯƠNG 1. MỞ ĐẦU

### Đặt vấn đề

Thị giác máy tính là một lĩnh vực đã và đang được ứng dụng rộng khắp trong hiều mặt của cuộc sống, là kĩ thuật liên quan đến việc nhận dạng và phân biệt đối tượng trong thực tế thông qua ảnh chụp của chúng. Kĩ thuật này ngày càng được áp dụng rộng rãi vào nhiều lĩnh vực của nông nghiệp như tự động hóa, nông nghiệp chính xác, phân loại nông sản, nhận dạng (loài cây, cỏ dại, trái cây...). Thực chất, đa số bài toán của thị giác máy tính là bài toán phân lớp. Chẳng hạn, việc xác định giới tính của một người thông qua ảnh chân dung của người đó, thực chất là cách xác phân lớp nam hoặc nữ, hay nhận dạng biểu cảm của con người: tức giận, vui vẻ , bất ngờ, lo lắng,…

Chính vì vậy, việc ứng dụng các thuật toán Machine learning vào thị giác máy tính ngày càng được chú trọng. Đặt biệt là các bài toán phân lớp. Trong Machine learning có nhiều bài toán phân lớp chẳng hạn như: kNN, Logistic Regression, SVM,… Nhìn chung các ,thuật toán trên ban đầu giải quyết bài toán phân lớp đơn giản (tức là gồm hai lớp - gọi là binary classification ), ngoài trừ thuật toán kNN. Nhưng sau đó, chúng được cải tiến để giải quyết các bài toán đa lớp (Multi-class).

Trong số các thuật toán kể ở trên, SVM (Support Vector Machine) là một thuật toán dễ sử dụng, đơn giản nhưng lại đem lại một kết quả với hiệu suất cao. Thậm chí khi đem so sánh với các thuật toán khác ở trong việc phân lớp (mặc dù SVM vẫn có thể áp dụng cho các bài toán hồi quy, nhưng em cảm thấy nó vẫn thích hợp cho bài toán phân lớp hơn), thì kết quả của SVM có thể gọi là nhỉnh hơn đôi phần.

Trong nội dung báo cáo thực hành chuyên môn lần này em sẽ tiến hành nghiêng cứu về thuật toán SVM, đồng thời sẽ áp dụng nó cho bài toán nhận diện chữ viết tay bằng một demo nhỏ.

### Mục tiêu nghiên cứu

Mục tiêu đề tài là tiến hành nghiên cứu về thuật toán SVM. Hiểu rõ được ý tưởng triển khai của SVM.

### 1.3 Phạm vi

- Nghiêng cứu về ý tưởng của SVM

- Các cơ sở về toán học

- Phương pháp biên mềm (Soft-margin) nhằm giúp SVM có thể chấp nhận nhiễu

- Phương pháp Kernel trick giải quyết các bài toán phi tuyến tính

### Cấu trúc báo cáo thực tập chuyên môn

* + 1. Chương 1: Mở đầu
    2. Chương 2: Giới thiệu chung về Machine Learning
    3. Chương 3: Thuật toán SVM
    4. Chương 4: Bài toán nhận diên chữ viể tay

# CHƯƠNG 2. GIỚI THIỆU CHUNG VỀ MACHINE LEARNING

### Giới thiệu về AI

Ngày nay, lĩnh vực AI - Artificial Intelligence (Trí tuệ nhân tao) đang phát triển một cách vũ bão và ngày càng len lỏi vào mọi lĩnh vực trong đời sống. Mục đích chính của lĩnh vực AI là tập trung nghiêng cứu và mô phỏng lại quá trình học tập, suy nghĩ va tư duy của con người để máy có thể học. Các khái niệm này lần đầu tiên được John McCarthy, một nhà khoa học máy tính Mỹ, vào năm 1956 tại Hội nghị The Dartmouth.

Trong lịch sử phát triển AI, các nhà nghiên cứu phân thành 4 hướng tiếp cận chính:

- Hành động như người (acting humanly)

- Suy nghĩ như người (thinking humanly)

- Suy nghĩ hợp lý (thinking rationally)

- Hành động hợp lý (acting rationally)

Trong đó, mức độ mô phỏng máy tính như người là khó nhất và đây cũng là mục tiêu mà các nhà khoa học đang hướng tới. Đến năm 1950, mục tiêu trên được “khai phá” nhờ bài kiểm tra “Turing Test”. Bài test này được Alan Turing đề xuất nhằm mục đích để kiểm tra hệ thống máy tính đã đạt đến khả năng thông minh hay chưa. Bài test tiến hành bao gồm một người đặt câu hỏi, một người trả lời câu hỏi và một máy tính phân tích trả lời câu hỏi. Nếu quá trình trao đổi đó, người đặt câu hỏi không còn phân biệt được đâu là câu trả lời của con người đâu là câu trả lời của máy tính, thì máy tính được xem là thông minh. Từ cách tiếp cận này đã cho ra các nhánh sau đây:

- Natural language processing: xử lý ngôn ngữ tự nhiên

- Computer vision : thị giác máy tính

- Knowledge representation: biểu diễn tri thức

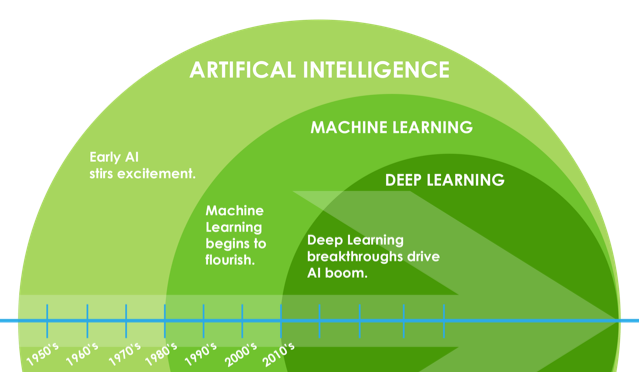
- Machine learnig: máy học

- Automated reasoning: Tự suy luận

- Robotics: Rô bốt học

- ….

### Machine learning (máy học)



##### *Hình 1.1 Miêu tả AI, ML, và DL*

Machine learning là một tập con của trí tuệ nhân tạo. Theo Wikipedia Machine learning được định nghĩa như sau.

“Machine learnig là một lĩnh vực của trí tuệ nhân tạo liên quan đến việc nghiên cứu và xây dựng các kĩ thuật cho phép các hệ thống "học" tự động từ dữ liệu để giải quyết những vấn đề cụ thể. Ví dụ như các máy có thể "học" cách phân loại thư điện tử xem có phải thư rác (spam) hay không và tự động xếp thư vào thư mục tương ứng. Học máy rất gần với suy diễn thống kê (statistical inference) tuy có khác nhau về thuật ngữ.

Học máy có liên quan lớn đến thống kê, vì cả hai lĩnh vực đều nghiên cứu việc phân tích dữ liệu, nhưng khác với thống kê, học máy tập trung vào sự phức tạp của các giải thuật trong việc thực thi tính toán. Nhiều bài toán suy luận được xếp vào loại bài toán NP-khó, vì thế một phần của học máy là nghiên cứu sự phát triển các giải thuật suy luận xấp xỉ mà có thể xử lý được.

Học máy có hiện nay được áp dụng rộng rãi bao gồm máy truy tìm dữ liệu, chẩn đoán y khoa, phát hiện thẻ tín dụng giả, phân tích thị trường chứng khoán, phân loại các chuỗi DNA, nhận dạng tiếng nói và chữ viết, dịch tự động, chơi trò chơi và cử động rô-bốt (robot locomotion).”

Như vậy thông qua định nghĩa trên ta có thể thấy được ML là một lĩnh vực tập trung nghiêng cứu về dữ liệu (data), và từ tập dữ liệu đó (do người lập trình đưa vào học do máy tính tự động thu thập) để từ đó máy tính có thể xử lý các bài toán liên quan.

Phân loại Machine learning. Dựa vào phương thức học ta có thể chia machine learining ra thành 4 loại sau:

- Supervised Learning (Học có giám sát): đây là thuật toán dự đoán đầu ra (outcome) của một dữ liệu mới (new input) dựa trên các cặp (input, outcome) đã biết từ trước. Cặp dữ liệu này còn được gọi là (data, label), tức (dữ liệu, nhãn). Supervised learning là nhóm phổ biến nhất trong các thuật toán Machine Learning. Học có giám sát tiếp tục được chia thành các nhánh khác bao gồm:

+ Các thuât toán Classification (Phân loại): giải quyết các bài toán dạng: “Cái này có dạng là A hay B? ”. Cụ thể là cái bài toán như: xác định giới tính một người, nhận dạng chữ viết tay, xác định email là spam hay không?…. Các thuật toán nổi tiếng trrong nhóm này bao gồm: Logistic Regression, SVM, KNN,….

+ Các thuật toán Regression (Hồi quy): giải quyết các bài toán dạng: “Với dữ liệu như thế này sẽ có kết quả xấp xỉ là bao nhiêu”. Cụ thể là cái bài toán như: “Dự đoán giá nhà với một tập dữ liệu diện tích, khoảng cách đến trung tâm thành phố,….”. Các thuật toán điển hình là: Linear Regression,Stepwise Regression,….

- Học không giám sát: tương tự với học có giám sát, phương pháp học không giám sát cũng tiến hành sát định đầu ra của một dữ liệu mới thông qua tập dữ liệu có sẵn. Tuy nhiên điểm khác biệt là học không giám sát, tập dữ liệu có sẵn (data training) chưa sát định được đầu ra (kết quả). Các bài toán Unsupervised learning được tiếp tục chia nhỏ thành hai loại:

+ Clustering (phân nhóm): Một bài toán phân nhóm toàn bộ dữ liệu thành các nhóm nhỏ dựa trên sự liên quan giữa các dữ liệu trong mỗi nhóm. Ví dụ bài toán xây dựng hệ thống sản phẩm liên quan.

+ Association: Là bài toán khi chúng ta muốn khám phá ra một quy luật dựa trên nhiều dữ liệu cho trước. Ví dụ: những khách hàng nam mua quần áo thường có xu hướng mua thêm đồng hồ hoặc thắt lưng; những khán giả xem phim Spider Man thường có xu hướng xem thêm phim Bat Man, dựa vào đó tạo ra một hệ thống gợi ý khách hàng (Recommendation System), thúc đẩy nhu cầu mua sắm.

- Semi-Supervised Learning (Học bán giám sát): Các bài toán khi chúng ta có một lượng lớn dữ liệu nhưng chỉ một phần trong chúng được gán nhãn (có kết quả sẵn) được gọi là Semi-Supervised Learning. Những bài toán thuộc nhóm này nằm giữa hai nhóm được nêu bên trên.

- Reinforcement Learning (Học tăng cường): là các bài toán giúp cho một hệ thống tự động xác định hành vi dựa trên hoàn cảnh để đạt được lợi ích cao nhất (maximizing the performance). Hiện tại, Reinforcement learning chủ yếu được áp dụng vào Lý Thuyết Trò Chơi (Game Theory), các thuật toán cần xác định nước đi tiếp theo để đạt được điểm số cao nhất.

# CHƯƠNG 3. THUẬT TOÁN SVM

### Bài toán tối ưu

Để có thể tiếp tục bài báo cáo, em xin phép được trình bày về một số khái niệm liên quan đến phương pháp học có giám sát và các khái niệm về bải toán tối ưu.

Đầu tiên, chúng ta sẽ tiến hành tiếp cận phương pháp học có giám sát. Như em đã đề cập trước đó, phương pháp học có giám sát là phương pháp tiến hành dựa phần nhiều vào khoa học dữ liệu. Ta có một tập các dữ liệu đã thu thập từ trước đó chúng bao gồm một tập dữ liệu khác chứa danh sách các thuộc tính, còn lại là một danh sách chứa các đầu ra (outcome) ứng với từng dữ liệu trên. Nhiệm vụ của các thuật toán học có giám sát đó là, từ một đầu vào mới -chúng bao gồm một danh sách các thuộc tính, ta sẽ thu dược một đầu ra ứng với đầu vào đó. Ví dụ với một bài toán dự đoán giá bán của một căn nhà căn cứ vào diện tích, khoảng cách từ căn nhà đến trung tâm thành phố, mức độ an ninh của khu vực xung quanh. Ta có thể nhận thấy, các giá trị của diện tích, khoảng cách, mức độ an ninh,… là các “thuộc tính” tác động trực tiếp đến giá bán của một căn nhà. Như vậy ta có thể hiểu đề bài ở đây chính là, từ một dữ liệu chứa giá trị diện tích, khoảng cách, mức độ an ninh,… hãy tìm một công thức để tính (ước lượng) giá của căn nhà ấy.

Để làm được điều này, nhất là với các bài toán ước lượng, chúng ta cần phải dựa vào một cái gì đó. Lẽ dĩ nhiên trong trường hợp này, thứ chúng ta dựa vào chính là một cơ sở dữ liệu mẫu, nó chứa một danh sách giá trị của các “thuộc tính” diện tích, khoảnh cách, an ninh,… và ứng với mỗi dòng trong danh sách là một cột chứa giá bán căn nhà.

##### Bảng 1.1 Miêu tả dữ liệu huấn luyện của bài toán ước tính giá bán nhà

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| STT | Diện tích (m2) | Khoảng cách(km) | An ninh (mức) | Giá (triệu đồng) |
| 1 | 20 | 12 | 1 | 200 |
| 2 | 22 | 5 | 1 | 400 |
| 3 | 25 | 18 | 1 | 260 |
| 4 | 20 | 12 | 3 | 150 |
| … |  |  |  |  |

Dựa vào dữ liệu trên ta sẽ tiến hành dự đoán một căn nhà có diện tích 30 m2 , khoảng cách đến trung tâm thành phố là 10km, mức độ anh ninh khu vực xung quanh là 1 thì giá căn nhà là bao nhiêu.

Tổng kết lại, mục tiêu chính của bài toán là tiến hành xây dựng một hàm (công thức) để làm được điều đó - ước lượng được giá bán của một căn nhà. Ta có thể thấy rằng, giá của một căn nhà có tỷ lệ thuận với diện tích của nó và đồng thời tỷ lệ nghịch với khoảng cách từ nhà đến trung tâm và mức độ an ninh khu vực (với 1 là mức an toàn nhất), ta có thể đưa ra công thức đơn giản sau:



Trong đó, x1 là diện tích, x2 là khoảng cách tới trung tâm và x3 là mức độ an ninh khu vực. Còn các giá trị a1,a2,a3 là các hằng số sẽ được xác định trong quá trình kiểm thử.

Trong phương pháp học có giám sát, các hàm như trên được gọi là các hàm mục giả thuyết. Đấy là nhưng hàm sẽ giải quyết bài toán ban đầu. Hàm giả thuyết sẽ được xây dựng dựa trên các dữ liệu mẫu có sẵn. Các dữ liệu này được gọi là training data (một số tài liệu viết là training set).

Tuy nhiên trên thực tế, quá trình này không đơn giản. Các hàm giả thuyết không thể nào cho ra các đầu ra đúng 100% như trong training data, mà nó chỉ có thể xấp xỉ gần bằng. Để tiến hành đánh giá hàm giả thuyết đưa ra có độ chính xác đên đâu, để còn có thể sửa chữa. Người ta lại dùng tiếp một tập dữ liệu, đã biết trước đầu ra, để kiểm thử. Tập dữ liệu này gọi là test data. Như vậy trong phương pháp học có giám sát chia ra làm hai giai đoạn riêng biệt tách rời nhau đó là : train (tạo hàm giả thuyết, tức tạo model) và test (kiểm tra lại model).

Vậy tại sao chúng ta không gộp hai giai đoạn này thành một. Tức là gộp dữ liệu của tập huấn luyện và tập kiểm thử, rồi sau khi tiến hành xây dựng xong model, ta sẽ dùng nó để kiểm thử ? Nguyên nhân là bởi vì nếu ta dùng cả hai tập dữ liệu cho cả hai quá trình. Ta có thể sẽ gặp trường hợp, mô hình hay hàm giả thuyết được xây dựng quá “khít” với toàn bộ dữ liệu. Nhưng khi ta dùng với dữ liệu thật lại không thể ra kết quả mong đợi. Nói dễ hiểu hơn là giống như tình trạng học vẹt, học tủ ở trường mình (như môn triết học Mác-Lênin). Thầy cho chúng ta 20 bộ đề để ôn. Chúng ta ôn “tủ” 19 đề, và tất cả đều chắc như bắp. Nhưng đến khi thi thì bị “tủ đè” vì đề thì dùng đề số 20.

Trường hợp trên ta gọi là overfitting. Ngược lại với overfitting ta có underfitting, khi mà hàm ta xây dựng không hiệu quả với tập dữ liệu. Hình ảnh dưới đây sẽ cho cái nhìn rõ hơn về cả hai trường hợp.

##### IMG_256*Hình 1.2 Overfitting và underfitting*

Hai vấn đè trên đều có cách giải quyết, tuy nhiên nó vượt quá bài báo cáo của em nên em xin tạm thời bỏ qua.

Quay trở lại với phương pháp học có giám sát. Em xin tiếp tục nêu thêm một khái niệm về bài toán tối ưu, hoặc quy hoạch toán học:

***Cho trước***: một hàm f: A → R từ tập hợp A tới tập số thực

***Tìm***: một phần tử x0 thuộc A sao cho f(x0) ≤ f(x) với mọi x thuộc A ("cực tiểu hóa") hoặc sao cho f(x0) ≥ f(x) với mọi x thuộc A ("cực đại hóa").

Trong đó, A của hàm f được gọi là không gian tìm kiếm, thường được xác định bởi một tập các ràng buộc, các đẳng thức hay bất đẳng thức mà các thành viên của A phải thỏa mãn. Các phần tử của A được gọi là các lời giải khả thi, gọi là các đối số tối ưu. Hàm f được gọi là hàm mục tiêu, hoặc hàm chi phí. Lời giải khả thi nào cực tiểu hóa (hoặc cực đại hóa, nếu đó là mục đích) hàm mục tiêu được gọi là lời giải tối ưu.Các bài toán tối ưu thường được ký hiệu như sau:

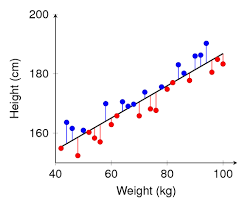
+ Với bài toán tìm max/min:  . Ký hiệu này biểu diễn bài toán tìm giá trị nhỏ nhất của x2  + 2 trên tập số thực R.

+ Đối số tối ưu: . Kí hiệu biểu diễn cho các nghiệm tìm được cho bài toán tìm min của hàm x2 +2, là tập từ (-○○ , -2].

Em đã trình bày xong về các kí hiệu mà em sẽ dùng cho phần tiếp theo của báo cáo. Tuy nhiên có một vấn đề mà em càng làm rõ, đó là giữa các thuật toán học có giám sát và bài toán tối ưu có liên quan gì với nhau. Đầu tiên chúng ta cần nhắc lại rằng, mục tiêu cốt lỗi của các thuật toán học có giám sát là xây dựng được một giả thuyết (model) mà đầu ra của nó là một giá trị cần xác định hay ước lượng như giá bán nhà. Nếu nhìn lại công thức:



Thì ta chỉ cần thay đổi một trọng số trong a0, a1,a2,a3 ,thì ta sẽ có một hàm mới. Điều đó có nghĩa là một bài toán chúng ta còn rất nhiều cách giải khác nhau. Bài toán lại chuyển thành việc tìm tập hợp a. Vấn đề là làm sau ta có thể chọn ra được tập a phù hợp nhất, cho ra kết quả gần sát nhất. Một hướng tiếp cận cho việc này đó là quy đổi nó sang một bài toán tìm giá trị lớn/nhỏ nhất. Ví dụ một bài toán hồi quy, từ cân nặng có thể ước lượng chiều cao của một người:



##### *Hình 1.3 Bài toán ước lượng chiều cao qua cân nặng*

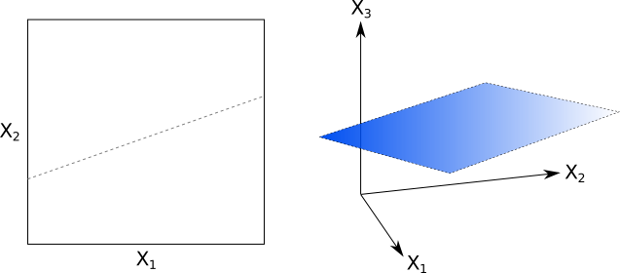
Rõ ràng ta có thể vẽ vô số các đường thẳng trong đồ thị trên, nhưng đường thẳng phù hợp nhất thì lại khác. Chúng ta có thể xét rằng đường thẳng phù hợp nhất chính là đường thẳng gần với tất cả các điểm dữ liệu nhất. Tức là tổng khoảng cách từ đường thẳng đến các điểm là nhỏ nhất. Khoảng cách ở đây ta có thể hiểu là độ chênh lệch giữa đầu ra của hàm giả thuyết với chiều cao thực. Vậy ta có thể chuyển thành bài toán thành

***Tìm***: 

***Sao cho***: 

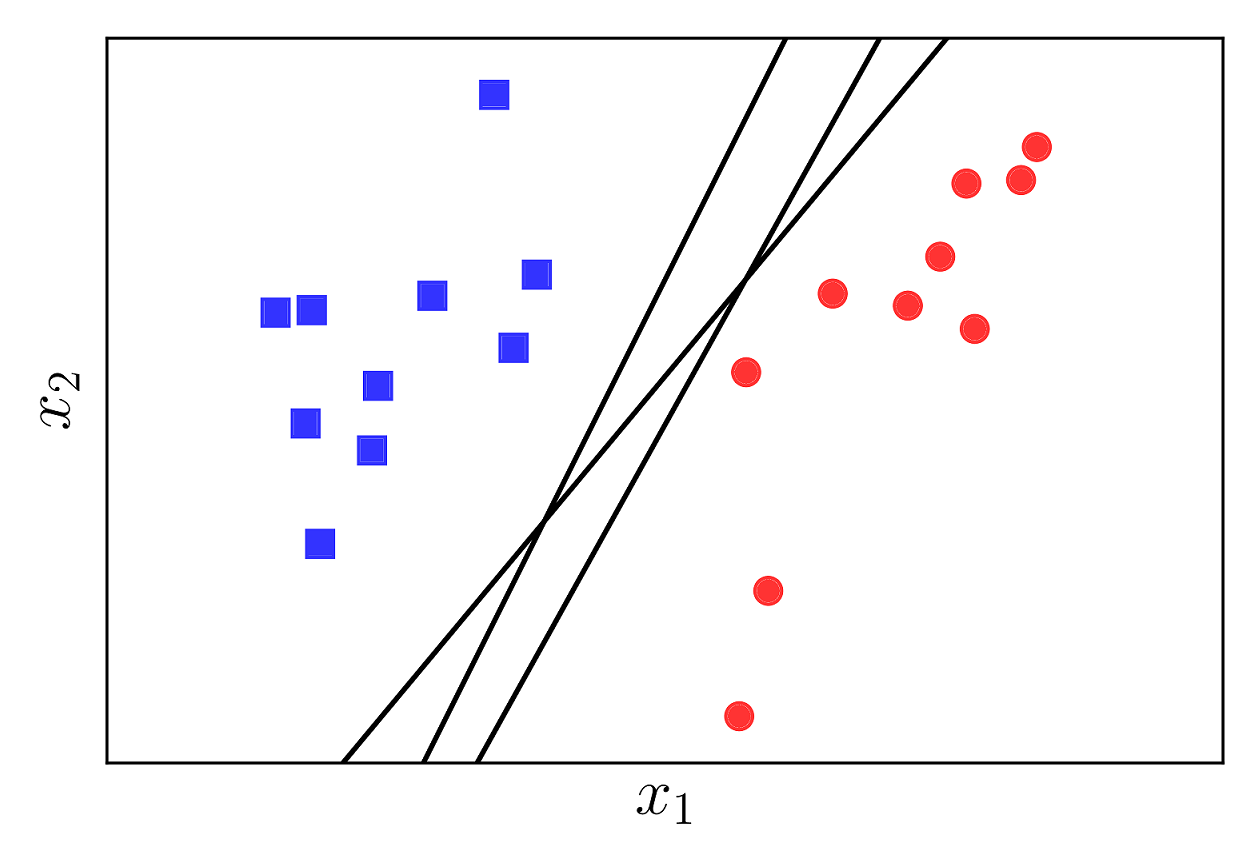
Trong đó, w là cân nặng, h là chiều cai. Wi và hi là dữ liệu thứ i trong tập dữ liệu, ứng với một điểm trong đồ thị.

### Thuật toán SVM

Thuật toán SVM là một trong những thuật toán phân lớp trong phương pháp học có giám sát. Nó giúp việc phân tách hai lớp (class) thành hai phần riêng biệt. Giả sử, ta có hai class được biểu diễn bởi các điểm trong một không gian nhiều chiều. Hai class này phân biệt tuyến tính với nhau, tức là giữa các điểm trong hai tập hợp tồn tại một mặt phẳng/đường thẳng chia không gian gốc thành hai phần, chính là phạm vi của hai class. Mặt phẳng này có số chiều nhỏ hơn không gian gốc. Ví dụ nếu không gian gốc có số chiều là 2 thì thứ chia tách hai lớp là một đường thẳng, với số chiều không gian gốc là 3 thì sẽ là một mặt phẳng, với số chiều là p thì sẽ làm một không **gian p-1 chiều.

##### *Hình 2.1 Giới thiệu về hyperlane*

Quay trở lại với bài toán, ta giả sử hai class của chúng ta được biểu diễn bởi một tập các điểm nằm trên không gian hai chiều. Như vậy sẽ có một đường thẳng tiến hành chia cắt hai class đó. Tuy nhiên, như em đã nói ở mục trước, ta sẽ có vô số các đường có thể đuợc vẽ ra với các hằng số a0,a1,a2,a3,…

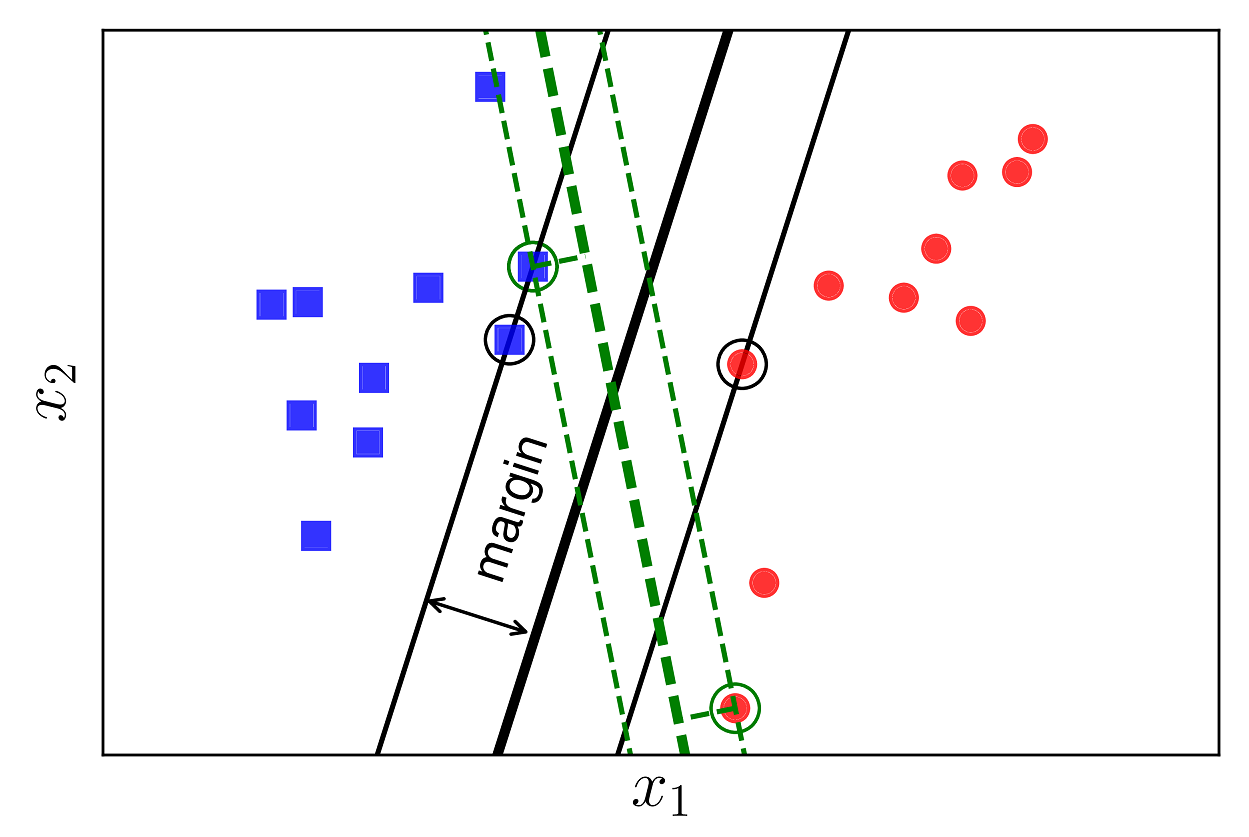


##### *Hình 2.2 Các đường chia cắt hai lớp*

Vậy đâu là đường thẳng tốt nhất trong tất cả các đường thẳng trên.

Việc ta xác định đường thẳng phân chia giống như chúng ta xác định đường biên cho phạm vi hai class. Đường biên phải làm sao cho phạm vi của hai class không bị thiệt quá nhiều. Chú ý phạm vi của một class phụ thuộc vào khoảng cáchngắn nhất của điểm thuộc một class đến đường thẳng phân chia. Xét ví dụ ở trên ta nhận thấy có một đường phân chia bị lệch về phía class đỏ (khoảng cách từ điểm gần nhất của class đỏ nhỏ hơn điểm gần nhất của class xanh), nên đường thẳng này không phù hợp. Vậy đường thẳng ta cần tìm phải có khoảng cách tới các điểm gần nhất là tương đương nhau. Thế mới công bằng!

Tuy nhiên, nhiêu đó cũng chưa đủ. Ta xét trường hợp tiếp theo:



##### *Hình 2.3 Đường biên phù hợp nhất*

Trong hinh ảnh trên tồn tại hai đường thằng lục và đen. Cả hai đường thẳng này đều chia hai class thành hai phần và khoảng cách từ điểm gần nhất đến cả hai đuờng thărng đều bằng nhau. Tuy nhiên, đường thẳng màu đen sẽ là đườg thẳn phù hợp nhất trong truờng hơp này. Lí do bởi vì no có thể chía hai class ra xa nhau nhất có thể. Điều này có lợi cho việc xác định hai class. Cũng như đường biên giới giữa hai quốc gia, chỗ nào hẻo lánh thưa dân sẽ nảy sinh ít vấn đề hơn chỗ biên giới gần các thành phố lớn.

Như vậy ý tưởng của thuật toán SVM đó là xác định một hyperlane của không gian gốc thoản mãn các điều kiện sau:

- Chia cắt hai class thành phần riêng biệt

- Có khoảng cách đến các điểm gần nhất thuộc hai class tương đương nhau.

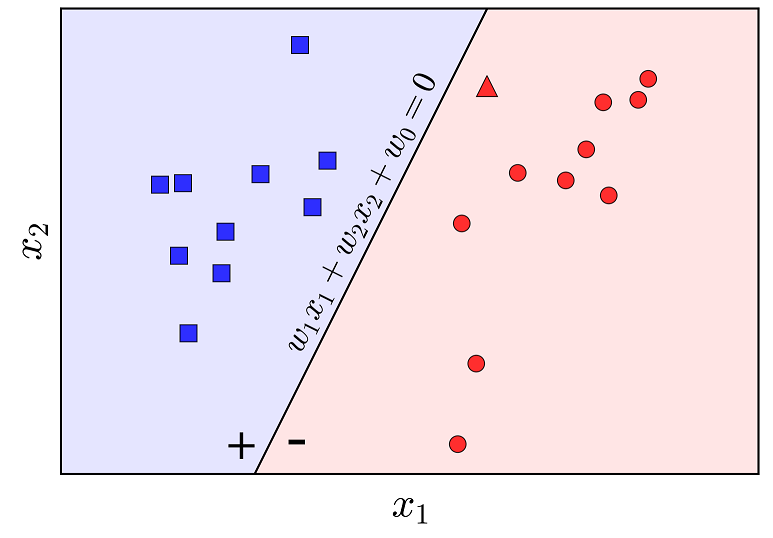
- Chia tách hai class càng xa nhau càng tốt.

### Xây dựng bài toán tối ưu

Giả sử tập các dữ liệu đầu vào cảu một training set là một ma trận hai chiều X = {x0 ,x1 ,x2 ,…xn} với mỗi hàng xi , là một vector biểu diễn một điểm trên đồ thị. Đồng thời, ứng với mỗi dóng, sẽ là một đầu ra yi tương ứng. Chúng ta sẽ gọi chúng là nhãn (label), chúng sẽ được dùng để đánh dấu cho các class. Do ở đây ta chỉ xét trên hai class nên vector y chỉ có hai giá trị: y = -1 cho class màu đỏ, ngược lại y = 1 cho class màu xanh.

Như vậy ta có thể hình thành được công thức ban đầu cho đường biên cho chúng ta:



Ở đây, x1,x1,x2,…,xn là các thuộc tính biểu diễn một điểm dữ liệu tức Xi.(với x0 -bây giờ là 1).Vector W ={w0,w1,w2,….,wn} là danh sách các hằng số được thêm vào. Với không gian thuộc tính là 2 thì hàm này sẽ có dạng là 

##### *Hình 2.4 Phương trình đường biên*

Như vậy là chúng ta đã xác định được công thức đường biên cần tìm. Từ đây ta có thể xác định được một diểm dữ liệu mới thuộc lớp nào, với hàm f(X) >= 0 sẽ cho đầu ra là 1 tức thuộc lớp xanh và ngược lại, các điểm cho ra f(X) <0 sẽ thuộc lớp đỏ.

Bước tiếp theo đẻ có thể hoàn thiện công thức, đó là xác định tiếp các hằng số w0,w1,w2,….,wn, chính là vector W. Làm được điểu này chúng ta cần phải nắm bắt lại một số kiến thức sau. Khoảng cách từ một điểm đến một đường thẳng sẽ là :



Trong không gian ba chiều tới một mặt phẳng có phương trình f(X)=w0 + w1x1 + w2x2 +w3x3 được xác định bởi



Ta có thể được tổng quát lên không gian nhiều chiều: Khoảng cách từ một điểm (vector) tới siêu mặt phẳng (hyperplane) có phương trình f(X)=w0 + w1x1 + w2x2 +w3x3 +…+wnxn = WTX được xác định bởi:



Với ||w||2 là chiều dài vector, WT là ma trận chuyển vị của W, X là tập đầu vào (x0 =1).

Áp dụng công thức trên vào bài toán của SVM. Ta nhận thấy một điều giá trị hàm f(x) trên mỗi hàng X luôn cùng dấu với label tương ứng, do đó khi ta nhân chúng lai với nhau kết quả luôn luôn dương. Vì vậy ta hoàn toàn có thế viết lại công thức trên thành.



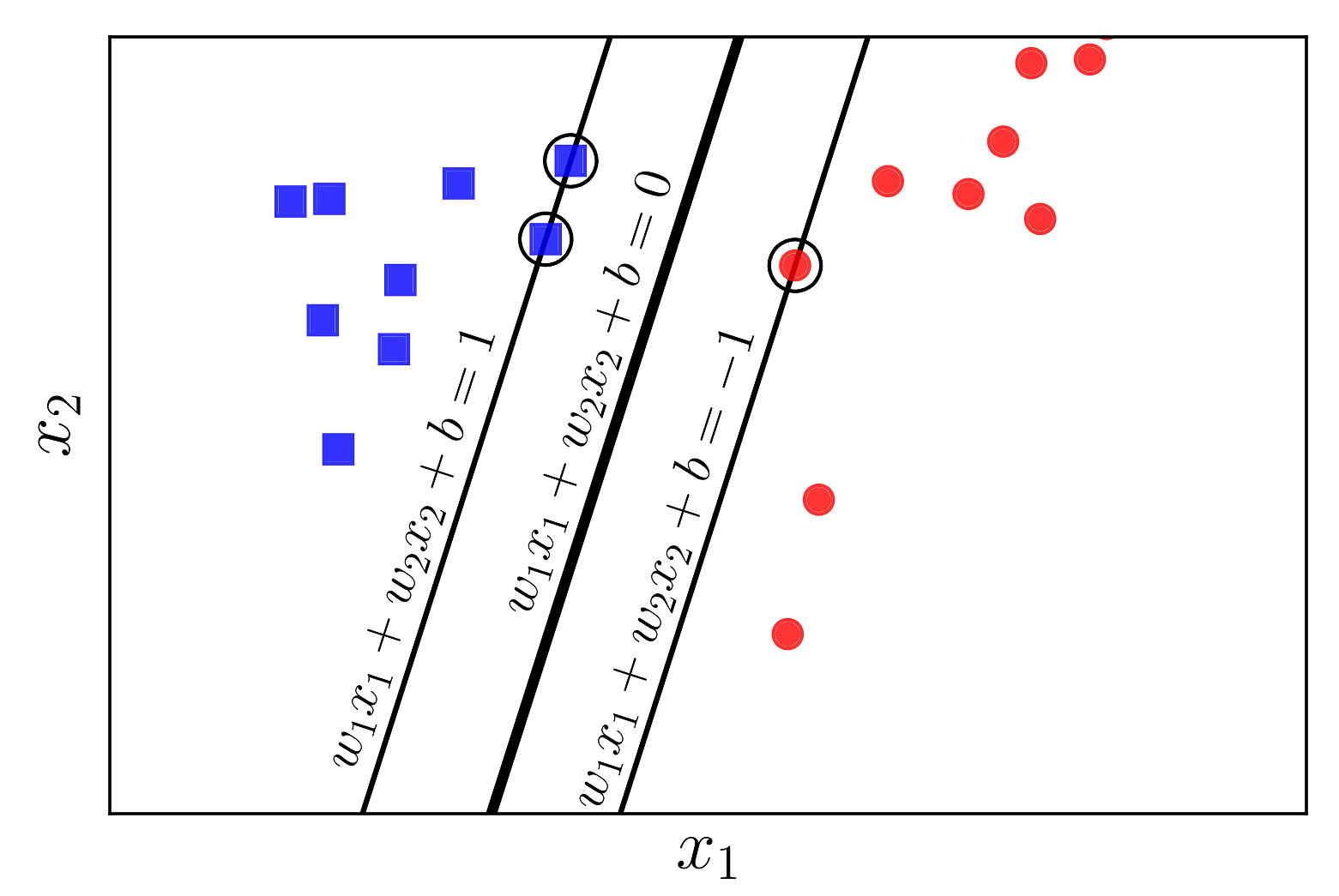
Nhắc lại về ý tưởng của SVM đó là tìm một hyperlane của không gian thuộc tính sao cho khoảng các tới các điểm gần nhất thuộc hai class là bằng nhau và có giá trị là lớns nhất. Khoảng cách này trong SVM gọi là margin và chúng được biểu diễn biểu diễn như sau:



Lời giải cho bài toán tìm W chính là các đối số tối ưu của bài toán trên:



Để có thể đơn giản hóa vấn đề, người ta sẽ giả dụ có các điểm dữ liệu gần nhất nằm ở các đường thẳng song song với đường biên. Lần lượt được biển diễn bởi WTX = -1 và WTX = 1. Chúng ta gọi chúng là các Support Vector. Lưu ý dựa vào hai công thức em vừa đề cập, không có nghĩa là khoảng các của hai Support Vector đến đường biên là 1. Mà nó còn tùy thuộc vào giá trị của W.



##### *Hình 2.5 Hai support vector*

Với việc giá trị nhỏ nhất yi(WTX) = 1 ta sẽ có được công thức sau:



Bài toán bây giờ ta nghịch đảo ||w||2 hàm sẽ trở thành tìm min, thêm nhân thêm 1/2 để

Dễ tính đạo hàm.

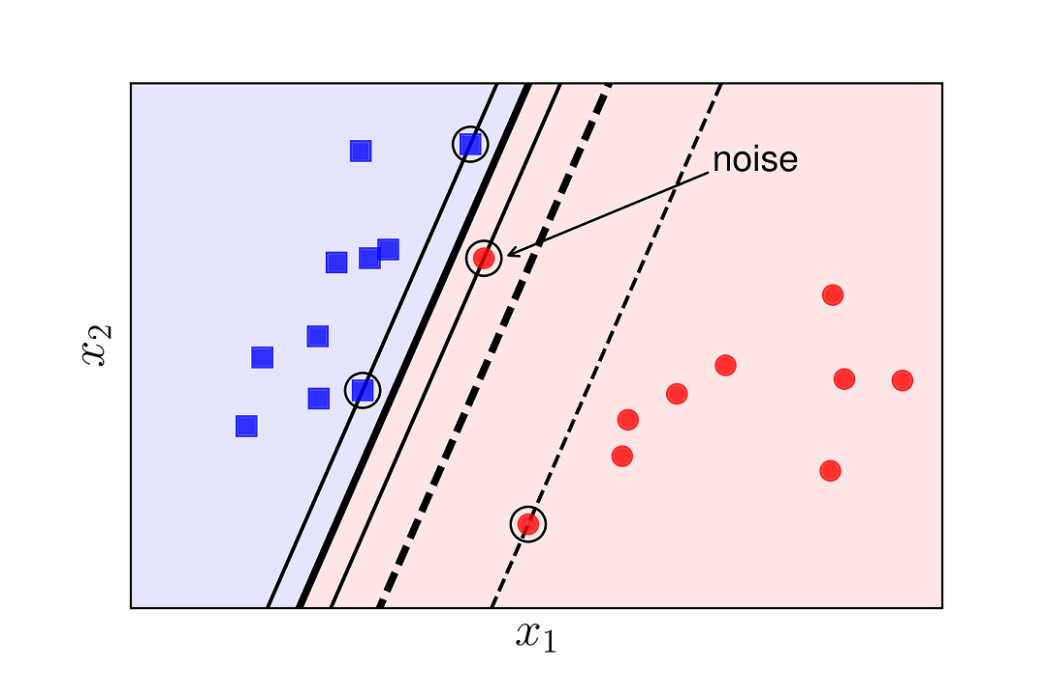


Việc giải bài toán trên liên quan đến một số lý thuyết toán học tương đối phức tạp như điều kiện Karush-Kuhn-Tucker, hàm đối ngẫu Lagrange, Convex optimization … Em sẽ không nói ở bài báo cáo này.

### Xử lý nhiễu bằng phương pháp biên mềm (soft margin)

Phần tiếp theo của báo cáo em xin trình bày cách xữ lý các điểm nhiễu, bằng phướng pháp soft margin.

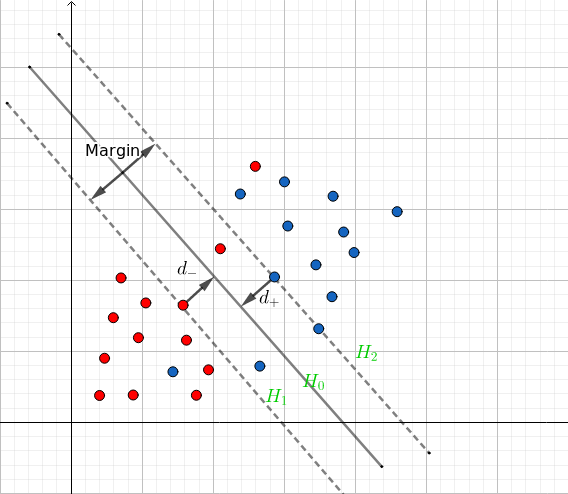
Như chúng ta đã biết, thuật toán SVM nói riêng cũng như các thuật toán phân loại khác nói chung sẽ tiến hành chia tách hai hay nhiều lớp. Nếu biểu diễn các điểm dữ liệu của các lớp này lên không gian (ta sẽ xét trên không gian 2 chiều), ta sẽ nhận thấy một điều. Đó là không phải lúc nào các lớp cũng nằm tách biệt nhau hoàn toàn, mà sẽ có các điểm sẽ nằm “nhằm vị trí” class. Các điểm đó được gọi là các điểm nhiễu. Các hình sau đây sẽ cho ta được cái nhìn tổng quan về vấn đề này.



##### *Hình 3.1 Điểm nhiễu*

Quan sát bức hình ta sẽ có các nhận xét như sau. Cả hai class xanh-đỏ đều được coi là đã phân biệt tuyến tính. Dễ thấy điều đó vì ta có thể hoàn toàn kẻ một (hoặc vô số) đường thẳng để tạo thành hai mặt phẳng riêng biệt chứa hai class. Tuy nhiên, hyperlane ta tìm được lại có độ rộng, tức margin quá nhỏ. Nguyên nhân dẫn đến đều này nằm từ phóa một điểm đó nằm quá gần class xanh. Vì vậy nếu ta có thể bỏ qua điểm này, coi nó như không tồn tại, để xét tìm hyperlane cho tập dữ liệu ta sẽ nhận được một đường thằng, chính là đường nét đứt chính giữa.

Một trường hợp khác mà ta có thể bắt gặp thường xuyên ngoài thực tế.



##### *Hình 3.2 Hai class không phân biệt tuyến tính*

Quan sát bức hình ta nhận thấy một điều. Đó chính là khi ta áp dụng thuật toán SVM cũ, thì với bài này chúng ta hoàn toàn không tìm được đáp án. Bởi vì, ngay từ đầu hai lớp mà chúng ta đang xét chẳng hề phân biệt tuyến tính. Và chính vì không phân biệt tuyến tính nên chẳng có đường thẳng nào chia đôi hai class ra cả (ba điều kiện của hyperlane mà em đã nói ở mục trên). Tuy nhiên, nếu ta có thể bỏ qua được các điểm nhiễu và lựa chọn support vector sao cho hợp lý, thì ta sẽ được kết quả là một hyperlane như trong hình.

Qua hai ví dụ trên ta phần nào có thể hình dung được những hạn chế mà thuật toán SVM truyền thống (Hard Margin) gặp phải. Cũng chính những khuyết điểm đó đã buộc các nhà nghiêng cứu cải tiến thuật toán SVM, cho chúng có khả năng bỏ qua các điểm nhiễu, những điểm không quan trọng đối với class. Hay nói cách khác SVM cho phép xảy ra ngoại lệ. Chính vì vậy mà người ta nói rằng, SVM là một thuật toán chịu lỗi.

Phương pháp giúp giải quyết các nhược điểm của thuật toán SVM truyền thống được gọi là Soft-margin (biên mềm).

Như đã đề cập ở trên, để có thể giải quyết các trường hợp thực tê. Chúng ta cần phải bỏ qua một vài điểm nhiễu. Tuy nhiên, nếu số lượng các điểm bị bỏ qua quá nhiều thì hyperlane của chúng ta sẽ có margin vô cùng lớn và các điểm dữ liệu đều nằm bên trong hai support vector. Như vậy nếu ta tiến hành đếm số lượng đó , thì số lượng phải là nhỏ nhất. Tất nhiên nó sẽ chẳng thể bằng không, vì như thế bài toán sẽ quay lại ban đầu , SVM sẽ chẳng bỏ qua được điểm nhiễu nào.

Vậy điều kiện của đường biên hay hyperlane khi này sẽ là:

- Chia cắt hai class thành phần riêng biệt

- Có khoảng cách đến các điểm gần nhất thuộc hai class tương đương nhau.

- Chia tách hai class càng xa nhau càng tốt.

- Số lượng bỏ qua các điểm nhiễu là thấp nhất.

Nhưng nếu ta tiến hành đếm các điểm nhiễu thì sẽ gặp trường hợp này. Có hai hyperlane chúng đều tiến hành bỏ qua hai điểm nhiễu. Nhưng ở hyperlane đầu nó tạo ra một điểm nhiễu nằm phía trong support vector. Tức là điểm này vẫn nằm ở phía bên class của nó, nhưng chỉ tiếc là nó nằm hơi gần hyperlane.Còn hyperlane còn lại thì tạo ra một điểm nhiễu nằm xa tít bên class bên kia. Như vậy, rõ rang hyperlane thứ nhất tốt hơn cái thứ hai, nhưng vì cách giải quyết là đêm điểm nhiễu nên giá trị của cả hai điều là 1.

Vì điều đó nên ta sẽ tiến hành điều chỉnh một chút cho phù hợp. Thay vì điểm thì ta sẽ tiến hành cộng, giá trị cộng sẽ tùy thuộc vào khoảng cách từ (chênh lệch đầu ra thì đúng hơn như em gọi vậy cho tiện ) nó đến support vector tương ứng, Nếu các điểm nào nằm đúng class thì giá trị này bằng 0 (đương nhiên vì nó không phài là điểm nhiễu). Còn các điểm nằm sai class hay quá gần hyperlane thì giá trị này được tính bằng |WTX-yi | (nhắc lại tập X đã them X0 = 1 ).

Khi đó công thức bài toán tối ưu ban đầu là :

Sunject to: yn(WTX) >= 1

Thì giờ bài toán sẽ là

Sunject to: yn(WTX) >= 1 - ξi

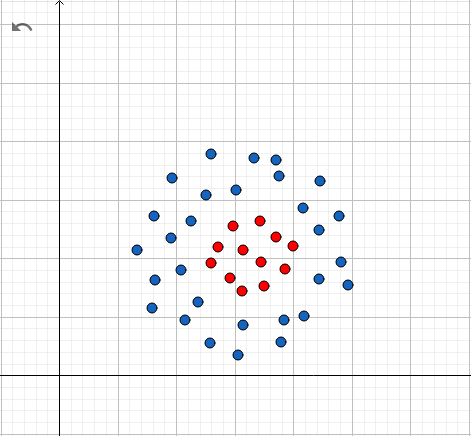
Trong đó giá trị ξ được gọi là slack variable, từng giá trị của nó được tính như những gì em đã nói ở trên. Còn C là một hằng số điều chỉnh margin theo giá trị “bỏ qua”.Quy ước:

* + C =∞ :Không cho phép sai lệch, đồng nghĩa với Hard Margin.
  + C lớn : Cho phép sai lệch nhỏ, thu được Margin nhỏ.
  + C nhỏ: Cho phép sai lệch lớn, thu được Margin lớn.

Bài toán này sẽ được giải bằng phương pháp đối ngẫu nhưng như những gì em đã nói trước đây việc giải nó cần 1 số kiến thức toán phúc tập nên em tạm bỏ qua phần này.

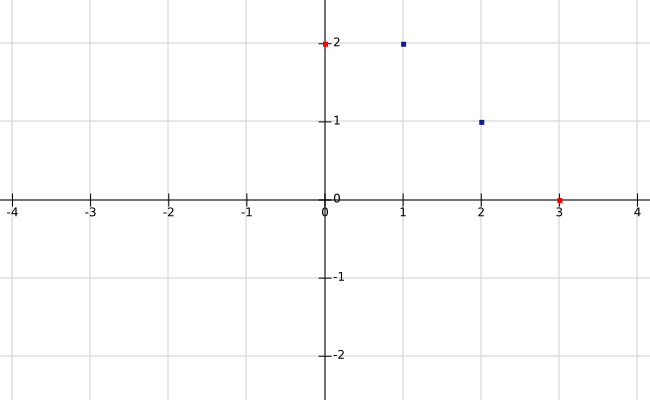
### Xử lý phi tuyến tính

Đây là một trường hơp em đã nói ở mục trước. Tuy nhiên thay vì nó có thể tìm ra hyperlane trường hợp này, tập dữ liệu lộn xộn vào nhau.Chẳng thể nào bỏ qua một vài điểm mà có khi còn phải bỏ qua hết



##### *Hình 3.2 Hai class không thể tìm đường biên*

Để giải quyết trong trường hợp này, ta sẽ chuyển không gian ban đầu thành một không gian có số chiều cao hơn. Ví dụ như sau:



##### *Hình 3.3 Biểu diễn các điểm dữ liệu lên không gain ban đầu*

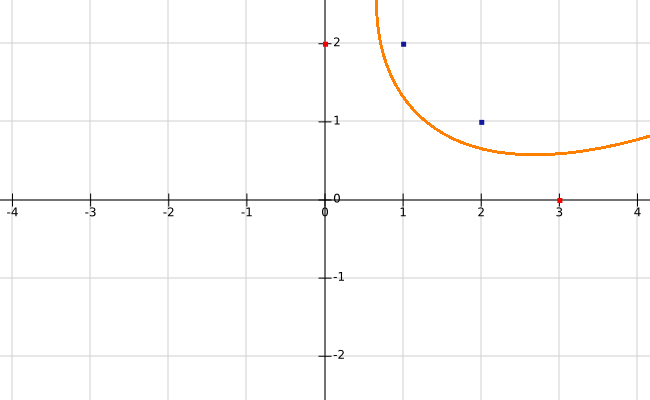
Trong hình, ta nhận thấy các điểm đỏ và xanh hoàn toàn có thể vẽ một đường thằng chia chai class đó. Tuy nhiên vì đây làm một ví dụ về việc ánh xạ sang không gian khác nên em sẽ không tiếp cận theo cách này. Ta sẽ chuyển toàn bộ các điểm trên hình sang không gian 3 chiều. Với chiều z sẽ là một giá trị của một hàm có đầu vào là các chiều ban đầu (ở đây là x và y). Vậy hàm ánh xạ cho z là gì.

Bằng một vài phép tính ta nhận thấy điều này. Nếu ta đêm nhân nhân hai chiều x và y, ta sẽ nhận được một giá trị mà giá trị này lại có thể tiến hành tách biệt cả hai class. Cụ thể như sau:

##### *Bảng 2.1 Tọa độ các điểm*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Class** | **x** | **y** | **z** |
| 1 | Đỏ | 0 | 2 | 0 x 2 = 0 |
| 2 | Đỏ | 3 | 0 | 3 x 0 = 0 |
| 3 | Xanh | 2 | 1 | 2 x 1 = 2 |
| 4 | Xanh | 1 | 2 | 2 x 1 = 2 |

Các điểm thuộc class đỏ có cùng kết quả =0, các điểm thuộc class xanh có cùng kết quả = 2. Như vậy các class đã có một không gian riêng khi ta tiến hành ánh xạ lên không gian khác. Tiếp theo, ta tiến hành phân tách hai class đó trên không gian mới, rồi kết quả lại được đem ánh xạ về không gian cũ. Ở ví dụ này, chúng ta sẽ có đường biên như hình (đây chỉ là minh họa, chứ không phải kết quả thật của tập dữ liệu này)



##### *Hình 3,4 Ánh xạ lên không gian mới*

Trong SVM các hàm như chúng ta đã xét được gọi là các hàm nhân (Kernel functions) . Các hàm nhân thường dùng bao gồm:

Linear: K(x,z) = xTz

Polynomial: K(x,z)=(r+γ xTz)d

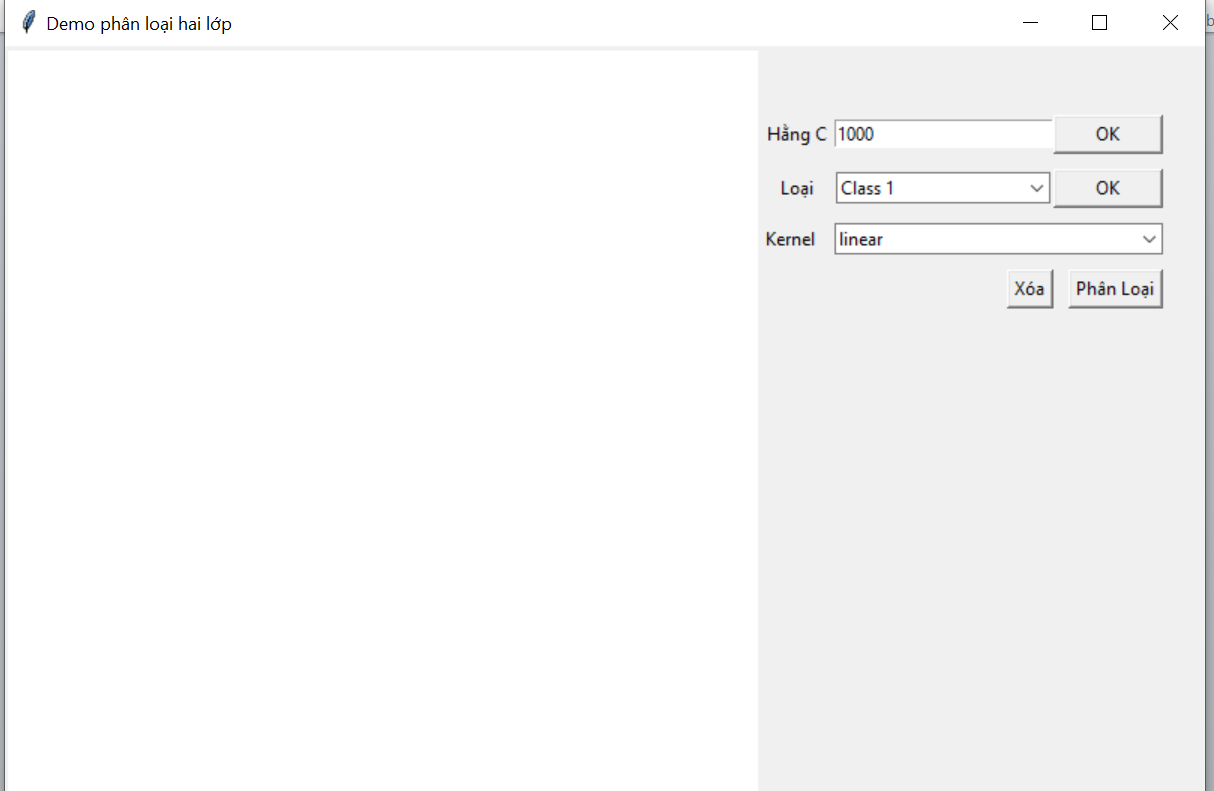
Gaussian RBF: K(x,z) = exp (−γ |),  γ>0

Sigmoidal: K(x,z)=tanh(γxTz+r)

### Ví dụ minh họa thuật toán SVM với thư viện Scikit-learn

Trong phần này em xin phép được trình bày về môt demo nhỏ do em viết ra trên python. Demo này được xây dựng bằng thư viện sklearn hay còn được gọi là scikit-learning. Về phần giao diện, do chỉ là một cửa sổ đơn gian không quá cầu kì, em sử dựng thư viện Tkinter có sẵn trong python 3. Em đã up video lên youtube để tiện theo dõi: <https://www.youtube.com/watch?v=vvqSrMcTtAA> .

Giao diện demo đơn giản như sau:

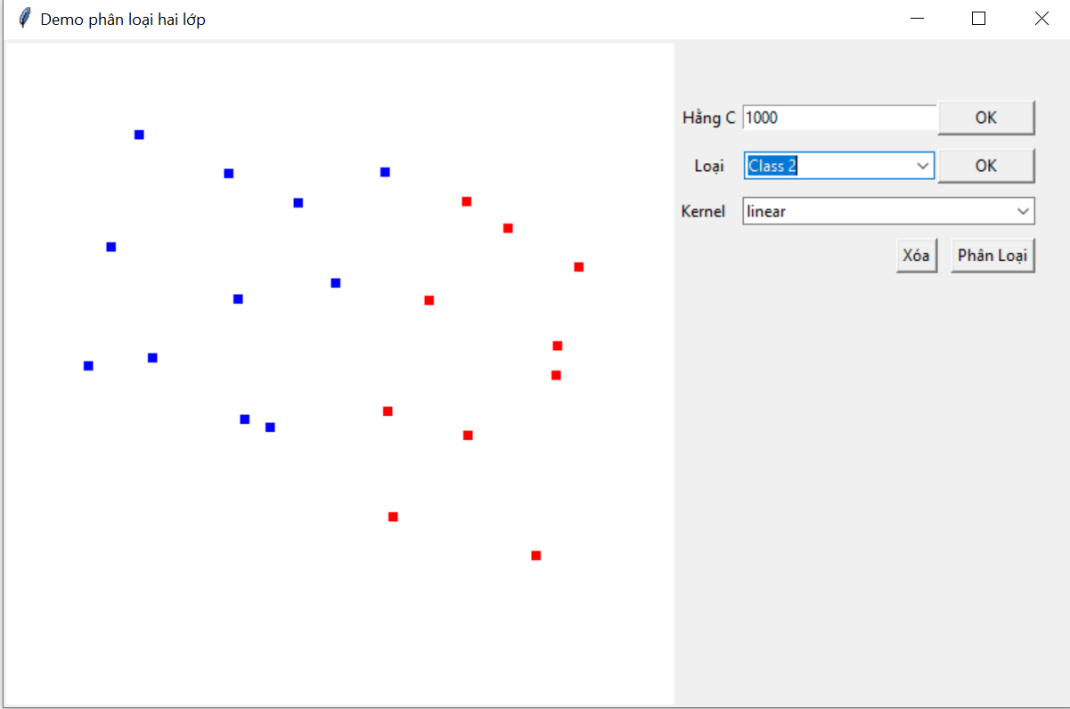


##### *Hình 4.1 Giao diện demo ban đầu*

Bên trái là một canvas màu trắng. Nơi đây sẽ là nơi ta vẽ các điểm đại diện cho hai class. Bên phải là một Frame chứa các control đơn giản bao gồm:

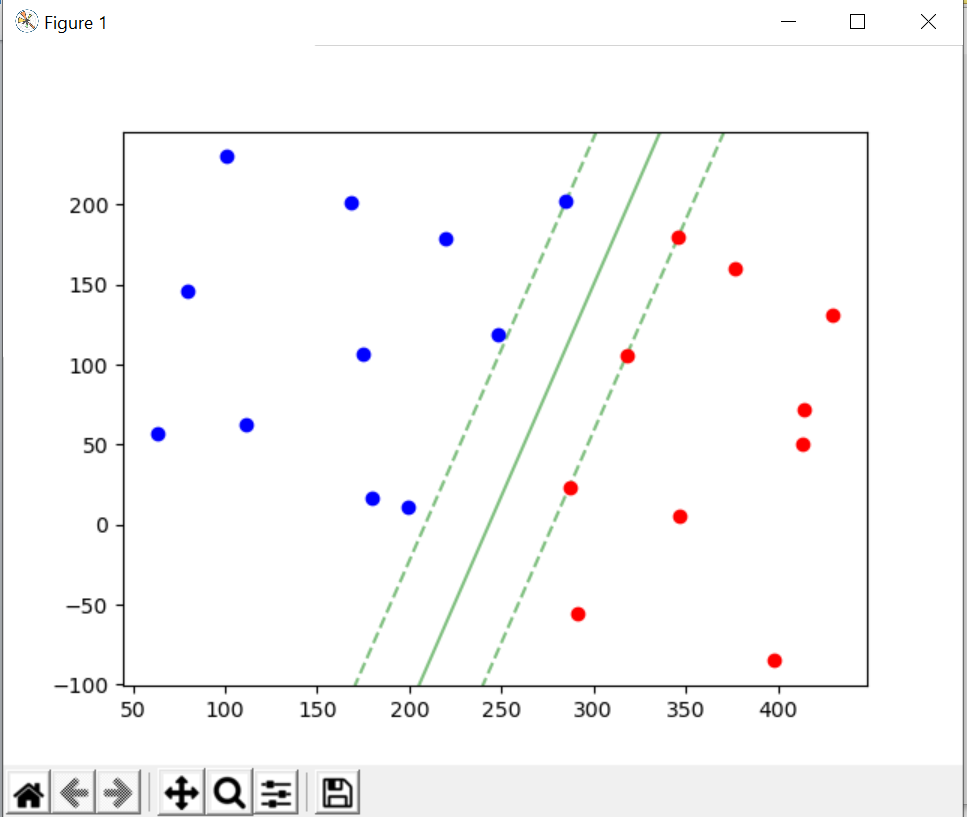
* + Một Entry: để nhập giá trị của C. Chính là giá trị thể hiện quan hệ giữa margin và số điểm nhiễu bỏ qua mà em đã trình bày trước đây. Mặc định giá trị này là 1000. Lưu ý khi thay đổi phải nhấn nút OK bên cạnh để chương trình có thể biết.
  + Một combobox chứa các class, ở đây em chỉ thưc hiện trên hai class nên chỉ có hai giá trị là “class 1 ” và “class 2”. Tương ứng lần lượt là các diểm màu xanh và màu đỏ. Lưu ý khi thay đổi phải nhấn nút OK bên cạnh để chương trình có thể biết.
  + Một combobox chứa các hàm nhân (hàm kernel), do đơn giản hóa nên em chỉ xét hai loại kernel là “linear” và “rbf” (Gaussian RBF) .
  + Cuối cùng là nhóm các button điều khiển. Bao gồm nút ”Xóa”, để xóa toàn bộ các điểm mà mình đã vẽ. Nút “Phân loại” để thực hiện việc tìm hyperlane.

Ta giả sử các điêm thuộc hai class được bố trí nhau sau:



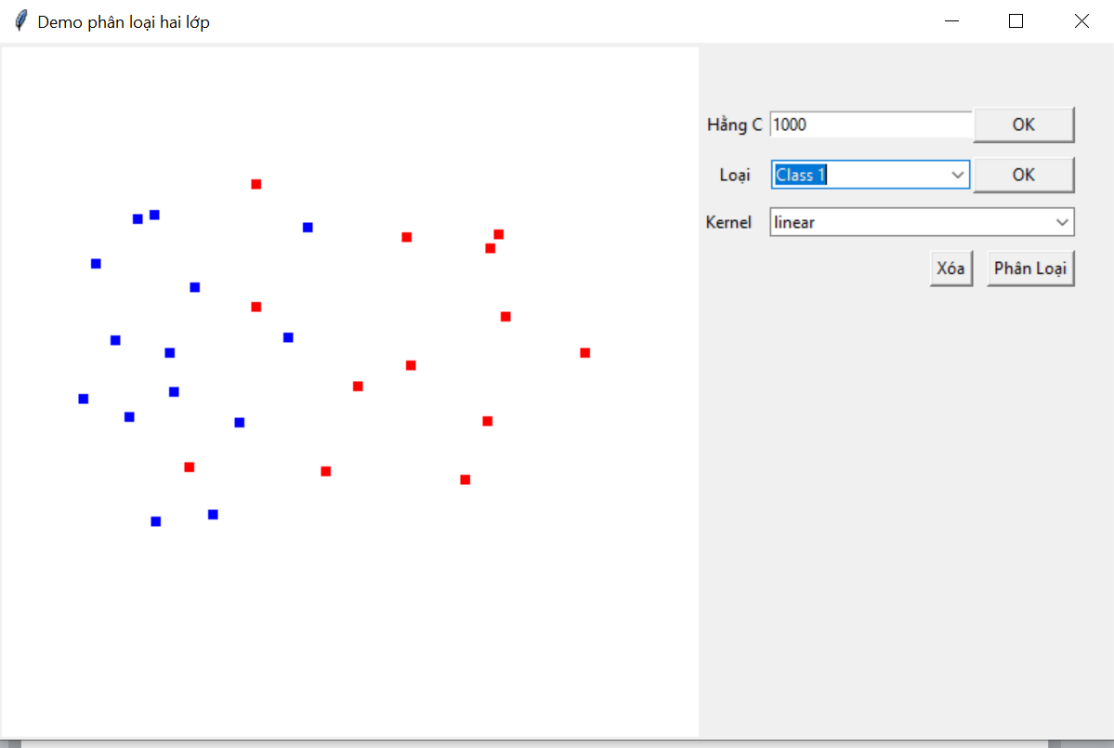
##### *Hình 4.2 Các điểm dữ liệu thuộc hai lớp được tạo trên demo*

Tiến hành phân loại, màn hình matplotlib hiện lên và ta tim được một đường biên :



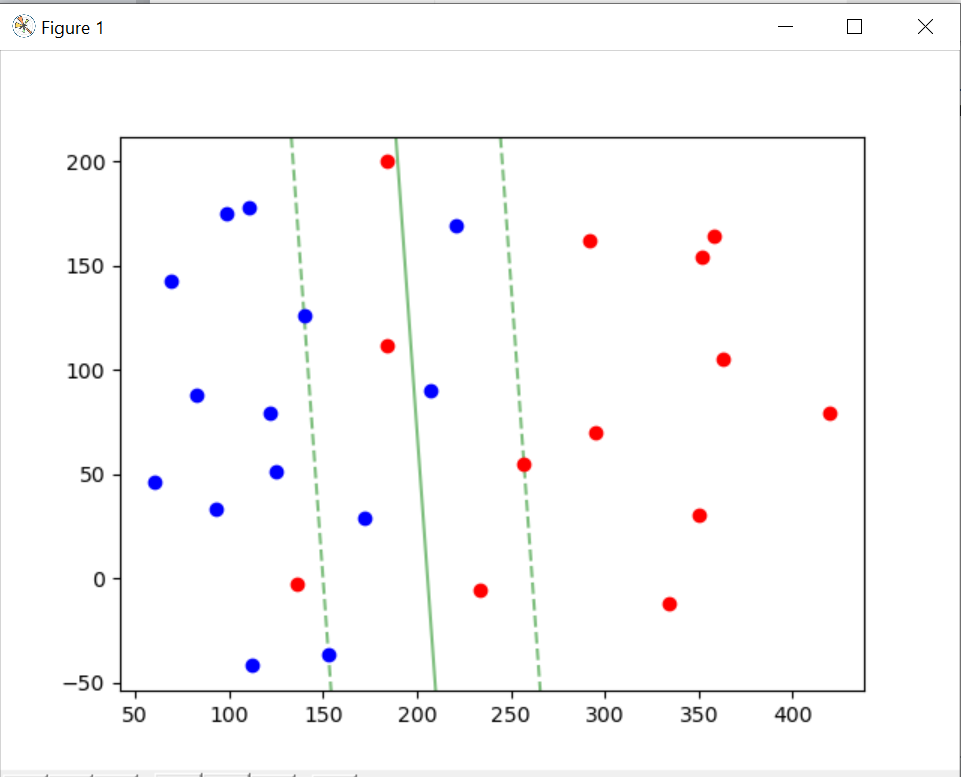
##### *Hình 4.3 Kết quả đường biên được tìm thấy*

Kết quả cho ra hợp lý. Ta tiến hành xét trường hợp tiếp theo:



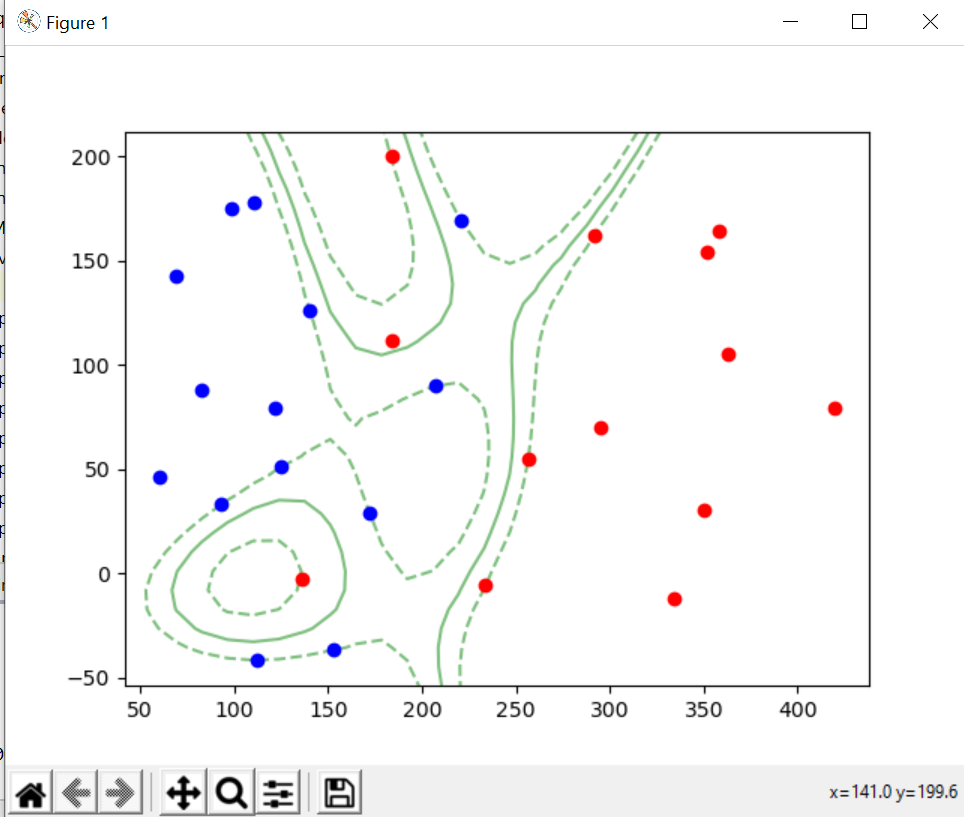
##### *Hình 4.4 Biểu diễn các điểm nhiễu*

Các điểm ở trên tuy nằm phân tán và có nhiều điểm nhiễu . Tiến hành phân loại, ta thấy chương trình bị chậm lại so với tập dữ liệu đã được phân biệt tuyến tính. Đợi một lúc ta được kết quả thể hiện trên matplotlib:



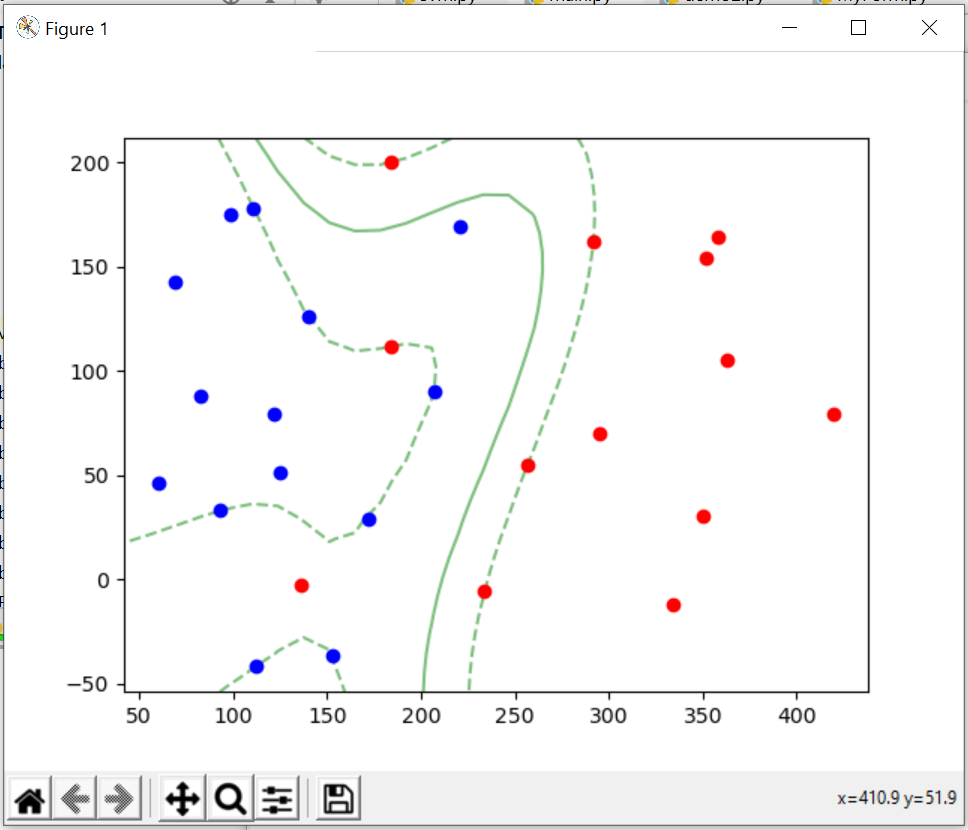
##### *Hình 4.5 Kết quả đường biên với các điểm nhiễu*

Kết quả chấp nhận được. Giờ ta thử đổi hàm kernel sang rbf.



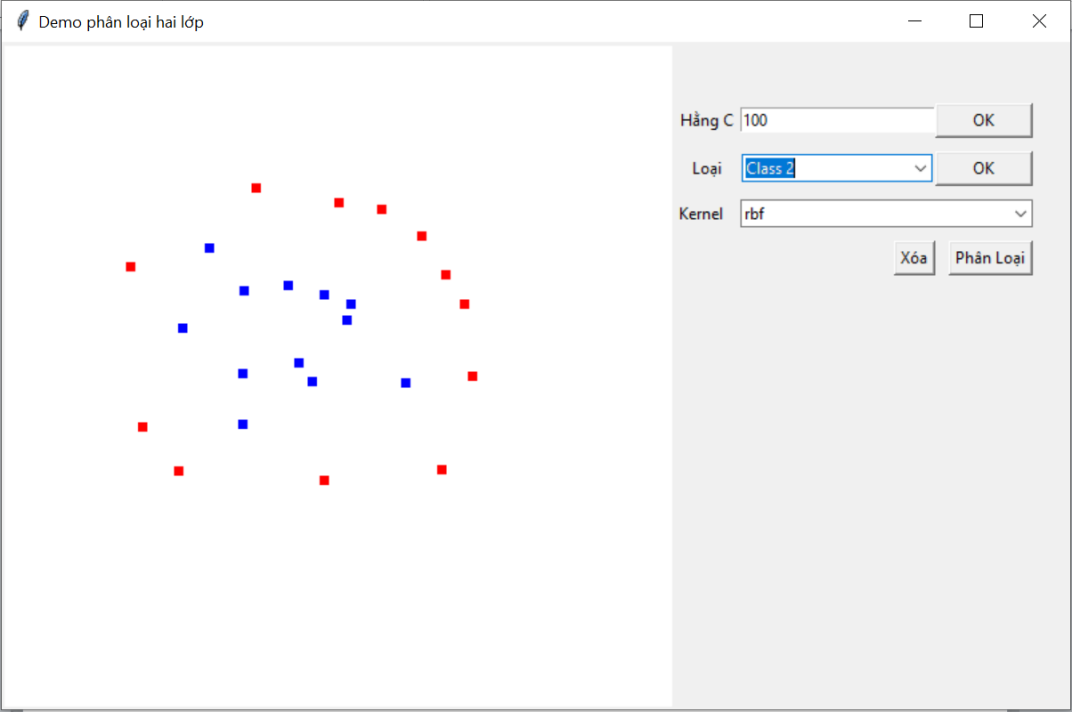
##### *Hình 4.6 Đường biên có điểm nhiễu với kernel là rbf*

Tốc độ nhanh hơn linear nhưng kết quả không ổn lắm. Có thể sẽ xảy ra tình trạng overfitting như em đã từng đề cập. Giải quyết vấn đề này ta thử sét lại hằng số C còn 100. Kết quả khác và ổn hơn nhiều.

**

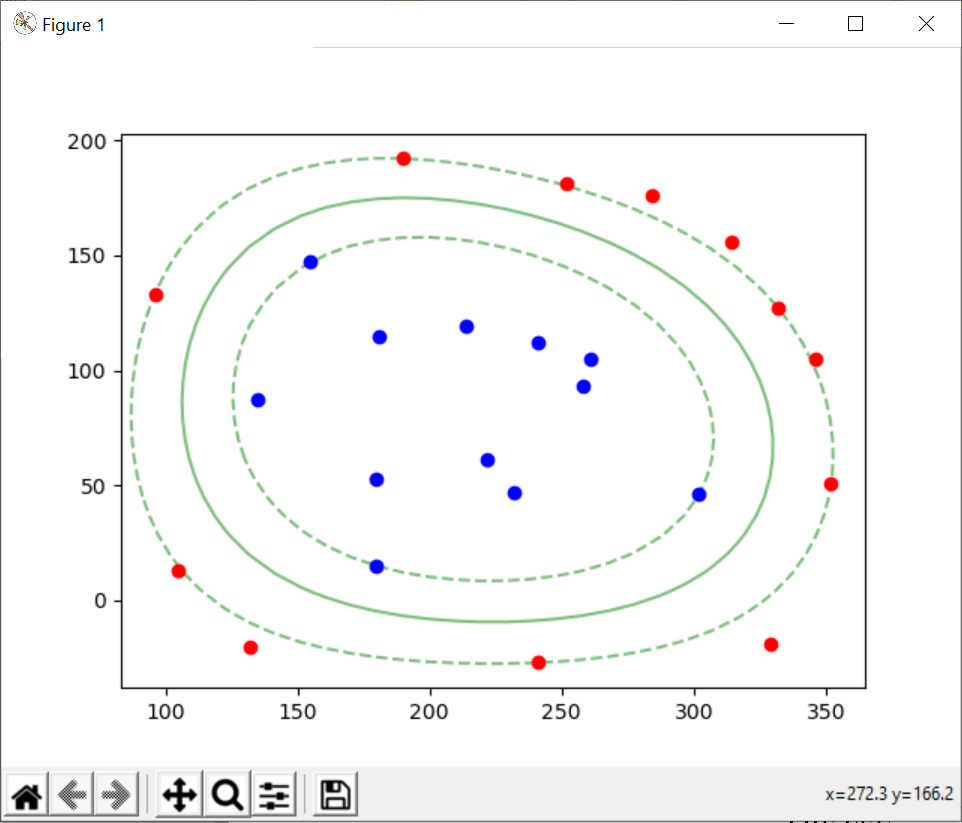
##### *Hình 4.7 Đường biên có nhiễu bằng rbf, đã được thay đổi hằng số C*

Giờ ta thử với tập dữ liệu không phân biệt tuyến tính. Lưu ý điều này buộc ta phải dung hàm kernel không phải là linear:



##### *Hình 4.8 Dữ liệu không phân biệt tuyến tính*

Kết quả thu được rất tốt với đường biên là một hình giống hình elipse:

**

##### *Hình 4.9 Kết quả đường biên với dữ liệu không phân biệt tuyến tính*

Sau demo em có một số nhận xét sau đây:

- Với các tập dữ liệu đã phân tach cụ thể, demo chạy rất tốt cho kết quả ngay và thời gian chạy nhanh.

- Với tập dữ liệu số lượng các điểm càng lớn , chương trình load tốn nhiều thời hian hơn.

- Chương trình làm việc hiệu quả trên các dữ liệu bị nhiễu. Cho ra kết quả khá tốt. Tuy nhiên, thời gian chờ lâu.

- Khi chạy trên kernel là rbf tốt độ chạy nhanh hơn nhiều trên linear. Điều chỉnh hằng số C để có kết quẩ mong muốn.

- Ở kernel là linear, hằng số C dường như không có tác đọng hoặc tác động khá ít.

Demo được đinh kèm ở github, và ở file 5851071042\_NguyenPhucHoaiLinh\_Demo\_1.zip .Lưu ý chạy file main để chạy demo\_1

# CHƯƠNG 4. ỨNG DỤNG SVM VÀO BÀI TOÁN NHẬN DIỆN CHỮ VIẾT TAY

### Bài toán nhận diện chữ số viết tay

Trong lĩnh vực thị giác máy tính, bài toán nhận diện đang ngày càng được chú ý. Nó được ứng dung nhiều trong đời sống chẳng hạn tag tự nhận dạng tài khoản của Facebook, tìm kiểm bằng hình ảnh, nhận dạng khuôn mặt,…

Trong bài báo cáo này, em xin được phép trình bày về bài toán nhận diện chữ số viết tay, là một công đoạn cực kì quan trọng trong các bài toán lớn như ứng dụng giải toán qua hình ảnh, ứng dụng dịch văn bản bằng hình ảnh,…

Trước khi đi chi tiết, em sẽ giới thiệu sơ qua về các bước mà một bài toán nhận diện sẽ tiến hành:

-Xử lý hình ảnh: bước này tiến hành các công đoạn như scale ảnh, cân bằng Histogam của ảnh để ảnh được sát nét hơn, chuyển ảnh về ảnh xám, ảnh nhị phân ,… Nhìn chung đây là bước xử lý sao cho hình ảnh có chất lượng tốt và phù hợp với đề bài

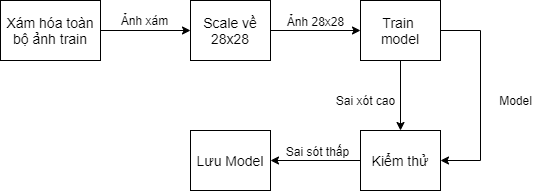
-Phát hiện đối tượng: bước này sẽ xác định được một đối tượng trong một bức ảnh lớn. Tiến hành crop ảnh, scale …. Như bước 1 để có một bức ảnh mới

-Nhận diện: từ ảnh mới, tiến hành huấn luyện model như yêu cầu đề bài.

-Kiểm thư: kiểm tra model huấn luyện có cho đúng với kết quả không. Lưu ý quá trình kiểm thử và huấn luyện cần tách biệt để tránh tình trạng overfitting.

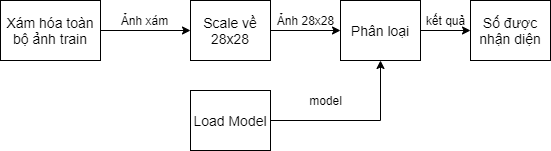
Tuy nhiên trong bài toán, nhận diện chữ số viết tay của em. Các chữ số sẽ được tiến hành vẽ trên một khu hình và nó sẽ được lưu thành một ảnh kích thước 28x28. Do đó, bước phát hiện coi như có thể bỏ qua.

Như vậy chương trình mà em sẽ làm có thể được miêu tả bằng sơ đồ sau đây. Đầu tiên là quy trình huấn luyện model.



##### *Hình 5.1 Sơ đồ huấn luyện model cho bài toán nhận diện chữ số*

Quy trình nhận diện một chữ viết tay:

****

##### *Hình 5.2 Sơ đồ quy trình nhận dạng một bức ảnh có chữ viết tay*

Ở quá trình xám hóa ảnh, giả sử bức ảnh của chúng ta đang ở hệ màu RGB, gồm 3 kênh: Red, Green và Blue. Chúng ta sẽ tiến hành chuyển bức ảnh sang ảnh xám, khi đó giá trị dòng i cột j của ma trận mxn (m,n lần lượt là chiều dài và chiểu cao của ảnh, đơn vị pixel), sẽ chuyển tữ một mảng 1x3, chứa các giá trị red ,green và blue của một điểm ảnh sang một số duy nhất. Con số này biểu thị cường độ điểm sang, chạy từ 0 (màu đen) đến 255 (màu trắng). Có nhiều công thức để chuyển đổi, nhưng công thức được dùng phổ biến để đưa một điểm D từ ảnh màu sang ảnh xám đó là:

Trong OpenCV, việc chuyển ảnh màu sang ảnh xám được thực hiện bởi hàm “**cv2.cvtColor()** ”.



##### *Hình 5.3 Minh họa về ảnh xám*

Sau khi có ảnh xám, nếu bức ảnh không đúng kích thước 28x28 ta tiến hành scale ảnh.

Bước tiếp theo ta sẽ tiến hành training model. Để có thể train được model chúng ta cần phải một tập dữ liệu training set. Ở đây em sẽ dung Mnist, đây là một tập dữ liệu dataset có sẵn, miễn phí và được rất nhiều người sử dụng. Trong Mnist chúng ta sẽ có 60 ngàn bức ảnh 28x28 để train, và 20 ngàn bức ảnh để test. Tuy nhiên, ở đây em xin phép chỉ dùng 16 ngàn mẫu để train và 4 ngàn mẫu để test nhằm rút ngắn thời gian.

Việc train model chính là việc tìm ra hàm bằng cách giải bài toán tối ưu tìm vector W :

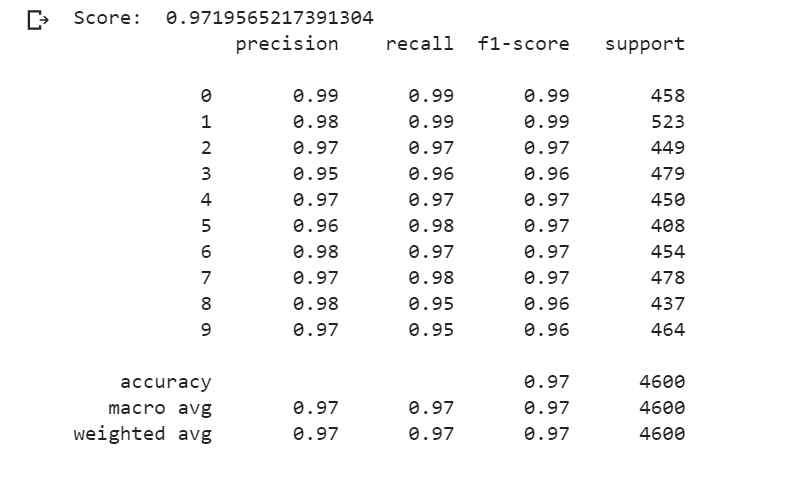
Sunject to: yn(WTX) >= 1 - ξi

Chúng ta có thể giải bài toán này bằng phương pháp đối ngẫu Lagrange**.** Tuy nhiên, Python có một thư viện hỗ trợ tốt trong việc giải quyết bài toán SVM nói riêng cũng như các thuật ML nói riêng đó là Scikit-learn.

Scikit-learn (Sklearn) là thư viện mạnh mẽ nhất dành cho các thuật toán học máy được viết trên ngôn ngữ Python. Thư viện cung cấp một tập các công cụ xử lý các bài toán machine learning và statistical modeling gồm: classification, regression, clustering, và dimensionality reduction. Thư viện được cấp phép bản quyền chuẩn FreeBSD và chạy được trên nhiều nền tảng Linux. Scikit-learn được sử dụng như một tài liệu để học tập.

SVM trong Scikit-learn nằm ở lớp SVC thuộc sklearn.svm(). Để tiến hành tiến hành train model. Ta sẽ dùng hàm svc.fit ( <tập dữ liệu huấn luyện>, <kết quả đầu ra tương ứng>). Trong công đoạn này, việc lấy đặc trưng (thuộc tính, đến đây em xin phéo được dùng từ đặc trưng thay cho thuộc tính cho dễ hiểu) của ảnh để tạo không gian đặc trưng. Tại đây chúng ta sẽ đơn giản dùng các giá trị cường độ xám của ảnh để là vector đặc trưng. Ta có bức ảnh 28x28 pixel vậy, vậy Vector đặc trưng chúng ta sẽ có 28x28=784 đặc trưng.

Chạy thử demo trên colab, ta kiểm thử được kết quả:



##### *Hình 5.4 Kết quả kiểm thử model*

Tuy nhiên, khi ta thử chạy demo với model trên ta được kết quả khá tốt tuy nhiên còn nhiều trường hợp không nhận dạng đúng.

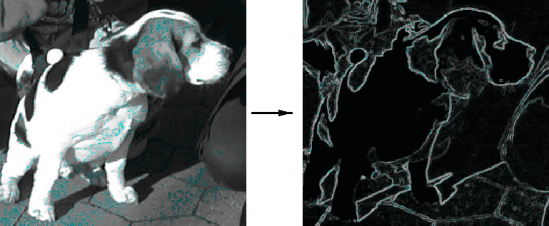


##### *Hình 5.5 Sai sót trong chương trình*

### Trích xuất đặc trưng ảnh bằng thuật toán HOG

Để giải quyết vấn đề trên ta tiến hành tạo ra các đặc trưng cụ thể hơn. Tức là trích xuất một vector đặc trưng từ một ảnh. Có nhiều thuật toán để giúp tạo ra các vector đặc trưng đó. Trong báo cáo này, em xin dùng một thuật toán trong số ấy. Đó là histogram of oriented gradient

Về bản chất các vật thể trong một bức ảnh có thể nhận biết được thong qua các biên của nó (đường nét). Bức ảnh sau đây là một ví du.



##### *Hình 5.6 Minh họa về đường biên trong một bức ảnh*

Ý tưởng của thuật toán HOG đó là từ các đường biên, ta sẽ tiến hành biểu diễn một hình có ma trận I sang hai ma trận. Một biểu diễn độ lớn của đường biên tại một điểm. Một biểu diễn hướng (góc) tại điểm đó. Từ hai ma trận ta có thể lấy được đặc trung của bức ảnh.

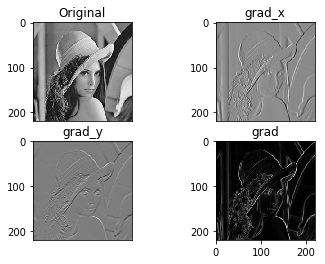
HOG có thể được chia thành các bước sau:

* + Tính toán gradient theo chiều x và y
  + Tính vector đặc trưng cho từng cell
  + Chuẩn hóa các block
  + Tính vector riêng

#### Tính Gradient

Đầu tiên ta cần hiểu Gradient là gì. Hiểu đơn giản Gradient là độ biến thiên về màu trong ảnh, hay còn gọi là đường nét. Để hiểu rõ hơn về Gradient ta sẽ giản sử một ảnh sẽ là một hàm f(x,y) với xi,yi là tọa độ của một pixel. Khi đó f(xi,yi) sẽ là độ sáng của ảnh, giá trị từ 0-255 (Lưu ý ta đang xét trên ảnh xám).Ta có thể thấy nó khá giống đạo hàm của một hàm số tại một điểm.

Giả sử f ’(x,y) là hàm tính gradient tại một điểm thì f ‘(xi,yi) = (fx ‘(xi,yi), fy ‘(xi,yi) ). Trong đó, fx ‘(xi,yi) sẽ tính gradient tại điểm (xi,yi) theo chiều ngang, còn fy ‘(xi,yi) là theo chiều dọc. Cụ thể:



##### *Hình 5.7 Chi tiết bức ảnh với Gradient*

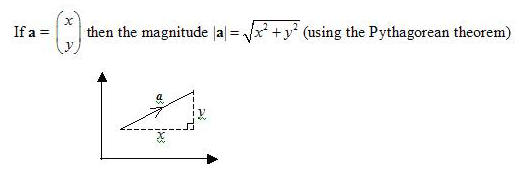
Bốn bức ảnh bên trên, ảnh đầu tiên là ảnh gốc. Ảnh bên phải là ảnh lấy gradient theo chiều ngang. Ảnh góc dưới trái là ảnh lấy gradient theo chiều dọc, còn ảnh cuối chính là gradient tổng.

Để tính được Gradient như các hình trên, ta sẽ dùng một ma trận nhỏ nhân lần lượt vào ma trận ảnh. Ma trận nhỏ này được gọi là bộ lọc, mặt nạ hay hàm nhần kernel. Có nhiều bộ lọc để làm việc này, trong đó em xem dùng bộ lọc Sobel:

Để lọc theo chiều dọc, ta tiến hành tạo một ma trận chuyển vị từ ma trận trên:

Vậy giả sử ta có ma trận I tượng trưng cho bức ảnh, để tính Gradient theo chiều nào ta sẽ nhân ma trận đó với I

Lúc này ta đã hai ma trận đại diện cho ảnh, kích thước 2 ma trận này bằng kích ảnh. Giả sư tại một điểm pixel (xi,yi), với xi,yi lần lượt là các giá trị tương ứng trên hai mảng vừa tìm được. Biểu diễn điểm này trên không gian hai chiều ta được hình như sau:



Đến đây ta hoàn toàn có thể tính được góc của đường thằng đi qua góc tọa độ và điểm biểu diễn.

Ta cũng có thể tính được độ lớn của vector này:

Trong OpenCV quá trình này đã được tính toán thong qua hàm.:

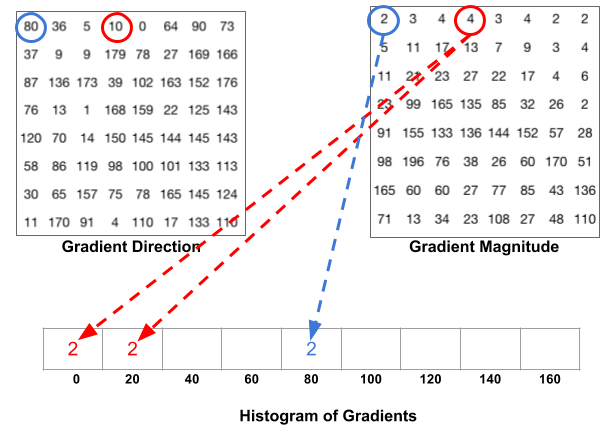
cv2.Sobel()

Đến đây ta lại có được hai ma trận mới, chỉ độ lớn và góc của gradient.

#### Tính Vector đặc trưng cho từng cell

Tại bước này, chúng ta sẽ tiến hành gộp hai ma trận thành những vector. Để làm được điều nàu, ta sẽ tiến hành chia bức ảnh ban đầu thành từng vùng nhỏ gọi là cell. Các cell trong bài của em sẽ có kích thước là 7x7 pixel, vậy với bức ảnh 28x28 px ta sẽ có 4x4 = 16 cell tất cả. Trên mỗi cell đó sẽ bao phủ 48 pixel, mà mỗi pixel sẽ ứng với một độ lớn và một góc nhất định, vậy một cell sẽ có 7x7x2 = 98 giá trị.

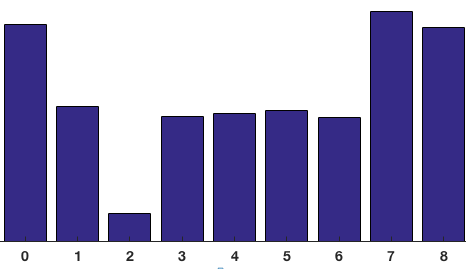
Tiếp theo, ta sẽ tiến hành cộng dồn các giá trị độ lớn theo góc tương ứng. Ta sẽ chia các các đoạn gọi là bin từ 0-180 độ (đã lấy giá trị tuyệt đối các góc âm). Số đoạn được chia thường là 9 hoặc 18. Tương ứng với các đoạn này là các điểm mốc. Sau đó ta tiến hành duyệt hai mảng độ lớn và góc gradient. Mỗi một góc gradient sẽ ghép cặp với một độ lớn gradient ở cùng vị trí tọa độ. Khi biết được góc gradient thuộc bins nào trong véc tơ bins, ta sẽ điền vào giá trị giá trị của độ lớn gradient vào chính bin đó.



##### *Hình 5.8 Cách tạo một vector đặc trưng trên một cell*

Trường hợp góc Gradient không rơi vào đúng các móc bin, ta sẽ tiến hành chia độ lớn theo tỉ lệ của góc. Trong ví dụ trên, ta có vị trí [0,3] có góc là 10 nằm giữa mốc 0 và 20. Như vậy tỷ lệ từ 0 đến 10 và từ 10 đến 20 là do đó độ lớn gradient được chia đều cho hai móc.

Tiếp theo là tính tổng tất cả các bins trong một cell (kích thước 7x7 pixels). Giả sử ta được một vector với độ lớn được biểu diễn như sau:



##### *Hình 5.9 Minh họa vector bin*

#### Chuẩn hóa block

Chúng ta thấy rằng véc tơ histogram sẽ bị phụ thuộc vào cường độ các pixels của một bức ảnh. Với 2 bức ảnh có cùng nội dung nhưng bức ảnh biến thể tối hơn được tạo thành từ ma trận ảnh gốc nhân 1/2. Khi đó giá trị véc tơ histogram của ảnh gốc cũng sẽ gấp đôi véc tơ histogram của ảnh biến thể. Chính vì thế cần chuẩn hóa véc tơ histogram để cả 2 bức ảnh có cùng một véc tơ biểu diễn

Chuẩn hóa norm chuẩn bậc 2: normalize(h)=

Trong đó h là véc tơ histogram của các gradient.

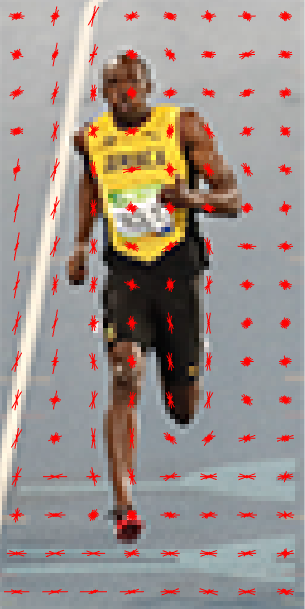
Qúa trình chuẩn hóa sẽ thực hiện trên một block kích thước 2x2 trên lưới ô vuông ban đầu (mỗi ô kích thước 7x7 pixel). Như vậy chúng ta sẽ có 4 véc tơ histogram kích thước 1x9, concatenate các véc tơ sẽ thu được véc tơ histogram tổng hợp kích thước là 1x36 và sau đó chuẩn hóa theo norm chuẩn bậc 2 trên véc tơ này.

#### Tính vector đặc trưng

Sau khi chuẩn hóa các véc tơ histogram, chúng ta sẽ concatenate (nối) các véc tơ 1x36 này thành một véc tơ lớn. Đây chính là véc tơ HOG đại diện cho toàn bộ hình ảnh.

Ví dụ: Hình ảnh của chúng ta được chia thành lưới ô vuông kích thước 4x4 (mỗi ô 7x7). Qúa trình tính toán HOG sẽ di chuyển 4x4 = 36 lần. \. Do đó cuối cùng véc tơ HOG sẽ có kích thước là 36 x 36=1296 chiều.

Chúng ta thử nhìn bức ảnh sau. Hãy để ý, chiều của nhóm các véc tơ sẽ tương đối giống với dáng của vận động viên trong ảnh, đặc biệt là tại các vị trí chân và tay. Cụ thể hãy xem hình bên dưới:



##### *Hình 5.10 HOG biểu diễn trên ảnh*

Trong OpenCV việc tính HOG, bao gồm các công đoạn trên đã được viết gọn dưới một hàm đó là:

*hog = cv2.HOGDescriptor(\_winSize= <kích thước của sổ, thường bằng với kích thước ảnh>,*

*\_blockSize=<kích thước block tính bằng pixel>,*

*\_blockStride= <kích thước khung trượt thườngng bằng với cell>,*

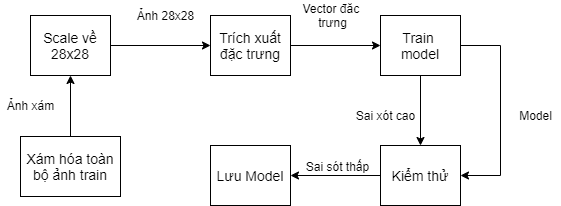
*\_cellSize= <kích thước mỗi cell tính bằng pixel>,*

*\_nbins= <số lượng bin>);*

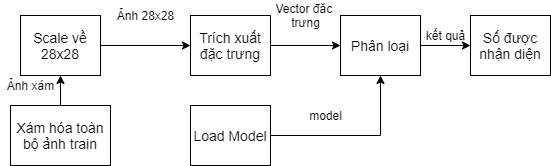
*hog.compute(<ảnh cần trích xuất đặc trưng>,None,None)*

#### Xây dựng lại demo

Như vậy ta nhận thấy việc áp dụng HOG trong bài toán nhận diện chữ só viết tay nằm ở hai giai đoạn tiền xử lý của 2 quá trình huấn luyện và nhận dạng.

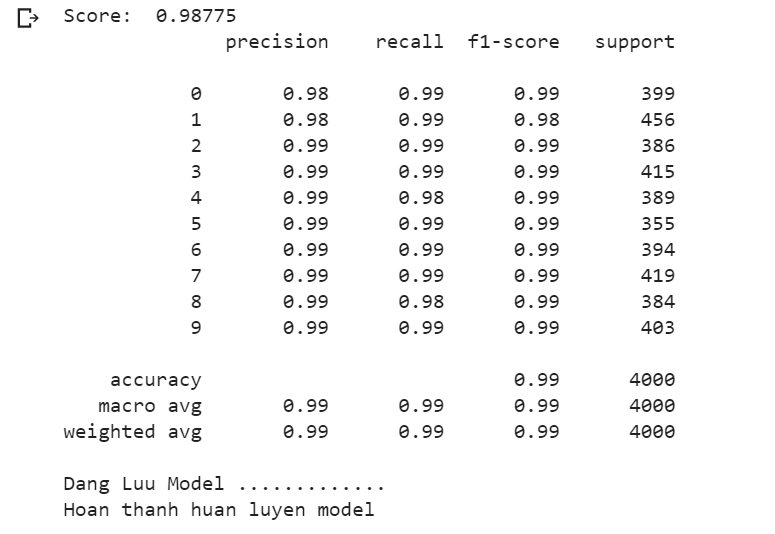


##### *Hình 5.11 Sơ đồ huấn luyện model đã thêm bước trích xuất đặc trưng*



##### *Hình 5.12 Sơ đồ nhận diện bức ảnh thêm bước trích xuất đặc trưng*

Quay lại demo, tiến hành code trên colab ta được một model có kết quả kiển thử như sau.



##### *Hình 5.13 Kết quả kiểm thử model sau khi chỉnh sửa*

Chi tiết demo xem tại: <https://www.youtube.com/watch?v=8wCtCHlV7Mk>

### Hạn chế demo

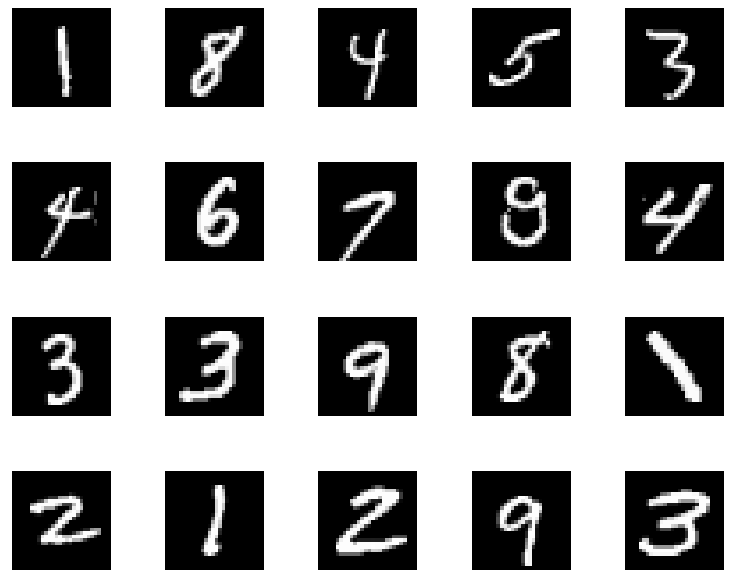
Chương trình tuy hoạt động tốt, cho ra kết quả ổn nhưng gặp phải một số các hạn chế sau:

* + Chương trình chưa phân biệt được kí tự nhập vào là chữ số hay không. Nếu không phải kí tự số thì cần phải thong báo. Còn nếu là kí tự số thì mới tiến hành nhận diện
  + Nếu viết số nhỏ quá sẽ không nhận diện được:
  + Khi kí tư được viết bị lệch, không còn nằm ở vị trí tương đối ở trung tâm khung vẽ. chương trình nhận diện sẽ sai hoàn toán.



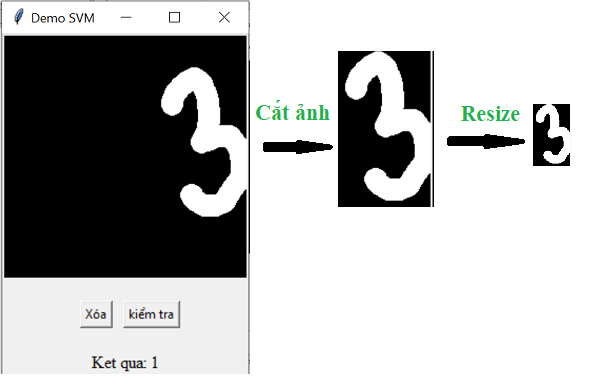
##### *Hình 5.14 Chữ số được vẽ lệch phía phải, nên kết quả không đúng*

Điều này là do model được huấn luyện dựa trên cơ sở dữ liệu MNIST. Đây là tập các ảnh chữ số viết tay, chúng có kích thước là 28x28 pixel. Và đa phần các chữ số này được vẽ ở giữa ảnh. Vì thế khi ảnh input của chương trình không được vẽ vào giữa, nó sẽ không tương đồng với tập huân luyện.



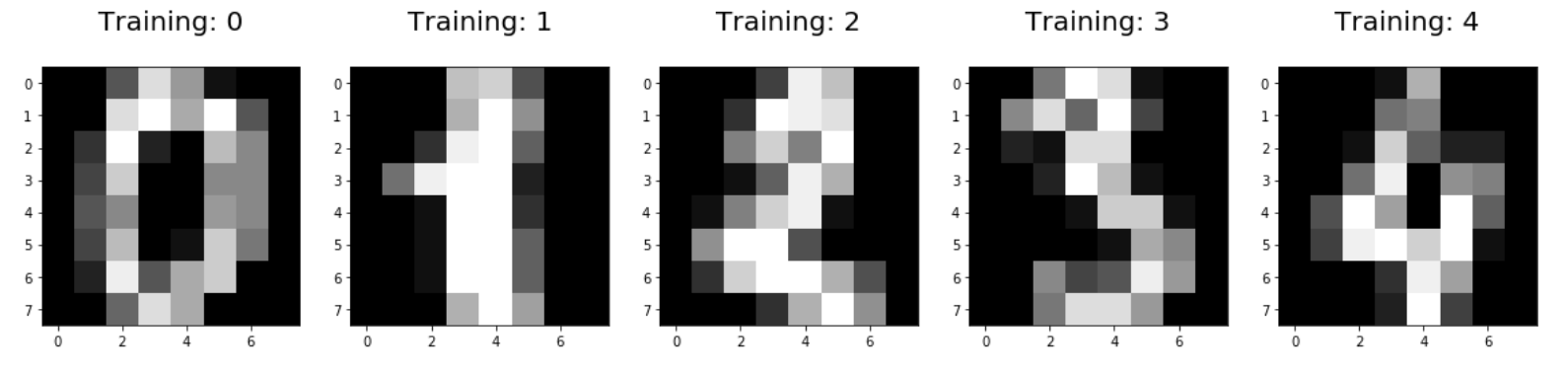
*Hình 5.15 Một vài ảnh của MNIST*

Phướng ánh giải quyết vấn đề này là chúng ta sẽ vẫn giữ lại bước phát hiện vật thể trong bài toán nhận diện. Có nghĩa là chúng ta sẽ tiến hành tìm thành phần trong ảnh có chứa chữ số, rồi cắt vùng đó ra được một ảnh mới. Scale ảnh đó về đúng kích thước yêu cầu rồi tiến hành nhận diện:



##### *Hình 5.16 Thay đổi bước tiền xử lý*

Tất nhiên, khi bước tiền xử lý của ảnh input đã thay đổi thì các ảnh trong tập huấn luyện cũng phải thay đổi giống như vậy để tạo ra các bức ảnh mà chữ viết tay nằm bao toàn bộ ảnh. Tuy nhiên chúng ta hoàn toán có thể dùng một tập dữ liệu mới thay thế cho MNIST đáp ứng được nhu cầu trên.



##### *Hình 5.17 Ví dụ về tập dữ liệu huấn luyện mới sau khi xử lý*

# TỔNG KẾT

Như vậy, trong em đã hoàn thành việc nghiêng cứu thuật toán SVM cũng như ứng dụng nó vào trong bài toán nhận diện chữ số viết tay.Tất nhiên, bài báo cáo cũng như demo sẽ gặp không ít các lỗi và thiếu sót. Chẳng hạn như báo cáo chưa trình bày được các chiến lược phân tách đa lớp thay vì hay lớp, chưa giải quyết vấn đề nhận biết chữ số và không phải chữ số…. Điều này cần đuộc khác phục

Riêng về chương trình demo. Tuy đã nhận thấy được thiếu sót, nhưng em vẫn chưa cụ thể hóa được cách khắc phục chúng vào trong chương trình. Do đó, tương lai chương trình cần phải hạn chế các lỗi em đã đề cập. Đồng thời cần nâng cấp chương trình. Vì công việc nhận diện chữ viết tay chi là một công việc nhỏ trong một bài toán lớn hơn. Chăng hặn, với bài toán xây dựng ứng dụng giải toán qua hình ảnh. Chương trình không chỉ phải nhận diện một kí tự viết tay. Mà là nhiều chữ số viết tay trong một bức ảnh. Ngoài ra còn phải nhận diện các kí tự toán học, các biến, phương trình,… Bài toán giải toán qua hình ảnh cũng chính là hướng mà em muốn chương trình mình đi theo.

**TÀI LIỆU THAM KHẢO**

1. <https://machinelearningcoban.com/>
2. <https://towardsdatascience.com/support-vector-machine-mnist-digit-classification-with-python-including-my-hand-written-digits-83d6eca7004a\>
3. <https://machinelearningcoban.com/>
4. <https://vi.wikipedia.org/wiki/T%E1%BB%91i_%C6%B0u_h%C3%B3a_(to%C3%A1n_h%E1%BB%8Dc)>
5. <https://codelearn.io/sharing/scikit-learn-trong-python-la-gi>
6. <https://phamdinhkhanh.github.io/2019/11/22/HOG.html>
7. <https://minhng.info/tutorials/histograms-of-oriented-gradients.html>
8. <https://tgmtcoban.wordpress.com/2019/02/17/gradient-cua-mot-anh-phan-1/>
9. <https://scikit-learn.org/stable/modules/svm.html>
10. <https://www.youtube.com/watch?v=PNp1prcWbkM&list=PLDpRz2wA0qZzTcDLeXP5PSCfmQ96l9-Qr>