BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC GIA ĐỊNH
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO TỐT NGHIỆP

TÊN ĐỀ TÀI
XÂY DỰNG HỆ THỐNG NHẬN DẠNG CHỮ SỐ VIẾT TAY SỬ DỤNG BỘ DỮ LIỆU MNIST

Cán bộ hướng dẫn: TRẦN HOÀI THUẬN
Sinh viên thực hiện: TRỊNH NGỌC MINH
MSSV: 22150318 Lớp: 221521 Khóa: K16

Thành phố Hồ Chí Minh, tháng 7 năm 2025

LỜI CẢM ƠN

Trong suốt quá trình thực hiện bài báo cáo tốt nghiệp với đề tài "Xây dựng hệ thống nhận dạng chữ số viết tay sử dụng bộ dữ liệu MNIST", em đã nhận được sự hướng dẫn tận tình và hỗ trợ quý báu từ các thầy cô, nhà trường và nhiều cá nhân có liên quan.

Trước hết, em xin được bày tỏ lòng biết ơn sâu sắc đến thầy Trần Hoài Thuận – giảng viên hướng dẫn của em. Thầy không chỉ truyền đạt những kiến thức chuyên môn giá trị về trí tuệ nhân tạo và học máy, mà còn tận tình định hướng, giúp em từng bước tháo gỡ những khó khăn trong quá trình triển khai hệ thống và hoàn thiện bài báo cáo. Những lời khuyên quý báu và sự đồng hành của thầy là nguồn động lực to lớn giúp em vững tin theo đuổi đề tài.

Em cũng xin chân thành cảm ơn các thầy cô trong Khoa Công nghệ Thông tin – Trường Đại học Gia Định đã tận tình giảng dạy và trang bị cho em nền tảng kiến thức vững chắc trong suốt thời gian học tập tại trường. Chính những kiến thức về lập trình, xử lý dữ liệu, thuật toán và các môn học nền tảng đã giúp em có đủ khả năng và tự tin để nghiên cứu, ứng dụng vào đồ án thực tế.

Đồng thời, em xin cảm ơn Trường Đại học Gia Định đã tạo điều kiện thuận lợi để em có cơ hội tiếp cận môi trường học thuật năng động, cùng với việc định hướng thực tập và hỗ trợ trong quá trình hoàn thiện bài báo cáo tốt nghiệp.

Dù đã nỗ lực để hoàn thiện bài báo cáo một cách chỉn chu và nghiêm túc, song với giới hạn về kinh nghiệm và thời gian, chắc chắn không tránh khỏi những thiếu sót. Em rất mong nhận được những góp ý, phản hồi quý báu từ quý thầy cô và hội đồng để có thể tiếp tục rèn luyện, cải thiện và hoàn thiện bản thân tốt hơn trong chặng đường tiếp theo.

NHẬN XÉT CỦA CÁN BỘ HƯỚNG DẪN

(Giảng viên chấm 1)

1. Họ và tên sinh viên:

2. Mã số sinh viên:

3. Lớp học phần:

4. Đánh giá chung:

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

....ngày…….tháng…….năm……..

Cán bộ hướng dẫn

(Ký tên, ghi rõ họ tên)

NHẬN XÉT CỦA CÁN BỘ HƯỚNG DẪN

(Giảng viên chấm 2)

1. Họ và tên sinh viên:

2. Mã số sinh viên:

3. Lớp học phần:

4. Đánh giá chung:

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

....ngày…….tháng…….năm……..

Cán bộ hướng dẫn

(Ký tên, ghi rõ họ tên)

MỤC LỤC

MỤC LỤC HÌNH ẢNH

LỜI MỞ ĐẦU

Trong thời đại công nghệ số phát triển mạnh mẽ như hiện nay, việc tự động hóa các quy trình xử lý thông tin đang trở thành một xu hướng tất yếu trong nhiều lĩnh vực như tài chính – ngân hàng, giáo dục, y tế, hành chính công,… Trong số đó, các bài toán liên quan đến nhận dạng và xử lý dữ liệu hình ảnh, đặc biệt là chữ viết tay, ngày càng thu hút sự quan tâm từ cộng đồng nghiên cứu và ứng dụng thực tiễn.

Nhận dạng chữ số viết tay là một bài toán kinh điển trong lĩnh vực Trí tuệ nhân tạo (AI) và Học máy (Machine Learning), được ứng dụng rộng rãi trong các hệ thống như: chấm điểm tự động, phân loại biểu mẫu, xử lý dữ liệu đầu vào trong ngân hàng, hay nhận dạng mã bưu chính. Tuy nhiên, bài toán này không hề đơn giản do sự đa dạng trong cách viết của mỗi người, các yếu tố gây nhiễu, và độ phân giải hình ảnh khác nhau. Do đó, việc xây dựng một hệ thống có thể nhận dạng chính xác và nhanh chóng các chữ số viết tay là một thách thức thực tiễn mang tính ứng dụng cao.

Xuất phát từ thực tế đó, em đã lựa chọn đề tài "Xây dựng hệ thống nhận dạng chữ số viết tay sử dụng bộ dữ liệu MNIST" làm nội dung chính cho báo cáo tốt nghiệp của mình. Đề tài tập trung nghiên cứu và ứng dụng thuật toán Support Vector Machine (SVM) – một trong những phương pháp học máy phổ biến – để giải quyết bài toán phân loại chữ số viết tay. Đồng thời, hệ thống còn được tích hợp với giao diện web, cho phép người dùng tương tác thông qua việc vẽ chữ số hoặc tải ảnh lên và nhận phản hồi trực tiếp từ mô hình.

Đề tài không chỉ giúp em củng cố kiến thức lý thuyết về học máy và xử lý ảnh, mà còn tạo cơ hội để rèn luyện kỹ năng lập trình, thiết kế hệ thống và triển khai ứng dụng thực tế. Em hy vọng rằng những kết quả đạt được trong quá trình thực hiện đề tài sẽ góp phần nhỏ vào việc ứng dụng công nghệ trí tuệ nhân tạo trong đời sống.

CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI

1.1.Giới thiệu bài toán nhận dạng chữ số viết tay

Nhận dạng chữ số viết tay (Handwritten Digit Recognition) là một lĩnh vực đã được quan tâm từ những năm 1980. Theo từ điển Collins, digit (chữ số) là ký hiệu viết tay thể hiện các con số từ 0 đến 9. Chữ số đóng vài trò thiết yếu trong đời sống hàng ngày và có mặt trong hầu hết các lĩnh vực như ngân hàng, y tế, bảo hiểm, bưu chính,… Các ngành công nghiệp này phụ thuộc rất nhiều vào dữ liệu dạng số.

Trong lĩnh vực ngân hàng, từ việc mở tài khoản đến giao dịch rút tiền, người dùng đều phải điến các biểu mẫu bằng cách viết tay các thông tin như số tài khoản, số định danh hoặc số điện thoại. Những chữ số này sau đó được nhân viên nhập thủ công và hệ thống hoặc quét và nhận dạng bằng máy tính. Tương tự, trong ngành y tế, các thông tin như số hồ sơ bệnh nhân, ghi chú của bác sĩ và liều lượng thuốc cũng yêu cầu phải hiểu đúng các chữ số. Ngay cả trong biểu mẫu thuế, phần lớn dữ liệu được nhập là chữ số. Ngoài ra, công nghệ nhận dạng chữ viết tay còn được ứng dụng trong máy tính bảng, hệ thống bưu chính, và các hệ thống thông minh khác.

Hệ thống nhận dạng chữ số viết tay ngày càng trở thành đề tài được quân tâm trong lĩnh vực công nghệ, đặc biệt trong các nghiên cứu về Trí tuệ nhân tạo (AI) và Máy học (Machine Learning). Mục tiêu của hệ thống này là chuyển đổi các chữ số viết tay của con người thành dạng số hóa mà máy tính có thể hiểu và xử lý. Việc đạt được độ chính xác cao là điều đặc biệt quan trọng, bởi vì sai sót dù chỉ là một chữ số cũng có thể dẫn đến những hậu quả nghiêm trọng trong thực tế. Ví dụ, nét viết tay không rõ ràng có thể khiến chữ số “0” bị nhầm với “8” hoặc “9”, gây ra sự hiểu sai nghiêm trọng trong hệ thống.

Tuy nhiên, việc nhận dạng chữ số viết tay không phải là một bài toán dễ dàng. Phong cách viết tay khác nhau của mỗi người, chất lượng ảnh kém, và sự khác biệt giữa các ký hiệu chữ số trong nhiều ngôn ngữ khác nhau là những thách thức lớn. Do đó, việc lựa chọn đúng thuận toán học máy và huấn luyện mô hình hiệu quản là yếu tố then chốt để tăng độ chính xác. Các thuật toán học máy phải được nghiên cứu

và đánh giá kỹ lưỡng để lựa chọn ra phương pháp tốt nhất cho bài toán này.

Mặc dù đã có nhiều nghiên cứu được thực hiện, nhưng việc đạt được độ chính xác tuyệt đối gần như là không thể. Chỉ cần sai lệch 1% cũng có thể dẫn đến những kết quả sai lệch nghiêm trọng. Vì vậy, trong khuôn khổ đồ án này, nhóm thực hiện tập trung, phân tích và đánh giá các phương pháp hiện có để xây dựng hệ thống nhận dạng chữ số viết tay sử dụng bộ dữ liệu MNIST, với mục tiêu tìm ra giải pháp có độ chính xác và độ tin cậy cao nhất.

Chính vì những khó khăn và yêu cầu cao trong nhận dạng chữ số viết tay, việc phát triển các hệ thông nhận dạng chính xác, đáng tin cậy là một hướng nghiên cứu tiềm năng. Phần tiếp theo sẽ trình bày cụ thể hơn về vai trò và các ứng dụng thực tiễn nỗi bật của công nghệ này trong đời sống và các ngành công nghiệp hiện đại.

1.2.Vai trò và ứng dụng của nhận dạng chữ số trong thực tế

Nhận dạng chữ số viết tay không chỉ là một bài toán kỹ thuật trong lĩnh vực trí tuệ nhân tạo mà còn là nền tảng cho nhiều ứng dụng tự động hóa trong thực tế hiện đại. Khả năng chuyển đổi dữ liệu dạng viết tay sang định dạng số một cách chính xác và nhanh chóng góp phần tối ưu hóa các quy trình xử lý thông tin và cải thiện hiệu suất hoạt động ở nhiều lĩnh vực.

Ứng dụng của công nghệ nhận dạng chữ số viết tay được mở rộng trong các hệ thống xử lý dữ liệu lớn, nơi hàng ngàn biểu mẫu chứa thông tin dạng số được thu thập hàng ngày. Việc tự động hóa bước đọc và nhập dữ liệu giúp giảm thiểu sai sót do con người, tiết kiệm chi phí vận hành và tăng tính bảo mật dữ liệu.

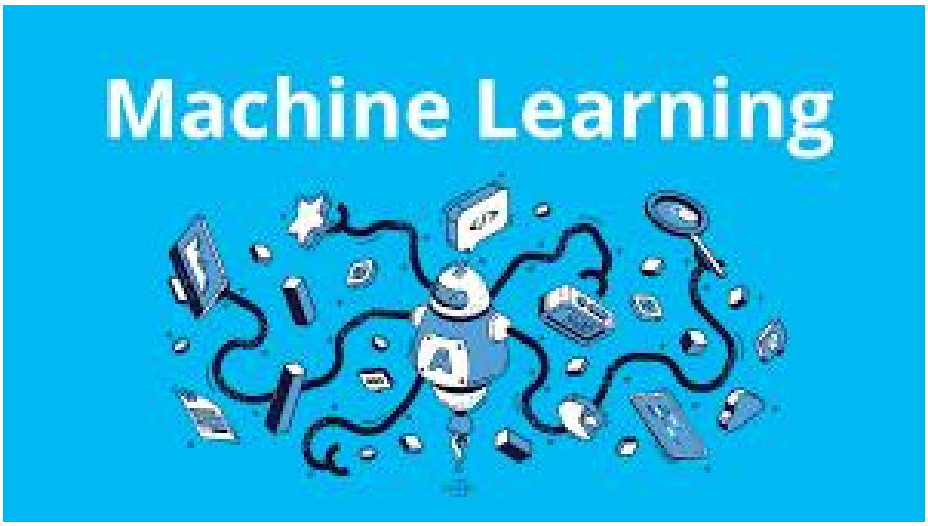
Ngoài các lĩnh vực phổ biến như ngân hàng, y tế, bảo hiểm hay bưu chính, nhận dạng chữ số còn là thành phần quan trọng tron hệ thống giáo dục (như phần mềm chấm bìa kiểm tra viết tay), các thiết bị hỗ trợ học tập số (table, bút cảm ứng), và trong các mô hình học tập tích hợp thực tế tăng cường (AR) và thực tế ảo (VR).

Với sự phát triển của các thiết bị đầu cuối thông minh và nhu cầu số hóa mạnh mẽ từ các tổ chức, doanh nghiệp, công nghệ nhận dạng chữ số viết tay ngày càng đóng vai trò quan trọng trong việc kết nối giữa dữ liệu truyền thống và hệ thống xử

lý thông tin hiện đại.

1.3.Tổng quan về học máy (Machine Learning)

Học máy (Machine Learning – ML) là một lĩnh vực nghiên cứu trọng tâm trong khoa học máy tính và trí tuệ nhân tạo, tập trung vào việc lập trình máy tính để chúng có khả năng học hỏi từ dữ liệu mà không cần được lập trình một cách tường minh cho từng tác vụ cụ thể. Thay vì dựa vào các quy tắc "nếu-thì" được mã hóa thủ công, học máy cho phép hệ thống tự động phát hiện các mẫu và mối quan hệ ẩn trong dữ liệu, từ đó cải thiện hiệu suất của mình trên một nhiệm vụ nhất định thông qua kinh nghiệm. Điều quan trọng là quá trình này không chỉ đơn thuần là việc thu thập dữ liệu; ví dụ, việc tải xuống toàn bộ Wikipedia không được coi là học máy nếu nó không giúp máy tính cải thiện khả năng thực hiện bất kỳ tác vụ nào.



Hình 1.1: Học máy (Machine Learning)

- **Lợi ích và ứng dụng của học máy:**

Giải quyết các vấn đề phức tạp: Học máy đặc biệt hiệu quả với những bài toán mà giải pháp hiện có đòi hỏi lượng lớn tinh chỉnh hoặc danh sách quy tắc dài, như việc phát triển bộ lọc thư rác có khả năng tự động nhận diện các mẫu spam. Nó cũng có thể tìm ra giải pháp cho các vấn đề quá phức tạp mà các phương pháp truyền thống không thể xử lý, ví dụ như nhận dạng giọng nói.

Thích nghi với môi trường biến động: Hệ thống học máy có khả năng tự động thích nghi với dữ liệu mới và môi trường thay đổi, giúp chúng duy trì hiệu suất trong

các tình huống thực tế liên tục biến đổi.

Thu thập thông tin chi tiết: Học máy là công cụ đắc lực để khám phá những thông tin sâu sắc từ các vấn đề phức tạp và khối lượng dữ liệu khổng lồ. Các ứng dụng phổ biến của học máy bao gồm nhận dạng ký tự quang học (OCR), hệ thống gợi ý, tìm kiếm bằng giọng nói, chẩn đoán y tế, và các mạng xã hội.

- **Các loại hình học máy cơ bản:**

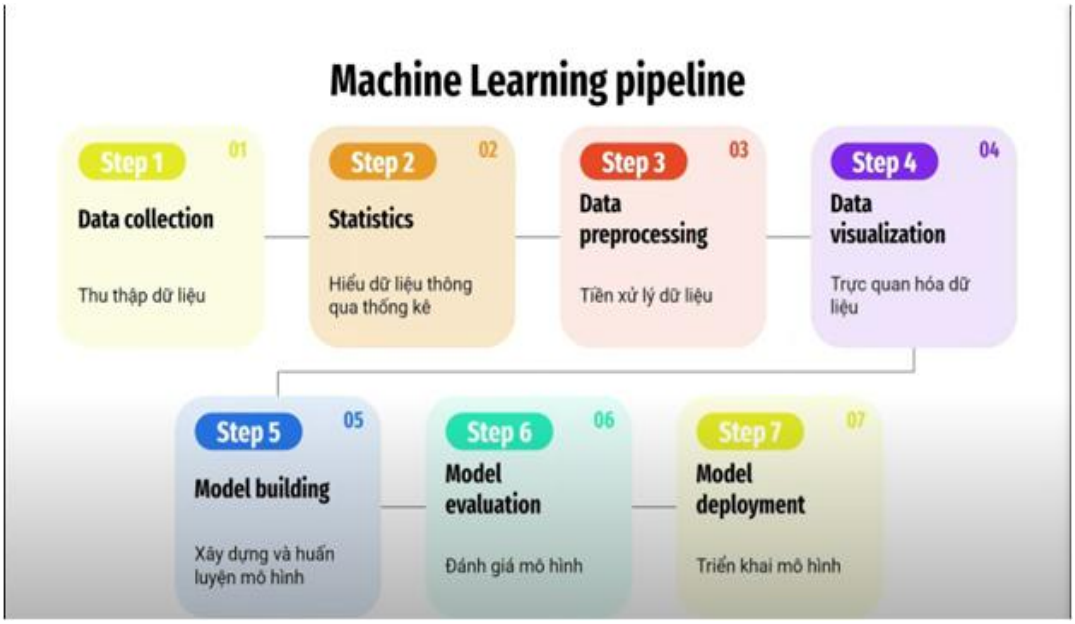
Học có giám sát (Supervised Learning): Mô hình được huấn luyện trên một tập dữ liệu đã được gán nhãn, bao gồm cả đầu vào và đầu ra mong muốn. Mục tiêu là học một ánh xạ từ đầu vào đến đầu ra để có thể dự đoán chính xác cho dữ liệu mới, chưa từng thấy. Các nhiệm vụ chính bao gồm phân loại (Classification), ví dụ như nhận dạng chữ số viết tay từ hình ảnh, và hồi quy (Regression).

Học không giám sát (Unsupervised Learning): Hệ thống được cung cấp dữ liệu không có nhãn và được yêu cầu tự động khám phá cấu trúc, mẫu hoặc mối quan hệ ẩn trong dữ liệu. Các ví dụ bao gồm phân cụm (clustering) và giảm chiều dữ liệu (dimensionality reduction).

Học bán giám sát (Semisupervised Learning): Kết hợp cả dữ liệu có nhãn và không có nhãn trong quá trình huấn luyện để tận dụng tối đa thông tin sẵn có.

Học tăng cường (Reinforcement Learning): Một tác nhân (agent) học cách đưa ra các quyết định thông qua tương tác với môi trường, nhận "phần thưởng" hoặc "hình phạt" để tối đa hóa mục tiêu dài hạn.

- **Quy trình điển hình của một dự án học máy:**



Hình 1.2: Quy trình dự án học máy

Nghiên cứu và thu thập dữ liệu: Đây là bước khởi đầu, tập trung vào việc thu thập dữ liệu liên quan đến bài toán và tổ chức chúng thành một tập huấn luyện.

Khám phá và chuẩn bị dữ liệu: Phân tích dữ liệu để hiểu rõ các mẫu tiềm ẩn và thực hiện các bước tiền xử lý cần thiết, như làm sạch dữ liệu, để chuẩn bị cho thuật toán học máy.

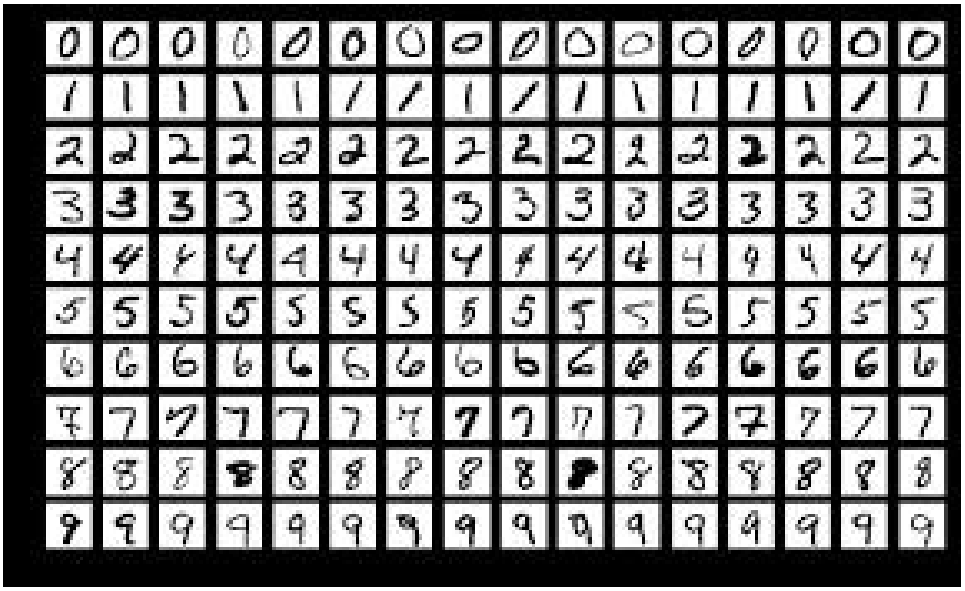
Lựa chọn và huấn luyện mô hình: Chọn một hoặc nhiều thuật toán học máy phù hợp với bản chất của bài toán và tiến hành huấn luyện chúng trên tập dữ liệu đã chuẩn bị. Quá trình huấn luyện này liên quan đến việc thuật toán tìm kiếm các giá trị tham số của mô hình để tối thiểu hóa một hàm mất mát.

Đánh giá mô hình: Sau khi huấn luyện, hiệu suất của mô hình cần được đánh giá khách quan trên dữ liệu mới mà nó chưa từng thấy (tập kiểm tra). Mục tiêu là đảm bảo mô hình có khả năng tổng quát hóa (generalize) tốt, tức là hoạt động hiệu quả không chỉ trên dữ liệu đã học mà còn trên dữ liệu thực tế.

1.4.Giới thiệu bộ dữ liệu MNIST

Bộ dữ liệu MNIST (Modified National Institute of Standards and Technology) là một trong những bộ dữ liệu phổ biến và được nghiên cứu rộng rãi nhất trong lĩnh vực học máy và thị giác máy tính. Nó thường được gọi là “hello world” của học máy,

đóng vai trò là một chuẩn mực quan trọng để đánh giá và so sánh hiệu suất của các thuật toán khác nhau.



Hình 1.3: Bộ dữ liệu MNIST

- **Đặc điểm của bộ dữ liệu:**

MNIST bao gồm tổng cộng 70.000 hình ảnh chữ số viết tay kích thước nhỏ, được thu thập từ học sinh trung học và nhân viên Cục Điều tra Dân số Hoa Kỳ.

Mỗi hình ảnh được gán nhãn (labeled) với chữ số mà nó đại diện, từ 0 đến 9, tạo thành 10 lớp phân loại.

Các hình ảnh có định dạng ảnh xám (grayscale) và có kích thước 28x28 pixel. Điều này có nghĩa là mỗi hình ảnh có thể được biểu diễn dưới dạng một vector 784 chiều (28 * 28 = 784).

Bộ dữ liệu đã được phân chia sẵn thành hai tập hợp:

Tập huấn luyện (training set): Gồm 60.000 hình ảnh.

Tập kiểm tra (test set): Gồm 10.000 hình ảnh.

Tập huấn luyện đã được xáo trộn (shuffled), đảm bảo các fold trong quá trình kiểm định chéo (cross-validation) là đồng nhất và giảm thiểu ảnh hưởng của thứ tự dữ liệu lên một số thuật toán học máy.

- **Vai trò và ứng dụng điển hình:**

Điểm khởi đầu cho người học: MNIST là bộ dữ liệu lý tưởng để những người mới bắt đầu học máy thực hành các kỹ thuật phân loại và hiểu cách các thuật toán

hoạt động.

Chuẩn mực đánh giá thuật toán: Do cấu trúc đơn giản nhưng đủ độ phức tạp, MNIST thường được sử dụng để kiểm tra và so sánh các thuật toán phân loại mới, từ đó thúc đẩy sự phát triển trong nghiên cứu học máy.

Tiền xử lý và giảm chiều dữ liệu: Các hình ảnh MNIST có thể được phân tích để tối ưu hóa, ví dụ, các pixel ở vùng biên thường có giá trị màu trắng và có thể được bỏ qua mà không làm mất nhiều thông tin. Các kỹ thuật giảm chiều dữ liệu như Phân tích thành phần chính (PCA) có thể giảm số lượng đặc trưng của mỗi hình ảnh từ 784 xuống còn khoảng 150, trong khi vẫn giữ được 95% phương sai dữ liệu, giúp tăng tốc độ huấn luyện các thuật toán phân loại. Các phương pháp khác như t-SNE (t-distributed Stochastic Neighbor Embedding) cũng được dùng để trực quan hóa dữ liệu MNIST trong không gian hai chiều, tạo ra các cụm chữ số được tách biệt rõ ràng.

Hiệu suất mô hình: Một mô hình tuyến tính đơn giản có thể đạt độ chính xác khoảng 92% trên MNIST. Các mạng nơ-ron sâu hơn, chẳng hạn như kiến trúc LeNet-5 (một trong những kiến trúc Mạng nơ-ron tích chập - CNN đầu tiên), đã được sử dụng rộng rãi và đạt hiệu suất cao trong nhận dạng chữ số viết tay trên MNIST. Các mô hình phức tạp có thể đạt độ chính xác trên 97% hoặc thậm chí 98%.

1.5.Mục tiêu và phạm vi đề tài

- **Mục tiêu**

Mục tiêu chính của đề tài là xây dựng một hệ thống có khả năng nhận dạng chính xác chữ số viết tay từ ảnh đầu vào thông qua việc ứng dụng các phương pháp học máy, cụ thể là thuật toán Support Vector Machine (SVM). Đề tài hướng đến việc triển khai một quy trình hoàn chỉnh từ tiền xử lý ảnh, huấn luyện mô hình học máy, đánh giá độ chính xác, đến xây dựng giao diện người dùng tương tác trực tiếp với hệ thống.

Cụ thể, đề tài đặt ra các mục tiêu sau:

Tìm hiểu và ứng dụng thuật toán SVM vào bài toán phân loại chữ số viết tay từ

tập dữ liệu chuẩn MNIST.

Tiền xử lý ảnh đầu vào sao cho phù hợp với định dạng dữ liệu mà mô hình SVM yêu cầu.

Huấn luyện và đánh giá mô hình SVM với các chiến lược phân loại đa lớp (One-vs-All).

Xây dựng giao diện web đơn giản cho phép người dùng vẽ chữ số hoặc tải ảnh từ thiết bị, sau đó nhận kết quả dự đoán từ hệ thống.

Hiển thị kết quả dự đoán một cách trực quan, kèm theo thời gian phản hồi, giúp nâng cao trải nghiệm người dùng.

Đề xuất một số hướng phát triển để cải thiện hệ thống trong tương lai.

- **Phạm vi đề tài**

Do giới hạn về thời gian và tài nguyên, phạm vi của đề tài được xác định như sau:

Thuật toán học máy: Đề tài chỉ tập trung triển khai mô hình SVM cơ bản, không bao gồm các mô hình học sâu như CNN.

Dữ liệu: Sử dụng tập dữ liệu MNIST gồm ảnh chữ số viết tay với kích thước 28×28 pixel, ảnh xám, tổng cộng 70.000 ảnh (60.000 huấn luyện, 10.000 kiểm tra).

Phân loại đa lớp: Áp dụng chiến lược One-vs-All để mở rộng SVM cho bài toán nhận dạng 10 chữ số (từ 0 đến 9).

Tiền xử lý ảnh: Thực hiện các bước tiền xử lý cơ bản như chuẩn hóa, chuyển ảnh sang ảnh xám, làm phẳng ảnh và thay đổi kích thước.

Giao diện người dùng: Phát triển giao diện đơn giản bằng HTML, CSS, JavaScript, kết nối với mô hình qua Flask API.

Triển khai mô hình: Mô hình hoạt động ở môi trường cục bộ, chưa triển khai lên nền tảng đám mây hoặc thiết bị di động.

Với phạm vi và mục tiêu như trên, đề tài hướng đến việc kết hợp giữa lý thuyết học máy và ứng dụng thực tế, tạo nền tảng cho việc phát triển các hệ thống nhận dạng thông minh hơn trong tương lai.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT VÀ CÔNG NGHỆ SỬ DỤNG

2.1.Các kỹ thuật trong học máy áp dụng

2.1.1.Phân loại (Classification)

Phân loại (Classification) là một trong những nhiệm vụ phổ biến và quan trọng nhất trong Học máy giám sát (Supervised Learning). Mục tiêu chính của phân loại là dự đoán một nhãn (label) hoặc hạng mục (category) rời rạc cho một mẫu dữ liệu đầu vào. Điều này khác biệt với Hồi quy (Regression), một nhiệm vụ học máy giám sát khác, nơi mục tiêu là dự đoán một giá trị liên tục.

Trong bài toán nhận dạng chữ số viết tay, phân loại đóng vai trò cốt lõi: hệ thống sẽ nhận một hình ảnh của chữ số viết tay (đầu vào) và dự đoán chữ số đó là số nào (ví dụ: 0, 1, 2,... 9).

- **Các khái niệm và đặc điểm chính của phân loại:**

Dữ liệu có nhãn huấn luyện một mô hình phân loại, chúng ta cần một tập dữ liệu huấn luyện (training set) bao gồm các mẫu đầu vào (được gọi là đặc trưng - features hoặc vector mẫu - sample vectors) và các nhãn lớp (class labels) tương ứng. Ví dụ, trong bộ dữ liệu MNIST, mỗi hình ảnh chữ số sẽ có một nhãn tương ứng (ví dụ: hình ảnh của số "5" sẽ có nhãn là "5").

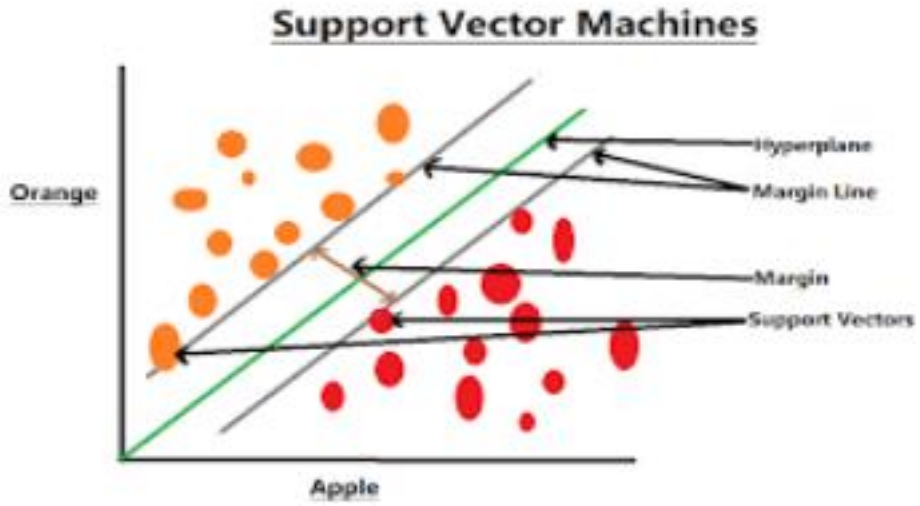
Phân loại nhị phân (Binary Classification): Là nhiệm vụ phân loại dữ liệu thành hai lớp riêng biệt. Ví dụ, xác định xem một hình ảnh có phải là chữ số "5" hay không ("5" hoặc "không phải 5").

Phân loại đa lớp (Multi-class Classification): Là nhiệm vụ phân loại dữ liệu thành nhiều hơn hai lớp. Trong trường hợp nhận dạng chữ số viết tay, đây là bài toán phân loại đa lớp vì có 10 chữ số (0 đến 9) cần được phân biệt.

Scikit-learn là một thư viện Python mạnh mẽ cung cấp nhiều thuật toán học máy, bao gồm cả các lớp phân loại phổ biến như SVC (cho Support Vector Machine), KNeighborsClassifier(k-NearestNeighbors), RandomForestClassifier.

2.1.2.Support Vector Machine (SVM)

Support Vector Machine (SVM) là một tập hợp các phương pháp học máy giám sát (supervised learning methods) được sử dụng cho các nhiệm vụ phân loại (classification), hồi quy (regression) và phát hiện ngoại lệ (outliers detection). Mặc dù có thể dùng cho cả hồi quy và phát hiện ngoại lệ, SVM được sử dụng chủ yếu và nổi bật nhất trong các bài toán phân loại.



Hình 2.1: Mô hình Support Vector Machine

- **Ý tưởng cốt lõi và cách hoạt động:**

SVM hoạt động bằng cách tìm kiếm một siêu phẳng (hyperplane) hoặc một tập hợp các siêu phẳng trong không gian đa chiều để phân chia các lớp dữ liệu một cách hiệu quả. Một cách trực quan, mục tiêu của SVM là tìm một siêu phẳng có khoảng cách lớn nhất đến các điểm dữ liệu huấn luyện gần nhất của mỗi lớp. Khoảng cách này được gọi là margin. Siêu phẳng với margin lớn nhất thường mang lại khả năng tổng quát hóa tốt hơn (generalization error) cho dữ liệu mới, chưa từng thấy.

Các điểm dữ liệu nằm trên biên của "con đường" (street) được tạo bởi siêu phẳng, hay nói cách khác là các điểm gần siêu phẳng phân chia nhất, được gọi là vector hỗ trợ (support vectors). Điều đặc biệt là hàm quyết định của SVM chỉ phụ thuộc vào các support vectors này; các điểm dữ liệu khác nằm "ngoài con đường" không ảnh hưởng đến biên quyết định. Điều này cũng làm cho SVM hiệu quả về bộ nhớ.

- **Các loại SVM và Kernel Trick:**
  - SVM tuyến tính (Linear SVM):

Được sử dụng khi dữ liệu có thể được phân tách tuyến tính, tức là có thể tìm thấy một đường thẳng (trong 2D) hoặc một siêu phẳng (trong không gian cao hơn) để phân chia hoàn hảo các lớp.

Hard Margin SVM: Tìm kiếm một siêu phẳng phân tách hoàn hảo các lớp mà không cho phép bất kỳ lỗi phân loại hay điểm nào nằm trong margin. Phương pháp này nhạy cảm với các nhiễu loạn hoặc ngoại lệ trong dữ liệu.

Soft Margin SVM: Linh hoạt hơn bằng cách cho phép một số lỗi phân loại hoặc các điểm dữ liệu nằm trong margin. Điều này được thực hiện thông qua việc giới thiệu các biến trượt (slack variables, ζ_i) và một tham số điều chuẩn **C**.

Tham số **C** kiểm soát sự cân bằng giữa việc tối đa hóa margin và việc chịu phạt cho các lỗi phân loại: giá trị **C** lớn sẽ phạt nặng hơn cho lỗi, trong khi **C** nhỏ sẽ cho phép nhiều lỗi hơn để có một margin rộng hơn.

  - SVM phi tuyến tính (Nonlinear SVM):

Trong nhiều trường hợp thực tế, dữ liệu không thể phân tách tuyến tính trong không gian ban đầu. SVM giải quyết vấn đề này bằng cách sử dụng Kernel Trick (kỹ thuật hạt nhân). Kỹ thuật này cho phép SVM ánh xạ dữ liệu đầu vào từ không gian ban đầu có chiều thấp lên một không gian đặc trưng có chiều cao hơn (thậm chí vô hạn) mà không cần phải tính toán rõ ràng các tọa độ của dữ liệu trong không gian mới đó. Thay vào đó, nó chỉ tính toán tích vô hướng (dot product) giữa các cặp điểm dữ liệu trong không gian chiều cao đó thông qua một hàm kernel.

  - Các hàm kernel phổ biến bao gồm:

Linear (tuyến tính): $K(x, x') = \langle x, x' \rangle$.

Polynomial (đa thức): $K(x, x') = (\gamma \langle x, x' \rangle + r)^d$, với **d** là bậc của đa thức và **r** là tham số **coef0**.

Radial Basis Function (RBF) hoặc Gaussian Kernel:

$K(x, x') = \exp(-\gamma |x - x'|^2)$, với γ (gamma) là tham số kiểm soát độ

rộng của hạt nhân Gaussian, ảnh hưởng đến "phạm vi" ảnh hưởng của mỗi điểm huấn luyện.

Sigmoid: $K(x, x') = \tanh(\gamma\langle x, x'\rangle + r)$.

- **Đánh giá và các thuộc tính khác:**
  - Ưu điểm:

Hiệu quả trong không gian chiều cao: SVM hoạt động tốt trong các không gian có số chiều lớn, đặc biệt hữu ích cho các bài toán phân loại văn bản và ảnh.

Hiệu quả bộ nhớ: Chỉ sử dụng một tập con của các điểm huấn luyện (support vectors) trong hàm quyết định, giúp tiết kiệm bộ nhớ.

Linh hoạt với các hàm Kernel: Có thể sử dụng các hàm kernel khác nhau để xử lý dữ liệu phi tuyến tính một cách hiệu quả.

Khả năng chống nhiễu/ngoại lệ (với Soft Margin): Chấp nhận một số ngoại lệ và tìm ra siêu phẳng có biên giới tối đa.

  - Nhược điểm:

Đào tạo chậm với bộ dữ liệu lớn: Thời gian tính toán và yêu cầu lưu trữ của SVM tăng nhanh chóng với số lượng vector huấn luyện. Nó hoạt động tốt cho các bộ dữ liệu phức tạp có kích thước nhỏ hoặc trung bình, nhưng có thể rất chậm khi số lượng mẫu dữ liệu lớn (ví dụ, hàng trăm nghìn mẫu trở lên).

Nhạy cảm với việc chia tỷ lệ dữ liệu: Các thuật toán SVM không bất biến với tỷ lệ, do đó rất khuyến nghị cần chuẩn hóa dữ liệu (feature scaling). Cùng một phép chuẩn hóa phải được áp dụng cho cả tập huấn luyện và tập kiểm thử.

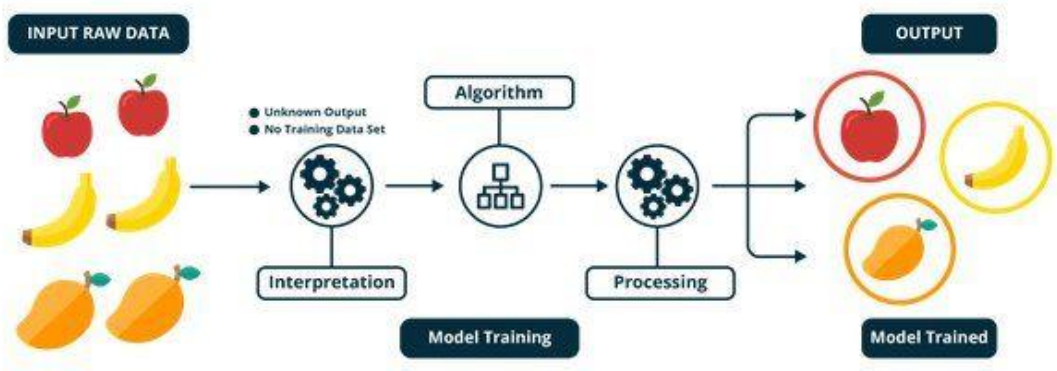
Khó tinh chỉnh tham số: Việc chọn hàm Kernel phù hợp và điều chỉnh các siêu tham số như **C** và γ đòi hỏi sự cẩn thận.

Không cung cấp trực tiếp ước tính xác suất: SVM không trực tiếp cung cấp ước tính xác suất, mà chúng phải được tính toán thông qua các phương pháp tốn kém như Platt scaling (sử dụng cross-validation).

Khó diễn giải: Sự phức tạp của siêu phẳng trong không gian chiều cao làm cho SVM ít có khả năng diễn giải hơn so với một số mô hình khác.

2.2.Tiền xử lý dữ liệu

Tiền xử lý dữ liệu là một bước quan trọng trong quy trình học máy, đặc biệt là đối với dữ liệu ảnh. Mục tiêu chính của tiền xử lý là chuyển đổi dữ liệu thô sang định dạng phù hợp hơn cho các thuật toán học máy, giúp cải thiện hiệu suất, độ ổn định và khả năng tổng quát hóa của mô hình. Trong nhận dạng ảnh, các kỹ thuật tiền xử lý thường bao gồm chuẩn hóa dữ liệu và giảm chiều dữ liệu.



Hình 2.2: Tiền xử lý dữ liệu

Chuẩn hóa dữ liệu (Data Normalization hoặc Feature Scaling) là một kỹ thuật tiền xử lý biến đổi các giá trị của đặc trưng (features) trong tập dữ liệu để chúng nằm trong một phạm vi (scale) tương tự nhau. Đây là một bước cực kỳ quan trọng đối với nhiều thuật toán học máy, đặc biệt là những thuật toán nhạy cảm với thang đo của dữ liệu, như Support Vector Machine (SVM).

- **Lý do cần chuẩn hóa dữ liệu:**

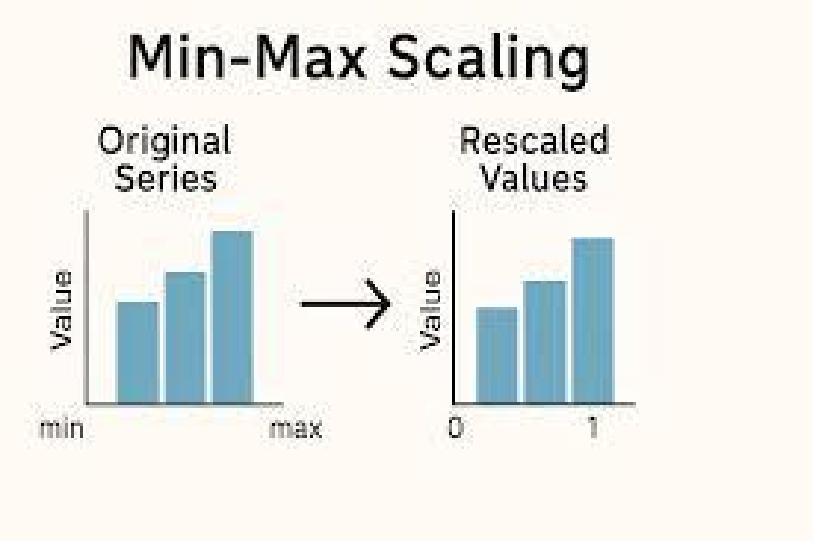
Tránh sự chi phối của các đặc trưng: Các đặc trưng có phạm vi giá trị lớn hơn có thể vô tình chiếm ưu thế trong quá trình tính toán khoảng cách hoặc gradient trong thuật toán. Ví dụ, trong các thuật toán dựa trên khoảng cách (như SVM), một đặc trưng với giá trị từ 0 đến 1000 sẽ có ảnh hưởng lớn hơn rất nhiều so với một đặc trưng có giá trị từ 0 đến 1, ngay cả khi đặc trưng thứ hai thực sự quan trọng hơn về mặt thông tin.

Cải thiện hiệu suất của thuật toán: Các thuật toán SVM không bất biến với tỷ lệ (not invariant to scale), do đó, rất khuyến nghị cần chuẩn hóa dữ liệu (feature scaling). Việc chuẩn hóa đảm bảo rằng tất cả các đặc trưng đóng góp tương đương vào việc xác định siêu phẳng phân chia tối ưu (hyperplane), dẫn đến khả năng học tốt hơn và kết quả phân loại chính xác hơn.

Hỗ trợ quá trình tối ưu hóa: Đối với các thuật toán tối ưu hóa dựa trên gradient, việc có các đặc trưng ở các thang đo khác nhau có thể khiến hàm mất mát (loss function) có hình dạng kéo dài, làm chậm quá trình hội tụ hoặc mắc kẹt tại các cực tiểu cục bộ. Chuẩn hóa giúp hàm mất mát "tròn" hơn, tạo điều kiện thuận lợi cho các thuật toán tối ưu tìm kiếm nghiệm hiệu quả hơn.

Cải thiện độ ổn định số học: Khi các giá trị quá lớn hoặc quá nhỏ, các phép tính có thể gặp vấn đề về độ chính xác số học (numerical precision) trên máy tính, dẫn đến kết quả không mong muốn.

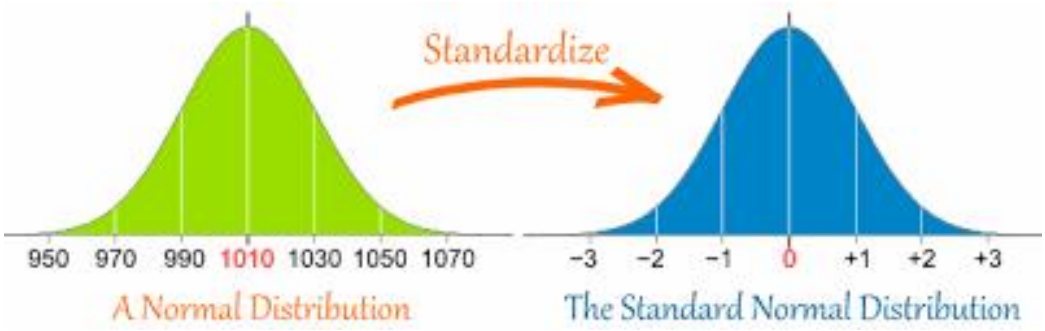
- **Các phương pháp chuẩn hóa phổ biến:**



Hình 2.3: Min-Max Scaling

Min-Max Scaling (Normalization): Chuẩn hóa các giá trị của đặc trưng để chúng nằm trong một phạm vi cố định, thường là từ 0 đến 1 hoặc từ -1 đến 1. Đối với dữ liệu ảnh, giá trị cường độ điểm ảnh thường nằm trong khoảng từ 0 đến 255 (đối với ảnh 8-bit). Một cách phổ biến để chuẩn hóa các đặc trưng này là chia cường độ điểm ảnh cho 255, để chúng trở thành các giá trị số thực trong khoảng. Phương pháp này

hữu ích khi bạn biết phạm vi phân phối của dữ liệu và muốn duy trì mối quan hệ tương đối giữa các giá trị.



Hình 2.4: Standardization

Standardization (Z-score Normalization): Biến đổi các giá trị đặc trưng để chúng có giá trị trung bình là 0 và độ lệch chuẩn là 1. Phương pháp này hữu ích khi dữ liệu tuân theo phân phối Gaussian hoặc khi bạn không biết phạm vi phân phối dữ liệu ban đầu.

Điều quan trọng là cùng một phép chuẩn hóa phải được áp dụng cho cả tập dữ liệu huấn luyện và tập dữ liệu kiểm thử. Các tham số (ví dụ: giá trị trung bình và độ lệch chuẩn đối với Standardization, hoặc giá trị min/max đối với Min-Max Scaling) phải được tính toán từ chỉ tập huấn luyện và sau đó được sử dụng để biến đổi cả tập huấn luyện và tập kiểm thử.

2.3.Các công cụ và thư viện sử dụng

Để triển khai hệ thống nhận dạng chữ số, việc lựa chọn các công cụ và thư viện phù hợp là rất quan trọng. Chúng cung cấp các nền tảng và chức năng cần thiết để xử lý dữ liệu, xây dựng mô hình học máy, và đánh giá hiệu suất.

2.3.1.Python

Python là ngôn ngữ lập trình được lựa chọn hàng đầu trong lĩnh vực học máy (Machine Learning) và khoa học dữ liệu nhờ vào nhiều ưu điểm nổi bật của nó.

Cú pháp đơn giản và dễ đọc: Python có cú pháp rõ ràng, giúp việc viết và hiểu mã dễ dàng hơn, giảm thiểu thời gian phát triển.

Hệ sinh thái thư viện phong phú: Python sở hữu một hệ sinh thái khổng lồ các

thư viện và framework chuyên biệt cho học máy, xử lý dữ liệu, và tính toán khoa học. Các thư viện này cung cấp các công cụ mạnh mẽ và tối ưu hóa cao, cho phép phát triển nhanh chóng và hiệu quả các mô hình phức tạp.

Tính linh hoạt và đa năng: Python có thể được sử dụng cho nhiều giai đoạn của một dự án học máy, từ thu thập và tiền xử lý dữ liệu, xây dựng và huấn luyện mô hình, cho đến triển khai ứng dụng và trực quan hóa kết quả.

Cộng đồng lớn và hỗ trợ mạnh mẽ: Một cộng đồng người dùng và nhà phát triển lớn mạnh đồng nghĩa với việc có nhiều tài liệu, hướng dẫn và sự hỗ trợ khi gặp vấn đề.

Phiên bản Python khuyến nghị: Để tận dụng tối đa các tính năng mới và đảm bảo khả năng tương thích với các thư viện hiện đại, phiên bản Python 3 (và các phiên bản mới nhất) luôn được khuyến nghị sử dụng cho các dự án mới. Python 2.7+ đã lỗi thời và nhiều thư viện lớn đang dần ngừng hỗ trợ nó.

Môi trường phát triển: Để quản lý các phiên bản thư viện và tránh xung đột giữa các dự án khác nhau, việc sử dụng môi trường ảo (virtual environments) như virtualenv là một phương pháp được khuyến nghị mạnh mẽ. Khi hoạt động trong môi trường ảo, mọi gói thư viện được cài đặt thông qua pip sẽ chỉ nằm trong môi trường đó, không ảnh hưởng đến các dự án khác trên hệ thống.

Python sẽ là ngôn ngữ nền tảng để tích hợp và điều khiển các thư viện chuyên biệt khác như NumPy, Matplotlib, và Scikit-learn, những công cụ sẽ được trình bày chi tiết hơn trong các mục tiếp theo và được sử dụng xuyên suốt quá trình xây dựng hệ thống nhận dạng chữ số.

2.3.2.NumPy, Pandas, Matplotlib

- **NumPy**

NumPy (Numerical Python) là một trong những gói cơ bản nhất cho tính toán khoa học trong Python. Ưu điểm cốt lõi của NumPy là khả năng xử lý mảng đa chiều (ndarray).

Cấu trúc dữ liệu trung tâm: Trong scikit-learn, mảng NumPy là cấu trúc dữ liệu

cơ bản để nhập dữ liệu. Bất kỳ dữ liệu nào bạn sử dụng đều phải được chuyển đổi sang định dạng mảng NumPy.

Chức năng toán học mạnh mẽ: NumPy cung cấp các hàm toán học cấp cao, bao gồm các phép toán đại số tuyến tính, biến đổi Fourier và các bộ tạo số giả ngẫu nhiên.

Hiệu suất tối ưu: Mặc dù các mảng NumPy tương tự như danh sách Python, chúng được tối ưu hóa cao cho các phép toán số học, cho phép xử lý dữ liệu lớn một cách hiệu quả. Để đạt hiệu suất tối ưu, nên sử dụng numpy.ndarray (mảng dày đặc) với dtype=float64.

Hình 2.5: NumPy, Pandas, Matplotlib

- **Pandas**

Pandas là một thư viện Python chuyên dụng cho việc xử lý và phân tích dữ liệu (data wrangling and analysis). Nó được xây dựng dựa trên một cấu trúc dữ liệu quan trọng là DataFrame, hoạt động tương tự như một bảng tính Excel hoặc một bảng trong cơ sở dữ liệu.

DataFrame linh hoạt: Khác với NumPy yêu cầu tất cả các phần tử trong một mảng phải cùng kiểu, Pandas cho phép mỗi cột trong DataFrame có một kiểu dữ liệu riêng biệt (ví dụ: số nguyên, ngày tháng, số dấu phẩy động và chuỗi).

Công cụ thao tác dữ liệu: Pandas cung cấp một loạt các phương thức để sửa đổi và thao tác trên bảng dữ liệu, bao gồm các truy vấn giống SQL và việc kết nối các bảng.

Khả năng nhập/xuất dữ liệu đa dạng: Một ưu điểm quan trọng khác của Pandas là khả năng đọc dữ liệu từ nhiều định dạng tệp và cơ sở dữ liệu khác nhau, như SQL, tệp Excel và tệp CSV. Điều này được thể hiện trong các ví dụ tải dữ liệu như pd.read_csv.

Hỗ trợ khám phá dữ liệu: Pandas giúp dễ dàng có được cái nhìn tổng quan về dữ liệu, ví dụ như qua phương thức describe() để xem tóm tắt các thuộc tính số hoặc hiển thị các hàng đầu tiên của DataFrame.

- **Matplotlib**

Matplotlib là thư viện cốt lõi cho trực quan hóa khoa học trong Python, cho phép tạo ra các biểu đồ chất lượng cao dùng cho xuất bản.

Đa dạng biểu đồ: Matplotlib cung cấp các chức năng để tạo nhiều loại biểu đồ khác nhau như biểu đồ đường (line charts), biểu đồ tần suất (histograms), biểu đồ phân tán (scatter plots).

Hỗ trợ khám phá dữ liệu: Việc trực quan hóa dữ liệu và các khía cạnh khác nhau của phân tích có thể mang lại những hiểu biết quan trọng. Ví dụ, nó được sử dụng để vẽ biểu đồ phân tán của các thuộc tính, hiển thị histogram của dữ liệu, hoặc đường ranh giới quyết định của các mô hình.

Tích hợp Jupyter Notebook: Khi làm việc trong Jupyter Notebook, có thể hiển thị biểu đồ trực tiếp trong trình duyệt bằng cách sử dụng các lệnh magic như %matplotlib inline. Điều này giúp quá trình phân tích và trực quan hóa trở nên tương tác và liền mạch.

2.3.3.Mô hình SVM tự xây dựng

Việc tự xây dựng mô hình SVM từ đầu, mặc dù phức tạp và tốn thời gian hơn, mang lại sự hiểu biết sâu sắc về cơ chế hoạt động bên trong của thuật toán.

Cơ sở toán học: Quá trình này yêu cầu nắm vững các khái niệm toán học như tìm siêu phẳng phân chia tối ưu (optimal hyperplane) bằng cách cực đại hóa biên độ (margin) giữa các lớp.

Bài toán tối ưu: Bài toán SVM được xây dựng dưới dạng một bài toán tối ưu lồi

(convex optimization problem), cụ thể là lập trình bậc hai (Quadratic Programming – QP). Việc giải bài toán này có thể thông qua các công cụ giải QP chuyên biệt hoặc thông qua việc chuyển đổi sang bài toán đối ngẫu (dual problem).

Support Vectors: Các điểm dữ liệu nằm gần biên độ nhất (được gọi là Support Vectors) đóng vai trò then chốt trong việc xác định siêu phẳng phân loại. Nghiệm của bài toán đối ngẫu thường là sparse (thưa thớt), chỉ một số ít các trọng số tương ứng với các support vector là khác không.

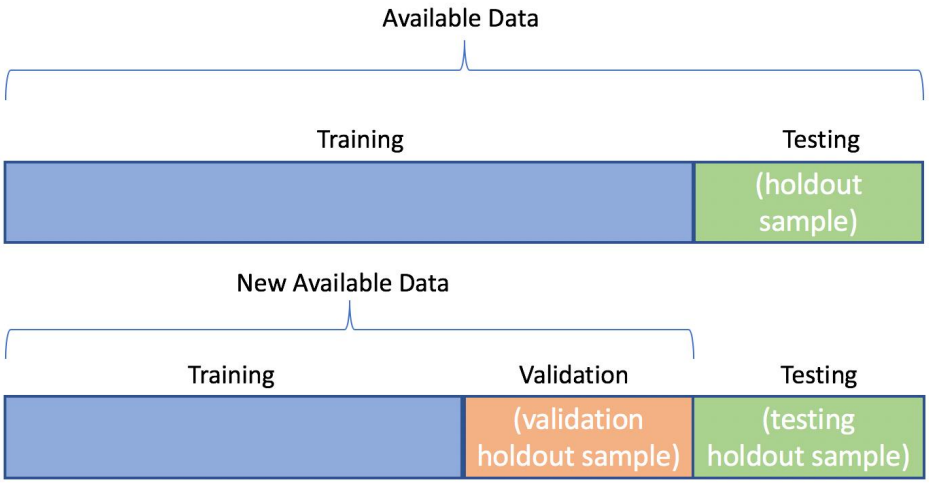
2.4.Đánh giá mô hình học máy

Sau khi xây dựng và huấn luyện một mô hình học máy, việc đánh giá hiệu suất của nó là bước tối quan trọng để xác định mức độ hiệu quả của mô hình trong việc giải quyết bài toán đặt ra. Một mô hình tốt không chỉ phải học tốt trên dữ liệu huấn luyện mà còn phải có khả năng tổng quát hóa (generalize) tốt trên dữ liệu mới mà nó chưa từng thấy. Quá trình đánh giá giúp chúng ta hiểu được ưu và nhược điểm của mô hình, từ đó đưa ra các cải tiến cần thiết.

2.4.1.Phân chia dữ liệu để đánh giá

Để đánh giá một mô hình một cách khách quan, dữ liệu thường được chia thành các tập con riêng biệt.

Tập huấn luyện (Training Set): Là phần lớn dữ liệu được sử dụng để huấn luyện mô hình, tức là để mô hình học các mẫu và mối quan hệ từ dữ liệu.



Hình 2.6: Chia dữ bộ dữ liệu

Tập kiểm định (Validation Set): Là một tập dữ liệu nhỏ hơn, được tách ra từ tập huấn luyện, dùng để tinh chỉnh siêu tham số (hyperparameters) của mô hình và để kiểm tra hiệu suất sơ bộ trong quá trình phát triển. Việc sử dụng tập kiểm định giúp tránh việc mô hình "học thuộc lòng" tập huấn luyện mà không tổng quát hóa được.

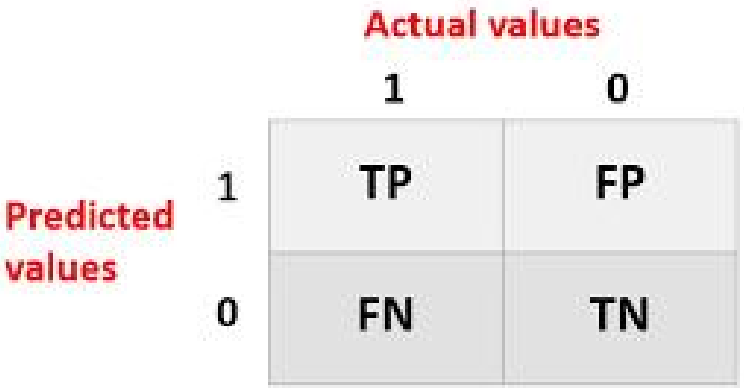
Tập kiểm tra (Test Set): Là tập dữ liệu được giữ lại hoàn toàn tách biệt khỏi quá trình huấn luyện và tinh chỉnh siêu tham số. Nó được sử dụng chỉ một lần duy nhất ở cuối cùng để đánh giá hiệu suất tổng quát cuối cùng của mô hình trên dữ liệu chưa từng thấy. Điều này đảm bảo rằng kết quả đánh giá là một ước lượng không thiên vị về khả năng tổng quát hóa của mô hình trong thực tế.

Một kỹ thuật phổ biến để khắc phục hạn chế về lượng dữ liệu và thu được ước tính hiệu suất đáng tin cậy hơn là kiểm định chéo (cross-validation). K-fold cross-validation chia tập dữ liệu thành K phần (folds). Mô hình được huấn luyện K lần; mỗi lần, một phần được dùng làm tập kiểm định và K-1 phần còn lại dùng làm tập huấn luyện. Sau đó, kết quả từ K lần chạy được tổng hợp để đưa ra đánh giá cuối cùng về mô hình.

2.4.2.Các chỉ số đánh giá cho bài toán phân loại

Vì nhận dạng chữ số là một bài toán phân loại (classification), các chỉ số đánh giá sau đây thường được sử dụng:

Độ chính xác (Accuracy): Là tỷ lệ các mẫu được phân loại đúng trên tổng số mẫu. Đây là chỉ số trực quan và dễ hiểu nhất, nhưng có thể gây hiểu lầm trong trường hợp dữ liệu mất cân bằng lớp.



Hình 2.7: Ma trận nhầm lẫn (Confusion Matrix)

Ma trận nhầm lẫn (Confusion Matrix): Cung cấp cái nhìn chi tiết hơn về hiệu suất phân loại bằng cách hiển thị số lượng các mẫu được phân loại đúng và sai cho mỗi lớp.

True Positives (TP): Số lượng mẫu dương được phân loại đúng là dương.

True Negatives (TN): Số lượng mẫu âm được phân loại đúng là âm.

False Positives (FP): Số lượng mẫu âm bị phân loại sai là dương (lỗi loại I).

False Negatives (FN): Số lượng mẫu dương bị phân loại sai là âm (lỗi loại II).

Độ chính xác (Precision): Tỷ lệ các mẫu được dự đoán là dương thực sự là dương. Nó trả lời câu hỏi: "Trong số các mẫu mà mô hình dự đoán là dương, có bao nhiêu mẫu thực sự là dương?".

Công thức (ngoài nguồn nhưng tính toán từ các yếu tố trong nguồn): Precision = TP / (TP + FP)

Độ thu hồi (Recall) (hay Độ nhạy - Sensitivity): Tỷ lệ các mẫu dương thực sự được mô hình phát hiện. Nó trả lời câu hỏi: "Trong số tất cả các mẫu thực sự là dương, mô hình đã tìm thấy bao nhiêu?".

Công thức (ngoài nguồn nhưng tính toán từ các yếu tố trong nguồn): Recall = TP / (TP + FN)

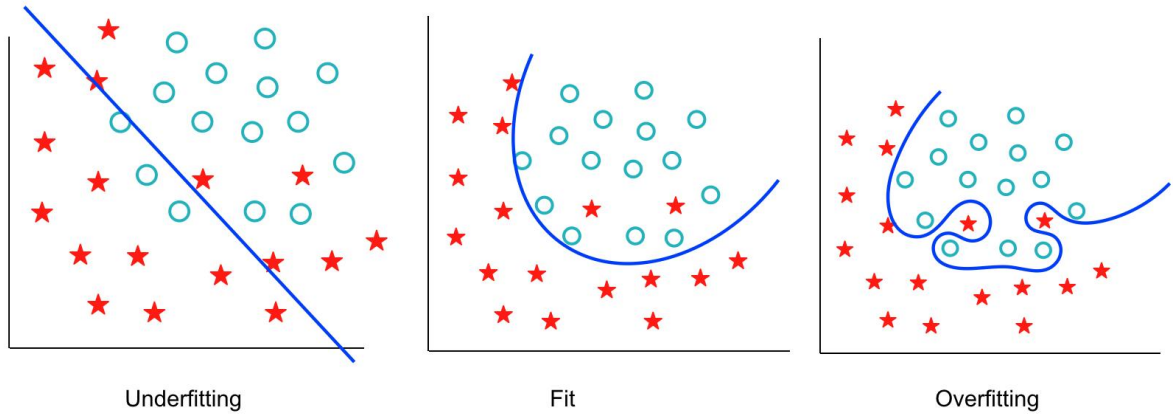
F1-Score: Là trung bình điều hòa của Precision và Recall. Chỉ số này hữu ích khi có sự mất cân bằng giữa Precision và Recall.

Công thức (ngoài nguồn nhưng tính toán từ các yếu tố trong nguồn): F1-Score =

2 * (Precision * Recall) / (Precision + Recall)

2.4.3.Hiện tượng Overfitting và Underfitting

Trong quá trình đánh giá, việc nhận biết và xử lý hiện tượng overfitting (quá khớp) và underfitting (kém khớp) là rất quan trọng:



Hình 2.8: Overfitting và Underfitting

Overfitting: Xảy ra khi mô hình học quá kỹ các chi tiết và nhiễu trong dữ liệu huấn luyện, dẫn đến việc mô hình hoạt động rất tốt trên tập huấn luyện nhưng lại kém hiệu quả trên dữ liệu mới hoặc tập kiểm tra. Các dấu hiệu của overfitting bao gồm điểm số cao trên tập huấn luyện nhưng thấp trên tập kiểm định/kiểm tra.

Underfitting: Xảy ra khi mô hình quá đơn giản để nắm bắt được các mẫu cơ bản trong dữ liệu, dẫn đến hiệu suất kém trên cả tập huấn luyện và tập kiểm tra.

Việc quản lý overfitting thường bao gồm các kỹ thuật như điều hòa (regularization) (ví dụ: điều chỉnh siêu tham số C trong SVM để kiểm soát sự cân bằng giữa biên độ lớn và số lỗi vi phạm biên độ ít, hoặc thêm các ràng buộc vào hàm mục tiêu) hoặc tăng thêm dữ liệu huấn luyện.

CHƯƠNG 3: XÂY DỰNG HỆ THỐNG NHẬN DẠNG CHỮ SỐ

3.1.Kiến trúc tổng thể của hệ thống

Hệ thống nhận dạng chữ số viết tay sử dụng bộ dữ liệu MNIST được thiết kế theo kiến trúc tổng thể gồm ba tầng chính, bao gồm: tầng tiền xử lý dữ liệu, tầng huấn luyện và đánh giá mô hình học máy, và tầng giao diện ứng dụng người dùng. Mỗi tầng đảm nhiệm một vai trò cụ thể, phối hợp chặt chẽ để tạo nên một hệ thống hoàn chỉnh, có khả năng học từ dữ liệu, đưa ra đoán chính xác, và phục vụ tương tác trực tiếp với người dùng.

Hình 3.1: Kiến trúc tổng thể của hệ thống

- **Tầng tiền xử lý dữ liệu**

Dữ liệu gốc được lấy từ bộ dữ liệu MNIST, bao gồm 60.000 ảnh huấn luyện và 10.000 ảnh kiểm tra, mỗi ảnh là một chữ số viết tay từ 0 đến 9, với kích thước 28x28 (định dạng ảnh xám).

Ở tầng này, hệ thống thực các bước xử lý như: Chuẩn hóa dữ liệu (normalization) để đưa giá trị pixel về cùng thang [0,1]. Làm phẳng ảnh (flatten) nếu sử dụng mô hình SVM dạng vector. Giảm nhiễu, giảm chiều dữ liệu (nếu cần, ví dụ bằng PCA). Phân chia dữ liệu thành tập huấn luyện và tập kiểm tra.

- **Tầng huấn luyện và dự đoán**

Đây là tầng lõi của hệ thống, nơi thực hiện việc huấn luyện và kiểm tra mô hình học máy.

Thuận toán chính sử dụng trong hệ thống là Support Vector Machine (SVM), một phương pháp phân loại mạnh mẽ cho các bài toán có nhiều đặc trưng đầu vào.

Các bước chính gồm: Huấn luyện mô hình với tập dữ liệu đã xử lý. Tối ưu hàm mất mát regularization (chính quy hóa) và gradient descent (suy giảm độ dốc) hoặc kỹ thuật tối ưu khác. Dự đoán kết quả cho dữ liệu mới dựa trên siêu phẳng phân chia (hyperplane). Đánh giá mô hình dựa trên các chỉ số như độ chính xác, ma trận nhầm lẫn (confusion matrix), F1-score,…

- **Tầng giao diện người dùng**

Tầng giao diện cho phép người dùng tương tác với hệ thông qua các hình thức như vẽ chữ số trực tiếp trên canvas hoặc tải ảnh số lên, nhận kết quả dự đoán trả về từ mô hình.

Giao diện có thể được xây dựng bằng Flask (Python backend) kết hợp HTML/CSS và JavaScript cho phần frontend.

Dữ liệu người dùng nhập vào được tiền xử lý tương tự dữ liệu MNIST, sau đó được đưa vào mô hình để dự đoán.

Kiến trúc hệ thống được thiết kế nhằm đảm bảo sự độc lập giữa các tầng chức năng, giúp hệ thống dễ bảo trì, mổ rộng và tối ưu. Với cách tổ chức này, hệ thống có thể vừa phục vụ mục đích nghiên cứu (với mô hình SVM học máy thủ công), vừa sẵn sàng tích hợp vào ứng dụng thực tiễn nhờ tầng giao diện trực quan và linh hoạt.

3.2.Thư viện và dữ liệu sử dụng

- **Thư viện sử dụng (Import library)**

```
# Processing image
import numpy as np
import pandas as pd
# Data Visualization
import seaborn as sns
import matplotlib.pyplot as plt
# Data Processsing
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
```

Numpy và pandas là các thư viện cơ bản dùng để thao tác với dữ liệu. Numpy cung cấp các công cụ để làm việc với mảng (arrays), trong khi pandas hỗ trợ làm việc với dữ liệu dạng bảng (DataFrame), dễ dàng cho việc thao tác và xử lý dữ liệu.

- **Dữ liệu sử dụng**

```
# Dataset
from tensorflow.keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

from tensorflow.keras.datasets import mnist: Nhập dữ liệu MNIST từ Keras.

(X_train, y_train), (X_test, y_test) = mnist.load_data(): Tải dữ liệu và chia thành tập huấn luyện và tập kiểm tra:

X_train và y_train: Ảnh huấn luyện và nhãn của chúng.

X_test và y_test: Ảnh kiểm tra và nhãn của chúng.

X_train.shape, y_train.shape, X_test.shape, y_test.shape: Hiển thị kích thước của từng tập dữ liệu. Kích thước dự kiến:

X_train.shape: (60000, 28, 28) – 60,000 ảnh huấn luyện, mỗi ảnh có kích thước

28x28 pixel.

y_train.shape: (60000,) – 60,000 nhãn cho các ảnh huấn luyện.

X_test.shape: (10000, 28, 28) – 10,000 ảnh kiểm tra.

y_test.shape: (10000,) – 10,000 nhãn cho các ảnh kiểm tra.

3.3.Tiền xử lý dữ liệu

```
1  # Image Preprocessing: Normalize the data (scale to 0-1 range)
2  X_train = X_train.astype('float32') / 255
3  X_test = X_test.astype('float32') / 255
```

Chuyển đổi kiểu dữ liệu của ảnh từ uint8 sang float32:

- X_train.astype('float32'): Chuyển đổi X_train từ kiểu dữ liệu nguyên không dấu uint8 (với giá trị pixel từ 0 đến 255) sang float32.
- Tương tự, X_test.astype('float32'): Chuyển đổi X_test sang kiểu float32.
- Việc chuyển đổi sang float32 giúp tăng độ chính xác trong quá trình tính toán, nhất là khi xử lý bằng các mô hình học máy hoặc học sâu.

Chuẩn hóa dữ liệu về phạm vi [0, 1]:

- X_train.astype('float32') / 255 và X_test.astype('float32') / 255: Sau khi chuyển đổi, giá trị của mỗi pixel được chia cho 255 để chuyển đổi các giá trị từ phạm vi [0, 255] về [0, 1].
- Chuẩn hóa dữ liệu về phạm vi [0, 1] giúp mô hình học máy hoặc học sâu hội tụ nhanh hơn và có độ ổn định cao hơn, do các giá trị không quá lớn.

```
1  # Reshape 28x28 images to 1D vector of 784
2  X_train = X_train.reshape(-1, 784)
3  X_test = X_test.reshape(-1, 784)
4  X_train.shape, X_test.shape
```

Thực hiện việc chuyển đổi kích thước của ảnh từ dạng ma trận 28×28 thành một vector 1 chiều gồm 784 phần tử (bởi vì 28×28=784). Việc chuyển đổi này giúp các mô hình học máy xử lý dữ liệu dễ dàng hơn khi đầu vào là một vector 1D thay vì ma trận 2D.

```
# Chage array NumPy to DataFrame
df_X_train = pd.DataFrame(X_train.reshape(-1, 28 * 28))
df_y_train = pd.DataFrame(y_train)
df_X_test = pd.DataFrame(X_test.reshape(-1, 28 * 28))
df_y_test = pd.DataFrame(y_test)
# Print DataFrame
df_X_train.head()
df_y_train.head()
# Rename column to label
df_y_train = df_y_train.rename(columns={0: 'label'})
df_digit = pd.concat([df_X_train, df_y_train], axis=1)
df_digit.head()
```

Chuyển đổi dữ liệu ảnh và nhãn từ NumPy sang pandas DataFrame, thực hiện một số thao tác tiền xử lý (như đổi tên cột), và kết hợp dữ liệu ảnh và nhãn để thuận tiện cho việc phân tích và huấn luyện mô hình.

```
# Đổi tên các cột
new_column_names = ['pixel_' + str(i) for i in range(784)] + ['label']
df_digit.columns = new_column_names
df_symboils=
pd.read_csv('/content/drive/MyDrive/symbols_dataset_new.csv')
df_symboils.columns = new_column_names
# Kết hợp hai bảng DataFrame train và df_symboils theo chiều dọc (axis=0)
df_concat = pd.concat([df_digit, df_symboils])
# Đặt lại chỉ mục của DataFrame kết hợp
df_concat = df_concat.reset_index(drop=True)
# In ra DataFrame kết hợp
df_concat.tail()
```

Tạo tên cột mới cho dữ liệu ảnh và nhãn, áp dụng chúng cho df_digit và

df_symboils, sau đó hiển thị 5 hàng đầu của df_symboils.

```
# Giả sử df_concat là DataFrame của bạn
label_mapping = {
'+': 10,
'-': 11,
'*': 12,
'/': 13
}
#  Thay thế nhãn trong cột 'nhãn' bằng cách sử dụng ánh xạ
df_concat['label'] = df_concat['label'].replace(label_mapping)
```

Thay thế các ký hiệu trong cột 'label' của df_concat bằng mã số tương ứng theo label_mapping.

```
# Tập dữ liệu Imblearn
smote = SMOTE(random_state = 42)
X_resampled, y_resampled = smote.fit_resample(X, y)
print(y.value_counts())
print(y_resampled.value_counts())
```

Sử dụng SMOTE để cân bằng lại dữ liệu bằng cách tạo mẫu mới cho lớp thiểu số trong y, sau đó in ra số lượng mẫu trước và sau khi cân bằng.

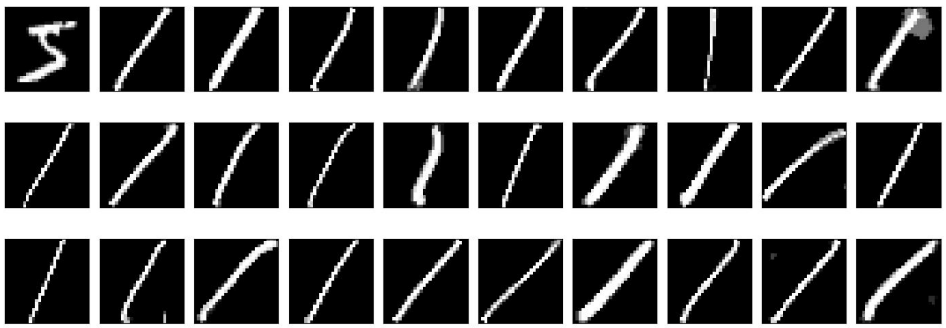
3.4.Trực quan hóa dữ liệu

```
# Vẽ chữ số hình ảnh
plt.figure(figsize=(14, 12))
for digit_num in range(0, 30):
    plt.subplot(7, 10, digit_num + 1)
    grid_data = X_resampled.iloc[-digit_num].values.reshape(28, 28)
    plt.imshow(grid_data, interpolation='none', cmap= 'gray')
    plt.xticks([])
    plt.yticks([])
```

```
plt.tight_layout()
plt.show()
```

Hiển thị 30 hình ảnh chữ số từ tập X_resampled dưới dạng lưới 7x10 với kích thước 28x28 pixel.

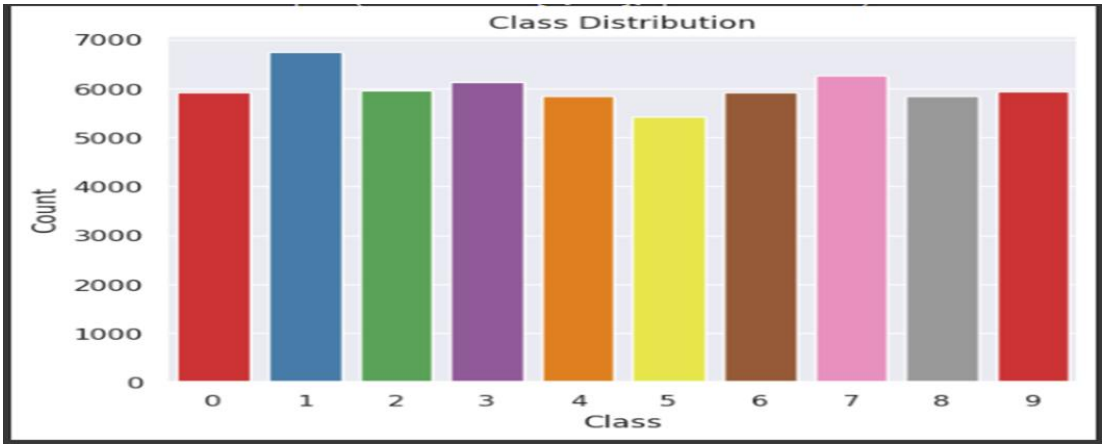
Output:



Hình 3.2: Hình ảnh các label

```
# Khám phá sự phân bổ lớp học (gần như phân bổ đều)
sns.set(style="darkgrid")
# Sử dụng cú pháp đúng cho countplot
counts = sns.countplot(x= y_resampled, palette='Set1')
# Thêm nhãn và tiêu đề
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Class Distribution')
plt.show()
```

Vẽ biểu đồ phân bố lớp học của y_resampled bằng countplot, thêm nhãn trục và tiêu đề, rồi hiển thị biểu đồ.



Hình 3.3: Số lượng các label

3.5.Xây dựng mô hình Support Vector Machine

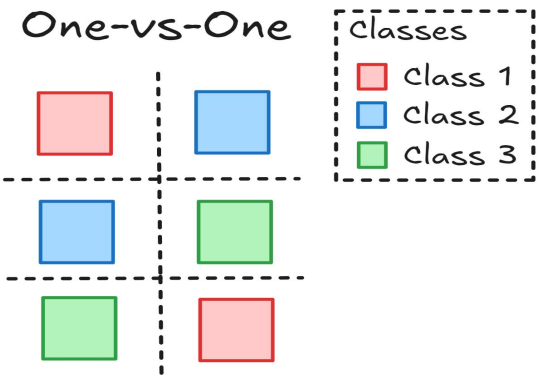
Trong hệ thống nhận dạng chữ số viết tay, mô hình Support Vector Machine (SVM) được lựa chọn làm phương pháp học máy tính để phân loại các ảnh chữ số từ bộ dữ liệu MNIST. SVM là một thuật toán mạnh mẽ trong việc xử lý các bài toán phân loại nhị phân và có thể mở rộng cho bài toán đa lớp bằng cách sử dụng chiến lược *one-vs-all* hoặc *one-vs-one*.

- **One-vs-one**

Phương pháp One-vs-One xây dựng một bộ phân loại nhị phân cho mỗi cặp lớp trong tập dữ liệu. Với C lớp, ta sẽ cần xây dựng tổng cộng:

$$\frac{C \times (C-1)}{2}$$

bộ phân loại. Ví dụ, với MNIST có 10 lớp (chữ số từ 0 đến 9), ta cần huấn luyện 45 bộ SVM khác nhau.



Hình 3.4: One-vs-One

Khi có một mẫu đầu vào mới, hệ thống sẽ đưa mẫu này qua toàn bộ các bộ phận loại nhị phân, và mỗi bộ sẽ bỏ phiếu cho một trong hai lớp của nó. Lớp nhận được nhiều phiếu bầu nhất sẽ được chọn làm kết quả cuối cùng.

Ưu điểm:

Tập trung phân biệt giữa hai lớp cụ thể, nên mô hình có thể đạt độ chính xác cao hơn, đặt biệt trong trường hợp các lớp có đặc trưng chống lấn.

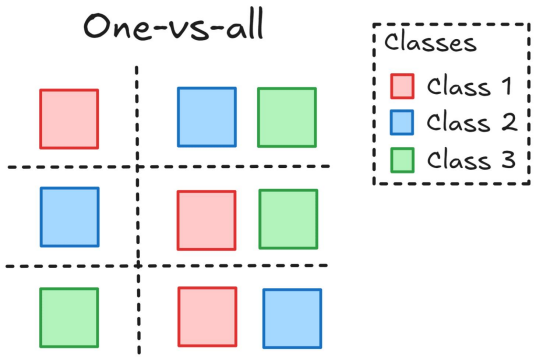
Hạn chế:

Số lượng bộ phận loại tăng nhanh theo cấp số nhân khi số lớp tăng, gây tốn thời gian và tài nguyên tính toán.

Các mẫu có thể được phân loại bởi những bộ classifier không liên quan (Ví dụ: mẫu số 1 được phân loại bởi bộ 5 và 6).

- **One-vs-all**

Phương pháp One-vs-all (còn gọi là One-vs-Rest) là chiến lược được sử dụng phổ biến hơn trong thực tế. Với C lớp, OvA huấn luyện C bộ phân loại nhị phân, mỗi bộ dùng để phân biệt một lớp cụ thể với toàn bộ các lớp còn lại.



Hình 3.5: One-vs-all

Ví dụ: Bộ classifier thứ nhất phân biệt “chữ số 0” với “không phải 0”, bộ thứ hai phân biệt “chữ số 1” với “không phải 1” cho đến chữ số 9.

Khi có mẫu mới, mỗi bộ phân loại trả về điểm tin cậy hoặc xác suất thuộc về lớp của nó. Lớp có điểm số cao nhất sẽ được chọn là đầu ra.

Ưu điểm:

Số lượng classifier chỉ bằng số lớp ít tài nguyên hơn OvO.

Phù hợp với các bài toán có dữ liệu mất cân bằng giữa các lớp.

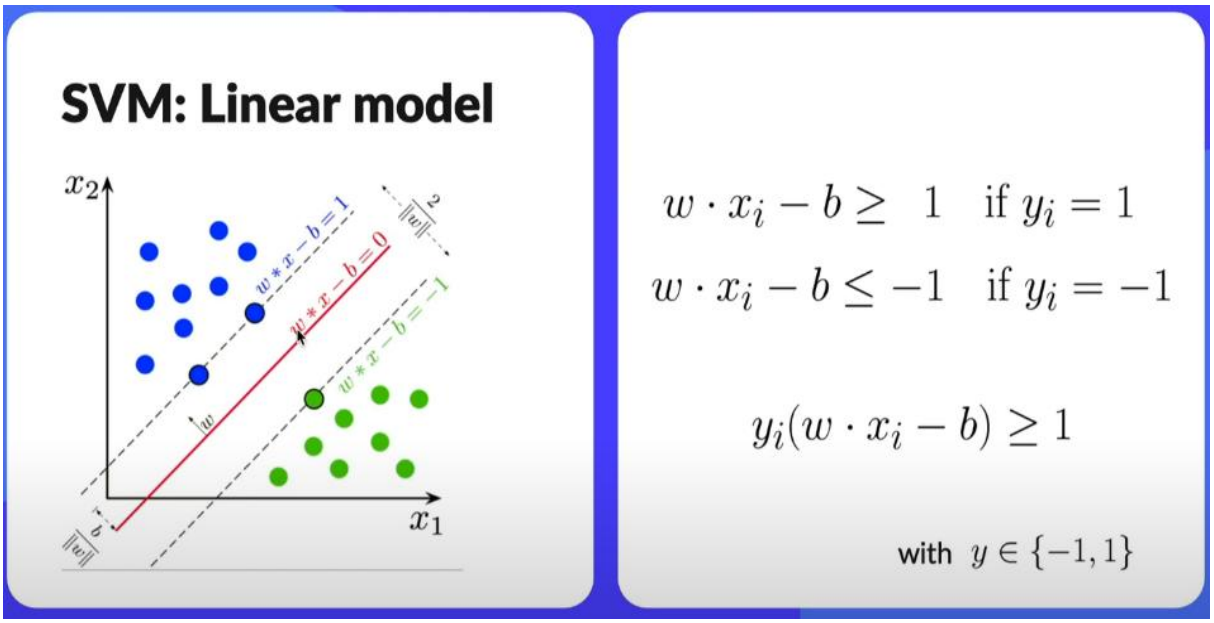
Hạn chế:

OvA có thể kém hiệu quả khi các lớp bị chồng lấn đặc trưng nhiều bộ phân loại có thể cùng tự tin khẳng định đầu vào thuộc về lớp của mình.

Dễ bị nhiễu do dữ liệu biên giới, vì mô hình phải học phân tách giữa một lớp và tất cả phần còn lại.

3.5.1.Siêu phẳng phân chia (Hyperplane)

Trong Support Vector Machine (SVM), siêu phẳng phân chia đóng vai trò là ranh giới quyết định để phân tách các lớp dữ liệu. Đối với bài toán nhận dạng chữ số viết tay sử dụng bộ dữ liệu MNIST, các ảnh chữ số được biểu diễn dưới dạng vector đặc trưng có chiều cao (ví dụ, một ảnh 28x28 pixel được coi là một điểm dữ liệu trong không gian 784 chiều). SVM đặc biệt hiệu quả trong các không gian có chiều cao như vậy.



Hình 3.6: Công thức Hyperlane

Mục tiêu chính khi xây dựng mô hình SVM cho bài toán này là tìm ra một hoặc nhiều siêu phẳng (tùy thuộc vào chiến lược phân loại đa lớp) sao cho khoảng cách từ siêu phẳng đó đến các điểm dữ liệu gần nhất của mỗi lớp (gọi là margin) là lớn nhất có thể. Việc tối đa hóa margin này giúp mô hình có khả năng tổng quát hóa tốt hơn và giảm thiểu lỗi phân loại cho dữ liệu mới.

Biểu diễn toán học một siêu phẳng tuyến tính được biểu diễn bằng phương trình

$\mathbf{w}^T\mathbf{x} + b = 0$, trong đó **w** là vector pháp tuyến của siêu phẳng (xác định hướng của nó) và **b** là hệ số tự do (xác định vị trí của nó). Khi huấn luyện mô hình SVM trên bộ dữ liệu MNIST, thuật toán sẽ giải một bài toán tối ưu lồi để tìm ra các giá trị tối ưu cho **w** và **b** sao cho margin được cực đại hóa. Trong thực tế, các thư viện như Scikit-learn cho phép truy cập các tham số này thông qua các thuộc tính coef_ (tương ứng với **w**) và intercept_ (tương ứng với **b**) của mô hình sau khi huấn luyện.

- **Kỹ thuật Kernel (Kernel Trick) cho dữ liệu phi tuyến tính:**

Dữ liệu ảnh chữ số viết tay MNIST, mặc dù có chiều cao, có thể không phân tách tuyến tính một cách rõ ràng trong không gian pixel gốc. Để giải quyết vấn đề này, SVM sử dụng kỹ thuật kernel.

Kỹ thuật này cho phép ánh xạ dữ liệu (các ảnh chữ số) từ không gian đầu vào ban đầu sang một không gian đặc trưng có chiueef cao hơn (thậm chí vô hạn chiều), nơi chúng có thể trở nên phân tách tuyến tính bởi một siêu phẳng. Điều quan trọng là kỹ thuật kernel thực hiện việc này mà không cần tính toán tường minh các đặc trưng trong không gian mới, mà chỉ cần tính toán tích vô hướng (hoặc độ đo tương tự) giữa các cặp dữ liệu trong không gian gốc.

Các hàm kernel phổ biến gồm Linear, Polynomial, Radial Basis Function (RBF). Việc lựa chọn kernel phù hợp cho bài toán MNIST (ví dụ: kernel Linear thường được sử dụng cho dữ liệu ảnh phức tạp) có thể giúp mô hình tạo ra các ranh giới phân chia tuyến tính phức tạp hơn trong không gian ban đầu, qua đó cải thiện hiệu suất nhận dạng chữ số.

- **Vector hỗ trợ (Support Vectors):**

Support vectors là các điểm dữ liệu huán luyện nằm gần siêu phẳng phân chia nhất (trên hoặc gần ranh giới của margin). Chúng là những điểm quan trọng nhất, quyết định vị trí hướng của mặt phẳng.

Trong ngữ cảnh của bài toán MNIST, các vector hỗ trợ (support vectors) chính là những hình ảnh chữ số "khó" phân loại, thường là nhưng hình ảnh mơ hồ hoặc nằm ở ranh giới giữa các lớp (ví dụ: một chữ số '4' có thể có những đặc điểm gần giống '9' hoặc '7'). Đặc tính quan trọng của SVM là hầu hết các điểm dữ liệu khác

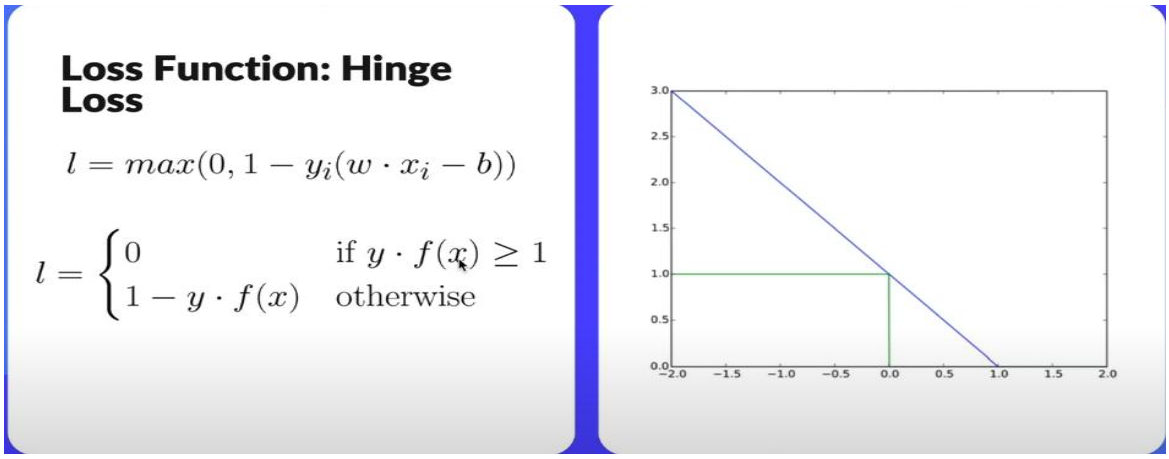
không ảnh hưởng đến siêu phẳng giúp mô hình tiết kiệm bộ nhớ và hiệu quả hơn.

- **Phân loại đa lớp cho MNIST:**

Vì SVM ban đầu được thiết kế như một bộ phân loại nhị phân (tức chỉ phân tách hai lớp), để áp dụng cho bài toán nhận dạng 10 chữ số (từ 0 đến 9) của MNIST, hệ thống cần sử dụng các chiến lược mở rộng như **One-vs-All** đây là phương pháp phổ biến, trong đó hệ thống huấn luyện **N** bộ phân loại nhị phân riêng biệt cho **N** lớp (trong trường hợp này là 10 lớp chữ số). Mỗi bộ phân loại sẽ được huấn luyện để phân biệt một chữ số cụ thể với tất cả các chữ số còn lại. Ví dụ, một bộ phân loại sẽ xác định xem ảnh là chữ số '0' hay "không phải '0'", một bộ khác xác định '1' hay "không phải '1'",…Mỗi bộ phân loại này sẽ có siêu phẳng **w** và **b** riêng biệt.

3.5.2.Hàm mất mát (Loss Function)

Để xây dựng mô hình Support Vector Machine (SVM) cho bài toán nhận dạng chữ số viết tay, việc lựa chọn và hiểu rõ Hàm mất mát (Loss Function) là vô cùng quan trọng. Hàm mất mát giúp định lượng mức độ "sai lệch" của mô hình so với dữ liệu huấn luyện, từ đó hướng dẫn quá trình tối ưu hóa để tìm ra mô hình tốt nhất.



Hình 3.7: Công thức Loss Function

Trong học máy, hàm mất mát là một hàm số định lượng mức độ "tệ" của mô hình khi đưa ra dự đoán cho một điểm dữ liệu cụ thể. Mục tiêu tổng thể của quá trình huấn luyện là tìm các tham số của mô hình sao cho tối thiểu hóa giá trị trung bình của hàm mất mát trên toàn bộ tập dữ liệu huấn luyện.

Đối với Support Vector Machine (SVM), hàm mất mát đặc trưng được sử dụng

là Hinge Loss (hàm mất mát bản lề). Hàm Hinge Loss được thiết kế đặc biệt để phù hợp với triết lý của SVM là tối đa hóa biên độ phân chia (margin).

- **Đặc điểm của Hinge Loss:**

Hinge Loss cho một điểm dữ liệu (x_i, y_i) được định nghĩa là $max0, 1 - t_i$, trong đó $t_i = y_i(w^T x_i + b)$. Giá trị t_i này, thường được gọi là "điểm quyết định" hoặc "khoảng cách chức năng", cho biết mức độ tự tin và hướng phân loại của mô hình đối với điểm dữ liệu x_i.

Nguyên tắc phạt:

Nếu một điểm dữ liệu được phân loại đúng và nằm ngoại biên độ (tức là $t_i >= 1$), Hinge Loss sẽ bằng 0. Điều này có nghĩa là mô hình không bị phạt khi dự đoán đúng và đủ tự tin.

Nếu một điểm dữ liệu bị phân loại sai hoặc nằm trong biên độ (tức là $t_i < 1$), Hinge Loss sẽ tăng lên tỷ lệ thuận với khoảng cách vi phạm. Mức phạt này khuyến khích mô hình đẩy các điểm dữ liệu ra xa khỏi siêu phẳng phân chia và nằm đúng phía của biên độ.

Ưu điểm Hinge Loss là một hàm lồi (convex function), và là một cận trên của hàm Zero-One Loss (là hàm lý tưởng, bằng 0 nếu dự đoán đúng, 1 nếu dự đoán sai, nhưng lại không lồi và khó tối ưu hóa).

Tính khả vi Hinge Loss không khả vi tại điểm t = 1. Tuy nhiên, các thuật toán tối ưu hóa như Gradient Descent vẫn có thể được áp dụng bằng cách sử dụng đạo hàm dưới.

- **Kết hợp Hinge Loss trong bài toán tối ưu của SVM:**

Để huấn luyện một mô hình SVM, chúng ta cần giải quyết một bài toán tối ưu hóa có ràng buộc. Đối với Soft Margin SVM - phiên bản phổ biến hơn cho phép một số lỗi phân loại để xử lý nhiễu và dữ liệu không phân tách tuyến tính - hàm mất mát Hinge Loss được kết hợp với một số hạng chính quy hóa (regularization term). Hàm mục tiêu tổng thể của SVM thường có dạng:

$minimize(1/2) * ||w||^2 + C * \Sigma$ (Hinge Loss cho từng điểm dữ liệu)

$(1/2) * ||w||^2$ là số hạng chính quy hóa, nhằm tối ưu hóa biên độ bằng cách giảm thiểu độ lớn của vector trọng số **w**.

Σ (Hinge Loss) là tổng các giá trị mất mát từ tất cả các điểm dữ liệu, đại diện cho phần lỗi phân loại.

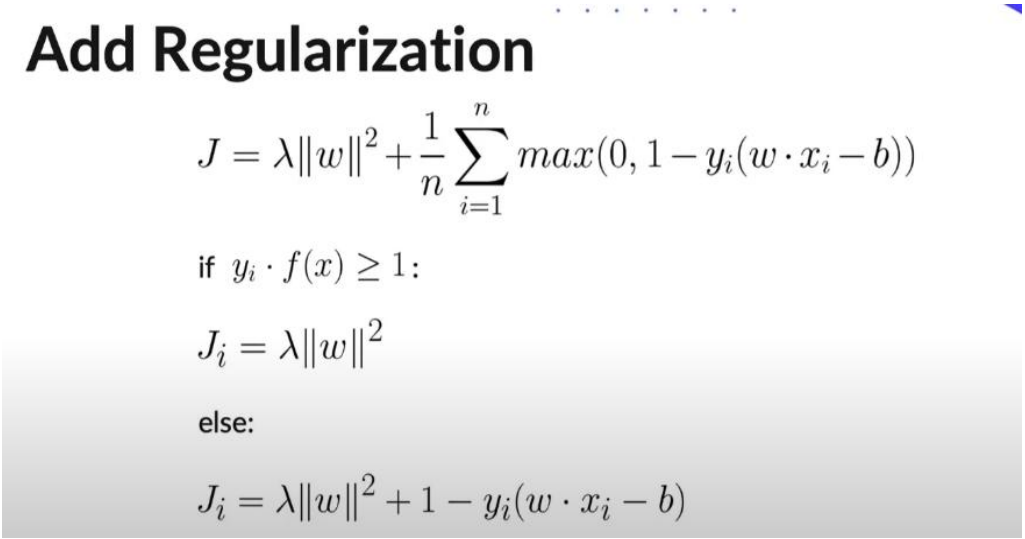
C là một tham số chính quy hóa (regularization parameter). Gía trị của **C** kiểm soát sự đánh đổi giữa việc tối đa hóa biên độ và việc chấp nhận các lỗi phân loại hoặc vi phạm biên độ.

Giá trị **C** cao buộc mô hình phải phạt nặng hơn các lỗi phân loại, dẫn đến một biên độ hẹp hơn nhưng ít lỗi huấn luyện hơn, có nguy có overfitting.

Giá trị **C** thấp cho phép nhiều lỗi phân loại hơn, dẫn đến một biên độ rộng hơn và khả năng tổng quát hóa tốt hơn, giảm nguy cơ overfitting.

3.5.3.Chính quy hóa (Regularization)

Trong học máy, chính quy hóa (Regularization) là một kỹ thuật được sử dụng để kiểm soát độ phức tạp của mô hình nhằm năng ngừa hiện tượng quá khớp (overfitting) trên dữ liệu huấn luyện và cải thiện khả năng tổng quát hóa của mô hình trên dữ liệu mới, chưa từng thấy. Nó được thực hiện bàng cách thêm một số hạng phạt vào hàm mục tiêu của mô hình, khuyến khích mô hình tìm ra các tham số "đơn giản hơn" hoặc "nhỏ hơn".



Hình 3.8: Công thức Regularization

- **Chính quy hóa được thực hiện thông qua cơ chế của Soft Margin SVM:**

Trong thực tế, rất hiếm khi dữ liệu có thể được phân tách hoàn hảo bởi một siêu phẳng tuyến tính. Dữ liệu có thể chứa nhiễu, các điểm ngoại lệ (outliers) hoặc bản chất phi tuyến tính.

Nếu cố gắng tìm một siêu phẳng phân chia cứng nhắc (Hard Margin SVM) cho dữ liệu không phân tách tuyến tính, bài toán tối ưu có thể không có lời giải hoặc cho ra một siêu phẳng quá khớp với dữ liệu huấn luyện, dẫn đến hiệu suất kém trên dữ liệu mới.

- **Cơ chế chính quy hóa trong Soft Margin SVM:**

Soft Margin SVM giải quyết vấn đề này bằng cách cho phép một số dữ liệu vi phạm biên độ (margin) hoặc thậm chí bị phân loại sai.

Để định lượng mức độ vi phạm này, các biến trược, thường được hiệu $\zeta_i (zeta)$ hoặc $xi_n (xi)$, được giới thiệu cho mỗi điểm dữ liệu.

Nếu $\zeta_i = 0$, điểm dữ liệu được phân loại đúng và nằm ngoài biên độ .

Nếu $0 < \zeta_i < 1$, điểm dữ liệu nằm trong biên độ nhưng vẫn được phân loại đúng.

Nếu $\zeta_i \geq 1$, điểm dữ liệu bị phân loại sai hoặc vượt quá biên độ.

Các biến trượt này được thêm vào hàm mục tiêu của SVM dưới dạng phạt. Mục tiêu của SVM lúc này là không chỉ tối đa hóa biên độ mà còn tối thiểu hóa tổng các vi phạm (tức là tổng các ζ_i).

Bài toán tối ưu Soft Margin SVM được biểu diễn như sau:

$$\min_{w,b,\zeta} \frac{1}{2}|w|^2 + C\sum_{i=1}^{n} \zeta_i$$

với các ràng buộc:

$y_i(w^T\phi(x_i) + b) \geq 1 - \zeta_i$ và $\zeta_i \geq 0$, $i = 0$, $i = 1, \ldots, n$.

- **Tham số chính quy hóa (Regularization Parameter) C:**

Tham số C là một siêu tham số (hyperparameter) quan trọng, có vai trò kiểm

soát sự đánh đối giữa việc tối đa hóa biên độ và việc chấp nhận các lỗi phân loại hoặc vi phạm độ.

Giá trị C cao mô hình sẽ phạt nặng hơn các lỗi phân loại và các vi phạm biên độ. Điều này buộc mô hình phải cố gắng phân loại đúng hầu hết các điểm dữ liệu huấn luyện, ngay cả khi điều đó dẫn đến một biên độ hẹp hơn. Mô hình có thể trở nên phức tạp hơn, có nguy cơ quá khớp với dữ liệu huấn luyện (overfitting).

Giá trị C thấp mô hình chấp nhận nhiều lỗi phân loại hơn và ít bị phạt hơn cho các vi phạm biên độ. Điều này dẫn đến một biên độ rộng hơn và một mô hình đơn giản hơn, có khả năng tổng quát hóa tốt hơn trên dữ liệu mới nhưng có thể có lỗi cao hơn trên tập huấn luyện.

- **Mối liên hệ với Hàm mất mát (Loss Function):**

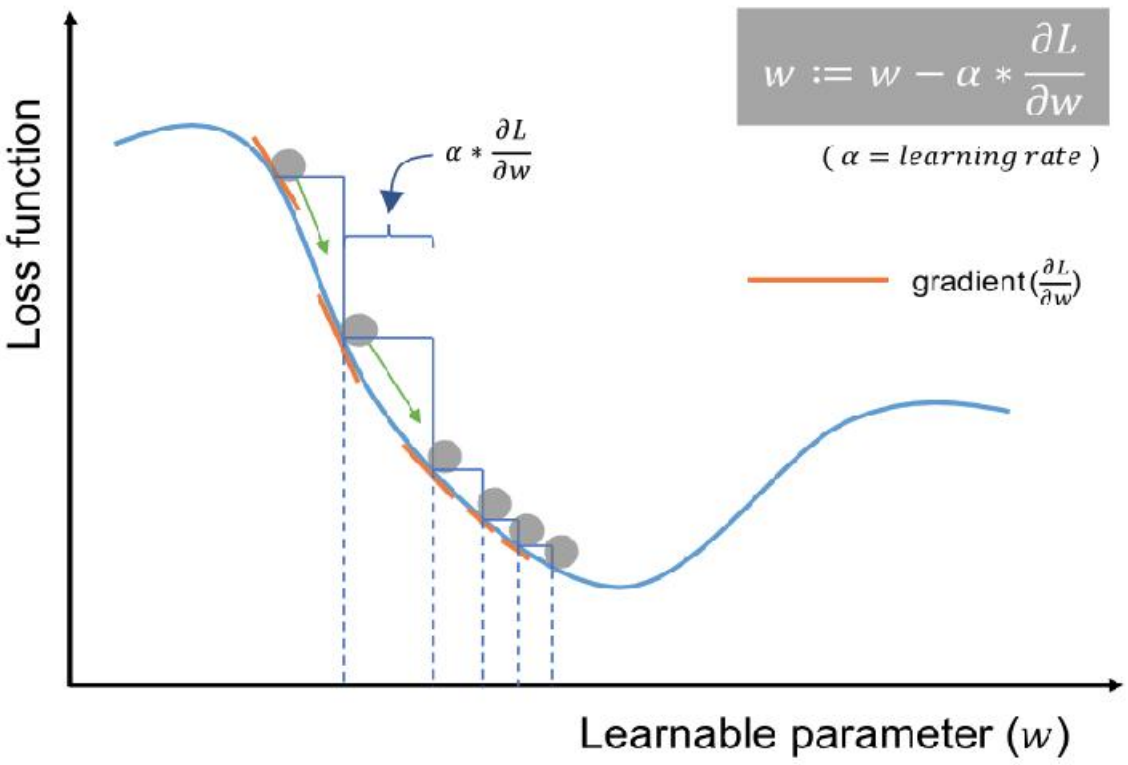
Trong công thức tối ưu của Sorf Margin SVM, số hạng $\frac{1}{2}|w|^2$ chính là số hạng chính quy hóa (regularization term). Nó khuyến khích tối đa hóa biên độ bằng cách giảm thiểu độ lớn của vector trọng số $\mathbf{w}$.

Số hạng $C\sum_{i=1}^{n}\zeta_i$ có thể được xem là tổng của Hinge Loss cho từng điểm dữ liệu. Hàm Hinge Loss phạt các điểm dữ liệu nằm sai phía của biên độ hoặc bị phân loại sai.

Do đó, bài toán tối ưu của SVM là sự kết hợp của việc tối thiểu hóa một hạng chính quy hóa (tương ứng với tối đa hóa biên độ) và một số hạng lỗi (tương ứng với Hinge Loss), với tham số C cân bằng giữa hai mục tiêu này.

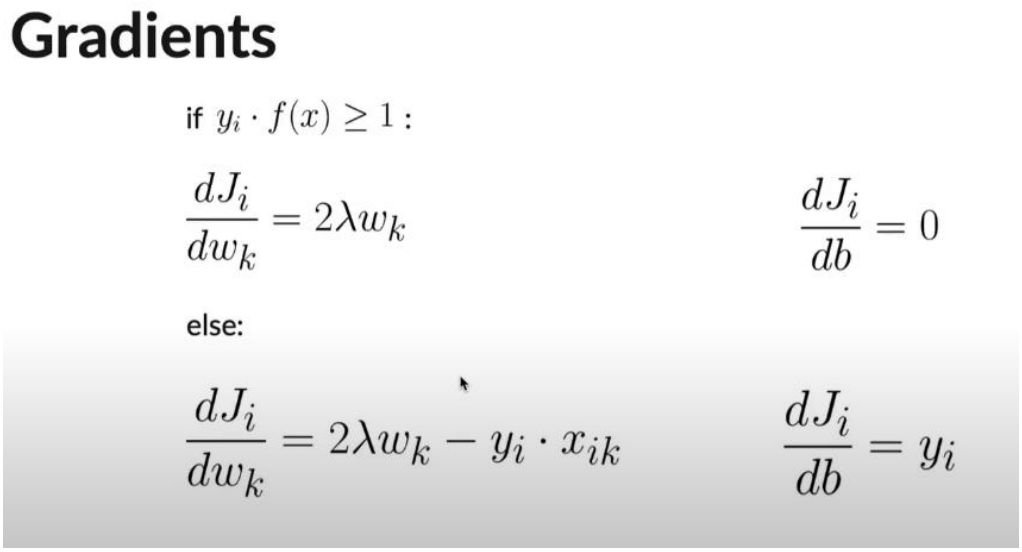
3.5.4.Suy giảm độ dốc (Gradient descent)

Để đạt được hiệu suất tối ưu va khả năng tổng quát hóa tốt cho mô hình Support Vector Machine (SVM), việc tối ưu hóa hàm mục tiêu là một bước thiết yếu. Trong bối cảnh học máy, Gradient Descent (GD) là một thuật toán tối ưu hóa mạnh mẽ va linh hoạt được sử dụng rộn rãi để tìm các tham số mô hình nhằm giảm thiểu một hàm mục tiêu (cost function).



Hình 3.9: Mô hình Gradient Decent

Gradient Descent là một thuật toán tối ưu hóa chung có khả năng tìm ra các giải pháp tối ưu cho nhiều bài toán khác nhau. Ý tưởng cơ bản của Gradient Descent là điều chỉnh các tham số của mô hình một cách lặp đi lặp lại để tối thiểu hóa hàm mục tiêu. Thuật toán bắt đầu bằng cách khởi tạo ngẫu nhiên các giá trị cho các tham số mô hình. Sau đó, nó cải thiện dần dần các tham số này, thực hiện từng bước nhỏ nhằm giảm giá trị của hàm mục tiêu. Để thực hiện mỗi bước, thuật toán tính toán gradient của hàm mục tiêu đối với từng tham số. Gradient là một vector chỉ ra hướng dốc nhất của hàm mục tiêu. Các tham số được cập nhật bằng cách trừ đi gradient nhân với một giá trị gọi là tốc độ học (learning rate - η).



Hình 3.10: Công thức Gradient Decent

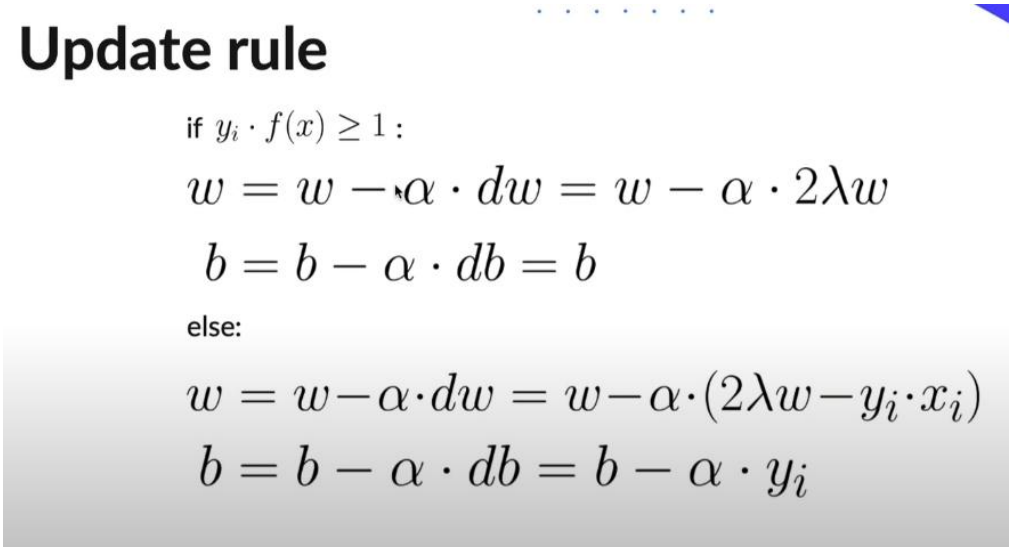
- **Hàm mục tiêu của SVM và tính lồi:**

Số hạng chính quy hóa $1/2 * ||w||^2$, nhằm khuyến khích một vector trọng số **w** nhỏ, dẫn đến biên độ lớn hơn.

Tổng của các biến trượt, đại diện cho các vi phạm biên độ, thường đo bằng Hinge Loss $(max(0, 1 - t_i * (w.T * x_i + b)))$. Hinge Loss bằng 0 nếu một điểm dữ liệu được phân loại đúng và nằm ngoài biên độ, và tăng lỷ lệ với khoảng cách vi phạm nếu điểm đó bị phân loại sai hoặc vi phạm biên độ. Mặc dù Hinge Loss không khả vi tại t = 1, Gradient Descent vẫn có thể hoạt động bằng cách sử dụng các đạo hàm con. Hàm mục tiêu của SVM là một hàm lồi (convex function), nghĩa là nó không có điểm cực tiểu cục bộ (local minima) mà chỉ có duy nhất một cực tiểu toàn cục (global minimum). Điều này bảo đảm rằng Gradient Descent sẽ hội tụ đến cực tiểu toàn cục nếu tốc độ học không quá cao và chạy đủ lâu.

- **Các biến thể của Gradient Descent trong huấn luyện SVM:**

Batch Gradient Descent (BGD): Tính toán gradient dựa trên toàn bộ tập dữ liệu huấn luyện ở mỗi bước. Mặc dù BGD được đảm bảo hội tụ đến giải pháp tối ưu nếu hàm mục tiêu là lồi và tốc độ học phù hợp, nó lại rất chậm với các tập dữ liệu lớn.



Hình 3.11: Update Gradient Descent

Stochastic Gradient Descent (SGD): Ở mỗi bước, SGD chọn một phiên bản dữ liệu ngẫu nhiên từ tập huấn luyện và tính toán gradient chỉ dựa trên phiên bản đó. Điều này làm cho thuật toán nhanh hơn đáng kể ở mỗi lần lặp và cho phép huấn luyện trên các tập dữ liệu rất lớn vì chỉ một mẫu cần nằm trong bộ nhớ tại mỗi lần lặp. Do tính chất ngẫu nhiên, hàm mục tiêu sẽ dao động lên xuống, nhưng trung bình sẽ giảm. SGD có cơ hội tốt hơn để thoát khỏi các điểm cực tiểu cục bộ nếu hàm mục tiêu không đều. Tuy nhiên, nó sẽ không bao giờ thực sự ổn định tại điểm cực tiểu mà sẽ tiếp tục dao động xung quanh nó, trừ khi tốc độ học giảm dần theo thời gian thông qua một lịch trình học (learning schedule). Trong thực tế, lớp SGDClassifier của thư viện Scikit-learn, với tham số loss="hinge", có thể được sử dụng để huấn luyện bộ phân loại SVM tuyến tính.

Mini-batch Gradient Descent (MBGD): Tính toán gradient trên các tập hợp nhỏ ngẫu nhiên của các phiên bản dữ liệu (gọi là mini-batches). Phương pháp này cân bằng giữa tốc độ của SGD và sự ổn định của BGD, đồng thời có thể tận dụng tối ưu hóa phần cứng cho các phép toán ma trận, đặc biệt khi sử dụng GPU. MBGD có xu hướng đi gần điểm cực tiểu hơn SGD, nhưng có thể khó thoát khỏi các điểm cực tiểu cục bộ hơn.

- **Các yếu tố ảnh hưởng khi sử dụng Gradient Descent cho SVM:**

Tốc độ học (Learning Rate - η) đây là một siêu thâm số cực kỳ quan trọng, quyết

định kích thước của mỗi bước di chuyển. Nếu tốc độ học quá nhỏ, thuật toán sẽ phải trải qua rất nhiều lần lặp để hội tụ, dẫn đến thời gian huấn luyện rất lâu. Ngược lại, tốc độ học quá lớn, thuật toán có thể "nhảy" qua điểm cực tiểu và thậm chí còn đi xa hơn khỏi giải pháp tốt nhất, khiến thuật toán phân kỳ.

Chia tỷ lệ đặc trưng (Feature Scalling): Khi sử dụng Gradient Descent, bạn nên bảo đảm rằng tất cả các đặc trưng có cùng tỷ lệ (ví dụ: sử dụng StandardScaler của Scikit-Learn). Nếu không, hàm mục tiêu sẽ có hình dạng một cái bát kéo dài, và thuật toán Gradient Descent sẽ mất nhiều thời gian hơn để hội tụ. Đối với các mô hình có chính quy hóa (regularization) như SVM, việc không chia tỷ lệ đặc trưng có thể dẫn đến một giải pháp dưới mức tối ưu.

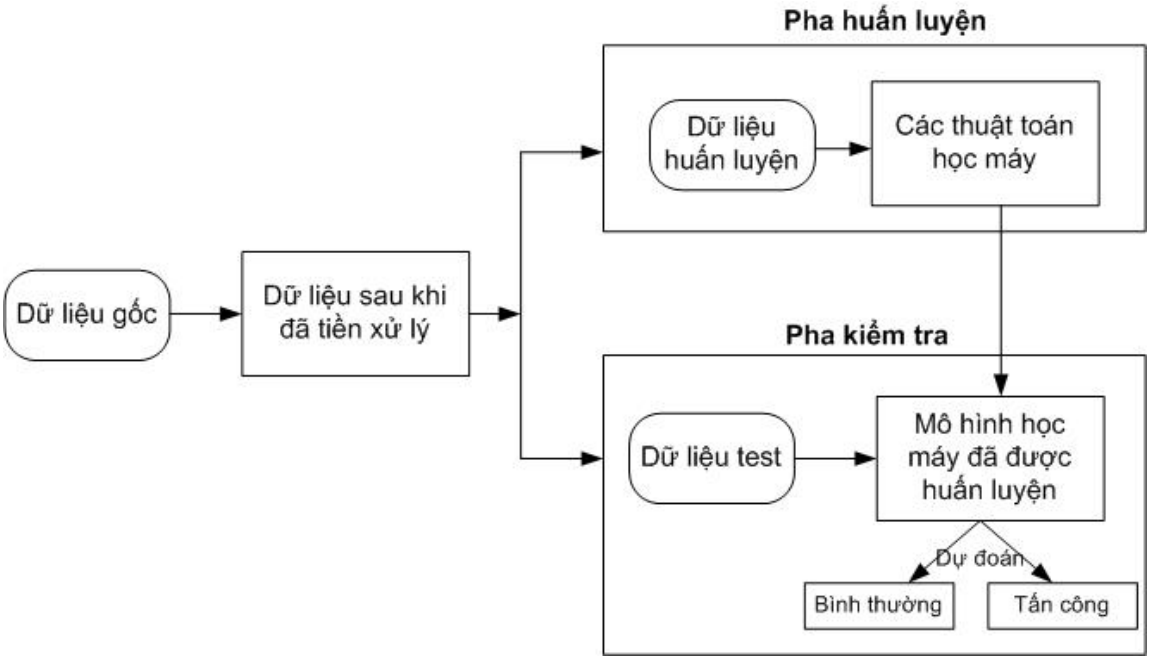
```
class SVM:

def __init__(self, learning_rate, no_of_iterations, lambda_parameter):
'''
learning_rate: Tốc độ học
no_of_iterations: Số vòng lặp
lambda_parameter: Tham số chính quy
model: Chứa các tham số weghts và bias
classes: Chứa các nhãn
binary_y: Chứa các nhãn nhị phân (1, -1)
w: Chứa các weghts
b: Chứa các bias
'''

    self.learning_rate = learning_rate
    self.no_of_iterations = no_of_iterations
    self.lambda_parameter = lambda_parameter
     self.models = []

# Huấn luyện mô hình SVM đa lớp dựa trên nhãn trong y
def fit(self, X, y):
  # Xác định các nhãn duy nhất từ y_train
   self.classes = np.unique(y)
   self.models = []

   # Huấn luyện một SVM nhị phân cho mỗi lớp (one-vs-all)
   for class_label in self.classes:
   # Tạo nhãn nhị phân cho one-vs-all: 1 cho nhãn hiện tại, -1 cho các nhãn
```

```
    khác
25      binary_y = np.where(y == class_label, 1, -1)

26      # Khởi weights và bias
27      w = np.zeros(X.shape[1])
28      b = 0

29      # Huấn luyện mô hình SVM nhị phân cho lớp hiện tại
30      for _ in range(self.no_of_iterations):
31          for index, x_i in enumerate(X):
32              condition = binary_y[index] * (np.dot(x_i, w) - b) >= 1

33              if condition:
                    dw = 2 * self.lambda_parameter * w
                    db = 0

34              else:
                    dw = 2 * self.lambda_parameter * w - x_i * binary_y[index]
                    db = binary_y[index]

35              # Cập nhật weights và bias
36              w -= self.learning_rate * dw
37              b -= self.learning_rate * db

38      # Lưu trọng số và bias cho mô hình của lớp hiện tại
39      self.models.append((w, b))

40 # Dự đoán nhãn cho mỗi mẫu đầu vào
41 def predict(self, X):
42     scores = np.zeros((X.shape[0], len(self.classes)))
43     # Tính điểm số dự đoán cho mỗi lớp
44     for i, (w, b) in enumerate(self.models):
45         scores[:, i] = np.dot(X, w) - b
46         # Trả về nhãn có điểm số cao nhất
47     return self.classes[np.argmax(scores, axis=1)]
```

3.6.Huấn luyện và kiểm thử mô hình

Sau khi đã xác định cơ sở lý thuyết về Support Vector Machine (SVM) và phương pháp tối ưu hóa bằng Gradient Descent, bước tiếp theo là áp dụng những kiến thức này để huấn luyện và kiểm thử mô hình nhận dạng chữ số. Quá trình này bao gồm việc đưa dữ liệu huấn luyện vào mô hình để nó học các tham số tối ưu, sau

đó sử dụng dữ liệu kiểm thử để đánh giá khả năng tổng quát hóa của mô hình.



Hình 3.12: Huấn luyện và kiểm thử mô hình

Mục tiêu của gia đoạn huấn luyện là tìm ra bộ tham số (vector trọng số **w** và hệ số chặn **b**) cho siêu phẳng phân chia tốt nhất, dựa trên hàm mục tiêu đã được định nghĩa và tối ưu hóa bằng Gradient Descent. Mô hình **model_svm** được khởi tạo và huấn luyện bằng cách gọi phương thức **fit**, truyền vào dữ liệu huấn luyện **X_train_array** và nhãn tương ứng **y_train_array**.

Trong quá trình huấn luyện, các siêu tham số (hyperparameters) đóng vai trò quan trọng trong việc điều khiển hành vị và hiệu suất của thuật toán tối ưu hóa. Dựa trên các giá trị được cung cấp trong mã nguồn.

Tốc độ học (learning_rate): Được đặt là 0.001. Tốc độ học (η) là một siêu tham số cực kỳ quan trọng trong Gradient Descent, quyết định kích thước của mỗi bước di chuyển xuống dọc theo gradient của hàm mục tiêu. Một tốc độ học phù hợp giúp thuật toán hội tụ hiệu quả đến cực tiểu toàn cục của hàm mục tiêu lồi của SVM. Nếu tốc độ học quá nhỏ, thuật toán sẽ hội tụ chậm, còn quá lớn thì có thể không hội tụ.

Số vòng lặp (no_of_iterations): Được đặt là 100. Đây là số lần thuật toán Gradient Descent sẽ lặp lại toàn bộ quá trình cập nhật tham số trên dữ liệu huấn luyện (tức là số epoch). Số vòng lặp đủ lớn là cần thiết để mô hình có thể hội tụ đến

một giải pháp tối ưu.

Tham số chính quy hóa (lambda_parameter): Được đặt là 0.0001. Tham số này kiểm soát mức độ chính quy hóa (regularization) được áp dụng cho mô hình. Trong ngữ cảnh của SVM, tham số chính quy hóa (thường được gọi là **C** trong thư viện Scikit-learn hoặc **alpha** trong SGDClassifier) giúp cân bằng giữa việc tối đa hóa biên độ (margin) và giảm thiểu các lỗi phân loại (margin violations). Một giá trị lambda_parameter nhỏ (như 0.0001) cho thấy mô hình được phép ít bị chính quy hóa hơn, tức là nó sẽ cố gắng phù hợp chặt chẽ hơn với dữ liệu huấn luyện bằng cách áp đặt một hình phạt nhỏ lên độ phức tạp của mô hình (như trọng số **w** lớn hơn) hoặc ít chấp nhận các lỗi vi phạm lề hơn. Điều này tương tự với việc sử dụng một giá trị **C** lớn trong SVC của Scikit-learn.

```
# Khởi tạo và huấn luyện mô hình SVC
'''
Các giá trị được điểm score cao nhất
learning_rate(Tốc độ học): 0.001
no_of_iterations(số vòng lặp): 100
lambda_parameter(tham số chính quy): 0.0001
'''

model_svm=SVM(learning_rate=0.001,no_of_iterations=100,
lambda_parameter=0.0001)

# Phân loại 10 lớp (các chữ số 0-9)
model_svm.fit(X_train_array, y_train_array)

# Dự đoán nhãn cho dữ liệu mới
y_pred = model_svm.predict(X_test_array)
print(y_pred[:5])  # Trả về nhãn dự đoán cho mỗi mẫu
```

Sau khi mô hình đã được huấn luyện, khả năng tổng quát hóa của nó được đánh giá bằng cách sử dụng tập dữ liệu kiểm thử (**test set**) mà mô hình chưa từng thấy trong quá trình huấn luyện. Phương thức predict của **model_svm** được sử dụng để dự đoán nhãn cho **X_test_array**, trả về một mảng **y_pred** chứa các nhãn dự đoán cho mỗi mẫu.

Ví dụ, **print(y_pred[:5])** sẽ hiển thị nhãn dự đoán cho 5 mẫu đầu tiên trong tập kiểm thử. Những dự đoán này sau đó sẽ được so sánh với nhãn thực tế (**y_test_array**) để đánh giá hiệu suất của mô hình.

3.7.Đánh giá hiệu quả mô hình

- **Độ chính xác (Accuracy)**

Độ chính xác là một trong những thước đo đơn giản và phổ biến nhất để đánh giá hiệu suất của mô hình phân loại. Nó được định nghĩa là tỷ lệ các mẫu được phân loại đúng so với tổng số mẫu trong tập dữ liệu kiểm thử.

Mã nguồn sử dụng hàm **accuracy_score** từ thư viện sklearn.metrics để tính toán độ chính xác:

```
1  from sklearn.metrics import accuracy_score

2  # Kiểm tra điểm score của y_test(nhãn test) và y_pred(nhãn dự đoán)
3  test_accuracy = accuracy_score(y_test_array, y_pred)
4  print(f"Test Accuracy: {test_accuracy*100:.3f}%")
```

```
Test Accuracy: 91.680%
```

Trong ngữ cảnh phân loại chữ số từ 0 đến 9, mô hình đạt độ chính xác là 91%, điều đó có nghĩa là 91% các chữ số trong tập dữ liệu kiểm thử đã được phân loại đúng.

Tuy nhiên, độ chính xác có thể gây hiểu lầm nếu tập dữ liệu không cân bằng (skewed datasets), tức là khi một số lớp xuất hiện thường xuyên hơn các lớp khác. Ví dụ, trong một bài toán phân loại nhị phân mà lớp "không phải số 5" chiếm 90% dữ liệu, một bộ phân loại luôn đoán là "không phải số 5" vẫn có thể đạt độ chính xác hơn 90%. Điều này cho thấy độ chính xác không phải lúc nào cũng là thước đo hiệu suất ưu tiên cho các bộ phân loại.

- **Ma trận nhầm lẫn (Confusion Matrix)**

Để có cái nhìn sâu sắc hơn về các lỗi mà mô hình mắc phải, ma trận nhầm lẫn là một cách đánh giá hiệu quả hơn. Ý tưởng chung là đếm số lần một trường hợp thuộc

lớp A được phân loại là lớp B. Đối với phân loại nhị phân, ma trận nhầm lẫn là một mảng hai chiều, trong đó các hàng tương ứng với các lớp thực tế và các cột tương ứng với các lớp dự đoán.

Các thành phần chính của ma trận nhầm lẫn bao gồm:

True Positives (TP): Số mẫu thuộc lớp dương được phân loại đúng là lớp dương.

True Negatives (TN): Số mẫu thuộc lớp âm được phân loại đúng là lớp âm.

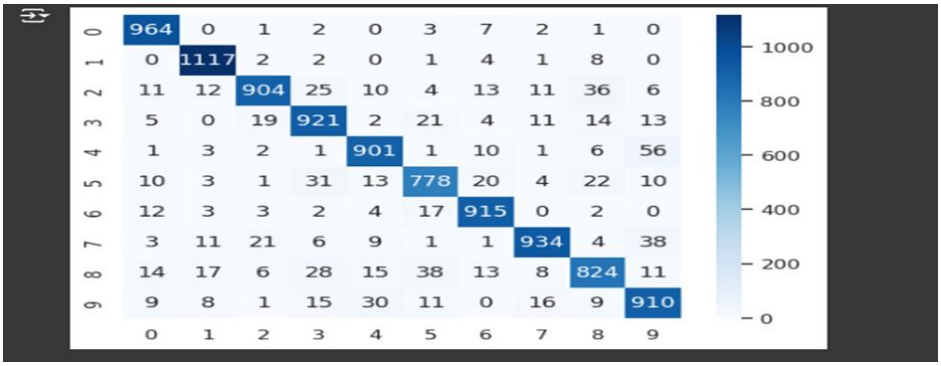
False Positives (FP): Số mẫu thuộc lớp âm bị phân loại sai là lớp dương (lỗi loại I).

False Negatives (FN): Số mẫu thuộc lớp dương bị phân loại sai là lớp âm (lỗi loại II).

Trong bài toán phân loại đa lớp (multi-class classification) như nhận dạng 10 chữ số (0-9), ma trận nhầm lẫn sẽ có kích thước $N \times N$ (ở đây là 10×10), với **N** là số lớp. Các mục trên đường chéo chính của ma trận nhầm lẫn cho biết số lượng phân loại đúng. Các mục khác cho biết số lượng mẫu của một lớp bị phân loại nhầm thành lớp khác. Một bộ phân loại hoàn hảo sẽ chỉ có các giá trị khác 0 trên đường chéo chính của ma trận nhầm lẫn.

Mã nguồn sử dụng **metrics.confusion_matrix** để tạo ma trận nhầm lẫn và **sns.heatmap** để trực quan hóa:

```
1  from sklearn import metrics

2  confusion = metrics.confusion_matrix(y_test_array, y_pred)
3  sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues')
4  plt.show()
```



Hình 3.13: Đánh giá mô hình bằng heatmap

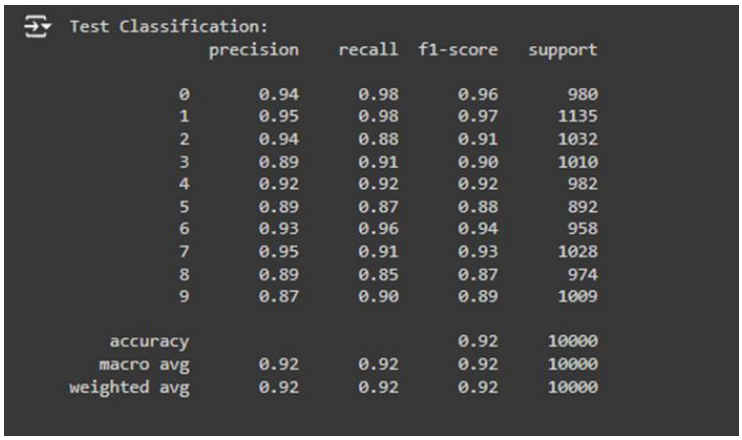
Việc trực quan hóa ma trận nhầm lẫn bằng heatmap (bản đồ nhiệt) giúp dễ dàng nhận biết các lỗi phân loại cụ thể mà mô hình thường mắc phải, ví dụ: liệu mô hình có thường nhầm lẫn số "7" với số "1" hay không.

- **Báo cáo phân loại (Classification Report)**

Để tóm tắt thông tin từ ma trận nhầm lẫn một cách súc tích hơn, đặc biệt là các chỉ số như Precision, Recall và F1-score cho từng lớp, hàm classification_report là một công cụ tiện lợi.

Mã nguồn tạo báo cáo phân loại:

```
1  class_wise = metrics.classification_report(y_test_array, y_pred)
2  print(f"Test Classification: \n{class_wise}")
```



Hình 3.14: Đánh giá mô hình bằng f1-score

Báo cáo này cung cấp các chỉ số sau cho mỗi lớp:

Precision (Độ chính xác): Đo lường tỷ lệ các mẫu được dự đoán là thuộc một lớp cụ thể mà thực sự thuộc lớp đó. Công thức: **Precision = TP / (TP + FP)**. Ví dụ, nếu

mô hình dự đoán 100 hình ảnh là số "5" và 72 trong số đó thực sự là số "5", thì độ chính xác là 72%.

Recall (Độ phủ/Độ nhạy): Đo lường tỷ lệ các mẫu thực sự thuộc một lớp cụ thể mà mô hình đã xác định đúng. Công thức: **Recall = TP / (TP + FN)**. Ví dụ, nếu có tổng cộng 100 hình ảnh là số "5" trong tập kiểm thử và mô hình phát hiện được 75 trong số đó, thì độ phủ là 75%.

F1-score: Là trung bình điều hòa của Precision và Recall. Nó cung cấp một thước đo tổng hợp, đặc biệt hữu ích khi cần cân bằng giữa Precision và Recall. Công thức: $F1 = 2 \times (Precision \times Recall)/(Precision + Recall)$. F1-score sẽ cao chỉ khi cả Precision và Recall đều cao.

Support: Là số lượng mẫu thực tế thuộc về mỗi lớp trong tập kiểm thử.

Trong phân loại đa lớp, F1-score có thể được tính theo các chiến lược trung bình khác nhau:

"macro" average: Tính trung bình F1-score của mỗi lớp mà không tính đến kích thước của lớp đó. Nó coi tất cả các lớp có trọng số bằng nhau.

"weighted" average: Tính trung bình F1-score của mỗi lớp, có trọng số theo số lượng mẫu thực tế trong lớp đó (support).

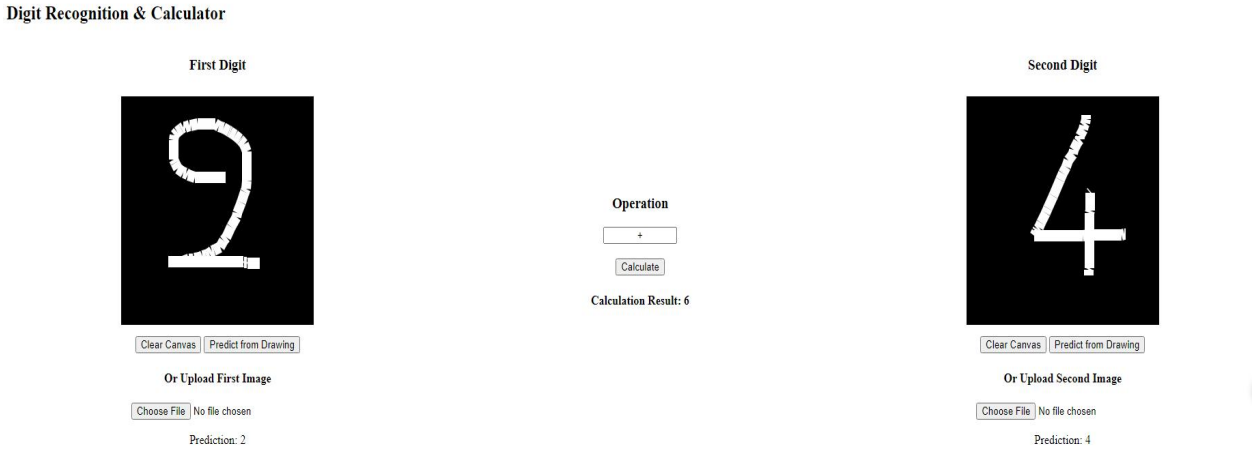
"micro" average: Tính tổng TP, FP, FN trên tất cả các lớp, sau đó tính Precision, Recall và F1-score dựa trên các tổng đó. Phương pháp này phù hợp nếu bạn quan tâm đến mỗi mẫu như nhau.

Việc phân tích báo cáo phân loại giúp hiểu rõ hơn về hiệu suất của mô hình trên từng lớp riêng biệt và là một chỉ số quan trọng để đánh giá mô hình, đặc biệt là trong các trường hợp dữ liệu không cân bằng, nơi độ chính xác đơn thuần có thể không phản ánh đúng hiệu quả.

CHƯƠNG 4: TRIỂN KHAI VÀ ỨNG DỤNG

4.1.Xây dựng giao diện người dùng

Khu vực vẽ chữ số (Canvas): Đây là thành phần tương tác cốt lõi, nơi người dùng có thể vẽ trực tiếp các chữ số bằng chuột hoặc thiết bị cảm ứng. Để đảm bảo dữ liệu đầu vào phù hợp với mô hình SVM, hình ảnh được vẽ trên canvas sẽ trải qua quy trình tiền xử lý tự động ở backend.



Hình 4.1: Giao diện người dùng

Các nút điều khiển Canvas:

"Clear Canvas": Nút này cho phép người dùng xóa bỏ bản vẽ hiện tại trên canvas, chuẩn bị cho việc vẽ một chữ số mới.

"Predict from Drawing": Khi người dùng hoàn tất việc vẽ một chữ số, nút này sẽ kích hoạt quá trình gửi dữ liệu hình ảnh đã vẽ (và được tiền xử lý) đến backend để mô hình SVM thực hiện dự đoán.

Chức năng tải ảnh lên (Image Upload): Ngoài tùy chọn vẽ, giao diện còn cung cấp khả năng cho người dùng tải lên các tệp hình ảnh chứa chữ số từ thiết bị của họ (ví dụ: "Or Upload First Image", "Or Upload Second Image"). Các hình ảnh được tải lên cũng sẽ trải qua quy trình tiền xử lý tương tự như dữ liệu từ canvas (process_image) trước khi được chuyển đến mô hình để dự đoán.

Khu vực hiển thị kết quả dự đoán:

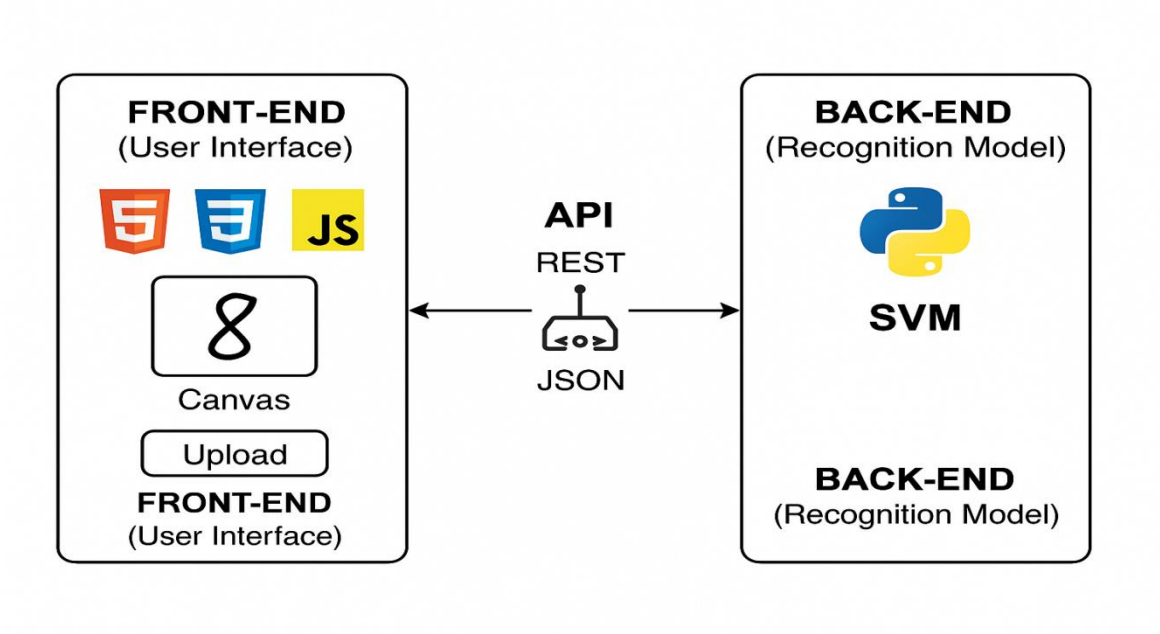
"Prediction: N/A": Hiển thị chữ số được mô hình SVM dự đoán từ dữ liệu đầu

vào.

"Prediction Time: N/A": Cho biết thời gian (tính bằng mili giây hoặc giây) mà mô hình cần để xử lý và đưa ra dự đoán.

4.2.Kết nối front-end với mô hình nhận dạng

Quá trình kết nối front-end với mô hình nhận dạng chữ số được thực hiện thông qua API (Application Programming Interface), cho phép hai thành phần này giao tiếp với nhau. Kiến trúc này đảm bảo tính linh hoạt, khả năng mở rộng và dễ dàng bảo trì hệ thống.



Hình 4.2: Kết nối front-end với mô hình nhận dạng

4.2.1.Kiến trúc giao tiếp

Front-end (Giao diện người dùng): Được xây dựng bằng các công nghệ web cơ bản như HTML, CSS và JavaScript. Đây là nơi người dùng tương tác trực tiếp, bao gồm các khu vực để vẽ chữ số trên canvas và các nút để tải ảnh lên.

Back-end (Mô hình nhận dạng): Chứa mô hình SVM đã được huấn luyện và các logic tiền xử lý dữ liệu. Back-end thường được triển khai bằng Python cùng với một framework web như Flask (thường được gợi ý cho các tuyến API như app.route) để xử lý các yêu cầu từ front-end. Các thư viện như sklearn được sử dụng để triển khai SVM.

API: Đóng vai trò là cầu nối. Thông thường, một REST API là lựa chọn phổ biến, cho phép front-end gửi các yêu cầu HTTP (ví dụ: POST requests) và nhận phản hồi dưới dạng JSON.

4.2.2.Quy trình xử lý dữ liệu từ giao diện

Quá trình kết nối diễn ra theo các bước sau, tùy thuộc vào phương thức nhập liệu của người dùng:

- **Nhận dạng chữ số vẽ trên Canvas:**

Nhập liệu từ người dùng: Người dùng vẽ một chữ số trên khu vực canvas được cung cấp trên giao diện.

Kích hoạt yêu cầu dự đoán: Khi người dùng nhấn nút "Predict from Drawing", mã JavaScript ở front-end sẽ thực hiện các bước sau:

Thu thập dữ liệu Canvas: Nội dung hình ảnh được vẽ trên canvas sẽ được trích xuất, thường dưới dạng base64 encoding của hình ảnh (ví dụ: PNG).

Gửi yêu cầu API: Dữ liệu hình ảnh này được đóng gói vào một yêu cầu HTTP POST và gửi đến một endpoint cụ thể trên back-end (ví dụ: /predict_drawing). Yêu cầu này cũng có thể bao gồm các thông tin khác nếu cần.

Xử lý tại Back-end:

Nhận yêu cầu: Back-end nhận yêu cầu HTTP chứa dữ liệu hình ảnh.

Tiền xử lý hình ảnh: Dữ liệu hình ảnh thô được giải mã và chuyển đổi. Các bước tiền xử lý cần thiết để chuẩn bị dữ liệu cho mô hình SVM bao gồm:

Thay đổi kích thước: Chuyển đổi hình ảnh về kích thước chuẩn (ví dụ: 28x28 pixel), tương tự như định dạng của bộ dữ liệu MNIST.

Chuyển đổi sang ảnh xám: Chuyển đổi hình ảnh màu (nếu có) hoặc hình ảnh đen trắng về định dạng ảnh xám (.convert('L')).

Chuẩn hóa pixel: Chuẩn hóa giá trị pixel về một dải nhất định (ví dụ: từ 0-255 về 0.0-1.0 bằng cách chia cho 255 và chuyển sang float32).

Làm phẳng (Flatten): Chuyển đổi ma trận 2D (ví dụ: 28x28) thành một mảng 1D (ví dụ: 784 phần tử) để phù hợp với định dạng đầu vào của mô hình SVM.

Dự đoán bằng mô hình SVM: Dữ liệu đã tiền xử lý được đưa vào mô hình SVM đã huấn luyện. Support Vector Machines (SVMs) là một tập hợp các phương pháp học giám sát được sử dụng cho phân loại. SVM hoạt động bằng cách tìm một "siêu mặt phẳng" (hyper-plane) chia dữ liệu thành các lớp, với mục tiêu là cực đại hóa khoảng cách (margin) từ siêu mặt phẳng đến các điểm dữ liệu gần nhất (support vectors). Thư viện sklearn.svm.SVC có thể được sử dụng để tìm nghiệm cho SVM.

Ghi lại thời gian dự đoán: Thời gian cần thiết để mô hình đưa ra dự đoán cũng được tính toán để cung cấp phản hồi cho người dùng.

Gửi phản hồi: Kết quả dự đoán (chữ số) và thời gian xử lý được đóng gói vào một phản hồi JSON và gửi về front-end.

Hiển thị kết quả tại Front-end:

Giao diện người dùng nhận phản hồi từ back-end.

Các trường hiển thị trên giao diện như "Prediction: N/A" và "Prediction Time: N/A" sẽ được cập nhật với kết quả nhận dạng và thời gian tương ứng.

Xóa Canvas: Nút "Clear Canvas" cho phép người dùng xóa bản vẽ hiện tại để chuẩn bị cho một chữ số mới.

- **Nhận dạng chữ số từ Ảnh tải lên:**

Tải ảnh lên: Người dùng sử dụng chức năng "Or Upload First Image" hoặc "Or Upload Second Image" để tải một tệp hình ảnh từ thiết bị của họ.

Gửi yêu cầu API: Tệp hình ảnh được gửi đến back-end thông qua một yêu cầu HTTP POST (ví dụ: /upload_image).

Xử lý tại Back-end:

Nhận và lưu trữ tạm thời: Back-end nhận tệp hình ảnh.

Tiền xử lý hình ảnh: Tương tự như với dữ liệu từ canvas, tệp hình ảnh tải lên cũng sẽ trải qua các bước tiền xử lý đã mô tả ở trên (thay đổi kích thước, chuyển đổi xám, chuẩn hóa, làm phẳng).

Dự đoán và phản hồi: Dữ liệu đã tiền xử lý được đưa vào mô hình SVM để dự đoán. Kết quả dự đoán và thời gian xử lý được gửi về front-end thông qua phản hồi JSON.

Hiển thị kết quả tại Front-end: Kết quả nhận dạng và thời gian xử lý được hiển thị trên giao diện người dùng tương tự như khi vẽ trên canvas.

CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1.Kết luận

Dự án xây dựng hệ thống nhận dạng chữ số viết tay sử dụng bộ dữ liệu MNIST đã thành công trong việc phát triển một mô hình học máy có khả năng nhận diện chính xác các chữ số từ 0 đến 9. Quá trình phát triển hệ thống đã bao gồm các bước quan trọng như thu thập dữ liệu, tiền xử lý hình ảnh, xây dựng và huấn luyện mô hình học máy, và đánh giá hiệu suất của mô hình thông qua các chỉ số như độ chính xác, độ phủ, độ nhạy và điểm F1. Mô hình SVM đã được chứng minh là hiệu quả trong việc phân loại các chữ số viết tay, mang lại kết quả nhận dạng chính xác cho hầu hết các mẫu thử nghiệm.

Bên cạnh đó, hệ thống cũng đã triển khai giao diện người dùng (UI) cho phép người dùng tải lên hoặc vẽ các chữ số trực tiếp trên màn hình. Hệ thống không chỉ có khả năng nhận dạng chữ số mà còn hỗ trợ thực hiện các phép toán cơ bản như cộng, trừ, nhân và chia, qua đó mở rộng ứng dụng của hệ thống vào các lĩnh vực thực tế như xử lý dữ liệu từ các mẫu giấy tờ viết tay, hóa đơn hoặc phiếu khảo sát.

5.2.Hướng phát triển

Cải thiện độ chính xác của mô hình: Mặc dù mô hình đã cho kết quả tốt, nhưng có thể cải thiện độ chính xác hơn nữa bằng cách áp dụng các kỹ thuật học sâu (Deep Learning) như Mạng Nơ-ron Tích chập (CNN), vốn thường cho kết quả tốt hơn trong các bài toán nhận diện hình ảnh. Việc sử dụng các phương pháp học sâu sẽ giúp hệ thống nhận diện chữ số viết tay phức tạp hơn với độ chính xác cao.

Mở rộng ứng dụng: Ngoài việc nhận dạng các chữ số, hệ thống có thể được mở rộng để nhận diện các ký tự chữ cái, phục vụ cho các ứng dụng trong lĩnh vực OCR (Nhận diện ký tự quang học). Điều này sẽ giúp hệ thống có thể được ứng dụng rộng rãi hơn, chẳng hạn như trong nhận dạng chữ viết tay của người dùng hoặc trong các công việc tự động hóa nhập liệu.

Xử lý dữ liệu không chuẩn: Một hướng phát triển quan trọng là mở rộng khả

năng xử lý các hình ảnh viết tay không chuẩn, như chữ viết lệch, méo mó hoặc chữ viết có yếu tố nhiễu lớn. Cải thiện khả năng nhận diện cho những trường hợp này sẽ làm cho hệ thống trở nên mạnh mẽ hơn khi đối mặt với các tình huống thực tế.

Cải thiện giao diện người dùng: Giao diện người dùng có thể được phát triển thêm với các tính năng như nhận diện chữ số trong ảnh tải lên từ camera, hỗ trợ người dùng trong việc vẽ hoặc nhập chữ số một cách dễ dàng hơn. Ngoài ra, các phép toán phức tạp hơn như căn bậc hai, lũy thừa cũng có thể được tích hợp vào hệ thống.

Triển khai hệ thống thực tế: Đưa hệ thống vào sử dụng trong các ứng dụng thực tế như thanh toán điện tử, nhận dạng số trong phiếu khảo sát, hay các công việc nhập liệu tự động. Việc tích hợp hệ thống vào các nền tảng web hoặc di động sẽ làm cho hệ thống trở nên linh hoạt và tiện ích hơn đối với người dùng.

TÀI LIỆU THAM KHẢO

[1] Zhou, Z. H. (2021). *Machine learning*. Springer nature.

[2] Alpaydin, E. (2021). *Machine learning*. MIT press.

[3] Mitchell, T. M., & Mitchell, T. M. (1997). *Machine learning* (Vol. 1, No. 9). New York: McGraw-hill.

[4] Mammone, A., Turchi, M., & Cristianini, N. (2009). Support vector machines. Wiley Interdisciplinary Reviews: Computational Statistics, 1(3), 283-289.

[5] Noble, W. S. (2006). What is a support vector machine?. Nature biotechnology, 24(12), 1565-1567.

[6] Cohen, G., Afshar, S., Tapson, J., & Van Schaik, A. (2017, May). EMNIST: Extending MNIST to handwritten letters. In 2017 international joint conference on neural networks (IJCNN) (pp. 2921-2926). IEEE.

[7] Baldominos, A., Saez, Y., & Isasi, P. (2019). A survey of handwritten character recognition with mnist and emnist. Applied Sciences, 9(15), 3169.