**HỌC VIỆN NÔNG NGHIỆP VIỆT NAM**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

**KHOÁ LUẬN TỐT NGHIỆP**

**Đề tài: PHÂN LỚP DỮ LIỆU DỰ ĐOÁN LỖI PHẦN MỀM**

**DỰA TRÊN THUẬT TOÁN SVM**

* **Giảng viên hướng dẫn:** ThS. Nguyễn Xuân Thảo
* **Người thực hiện:** Nguyễn Thị Duyên – 637614
* Lớp: K63HTTT

**HÀ NỘI, NĂM 2024**

**LỜI CẢM ƠN**

Em xin chân thành cảm ơn thầy Nguyễn Xuân Thảo đã tận tình giúp đỡ em hoàn thành Khoá luận tốt nghiệp này. Để hoàn thành chuyên đề này, em đã nỗ lực thực hiện và đồng thời cũng nhận được nhiều sự giúp đỡ từ thầy và khoa CNTT. Em cũng xin gửi lời cảm ơn đến Học viện Nông Nghiệp Việt Nam đã tạo điều kiện tốt nhất để em có thể học tập, trao đổi và nâng cao kiến thức của mình. Mặc dù em đã rất cố gắng hoàn thành trong phạm vi và khả năng cho phép nhưng chắc chắn sẽ không tránh khỏi những thiếu sót. Em kính mong nhận được sự cảm thông và tận tình chỉ bảo của thầy.

Em xin chân thành cảm ơn sự hướng dẫn và chỉ bảo tận tình của thầy!

Hà Nội*, ngày … tháng … năm 2024*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

**CÔNG TRÌNH ĐƯỢC HOÀN THÀNH**

**TẠI HỌC VIỆN NÔNG NGHIỆP VIỆT NAM**

Em xin cam đoan đây là công trình nghiên cứu của riêng em và được sự hướng dẫn khoa học của ThS Nguyễn Xuân Thảo. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong chuyên đề còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào em xin hoàn toàn chịu trách nhiệm về nội dung Chuyên đề của mình**. Học viện Nông Nghiệp Việt Nam không liên quan đến những vi phạm tác quyền, bản quyền do em gây ra trong quá trình thực hiện (nếu có).

*Hà Nội, ngày … tháng … năm 2024*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

**PHÁT HIỆN LỖI SẢN PHẨM**

**TÓM TẮT**

Chuyên đề nghiên cứu em làm dưới đây liên quan đến vấn đề phát hiện lỗi sản phẩm phần mềm, một vấn đề quan trọng trong đảm bảo chất lượng và độ tin cậy của hệ thống phần mềm. Trong bài nghiên cứu này, em đã tìm hiểu về các loại lỗi phần mềm, các phương pháp để phát hiện và phân loại lỗi, và vai trò của học máy trong quá trình này. Em đã tìm hiểu về mô hình học máy: Máy Vector Hỗ trợ (SVM), và cách áp dụng mô hình để phát hiện lỗi phần mềm. Cuối cùng là triển khai phương pháp học máy SVM vào bộ dữ liệu để đưa ra kết quả, đánh giá hiệu suất của mô hình.

**MỤC LỤC**

[**DANH MỤC HÌNH VẼ** 9](#_Toc169423381)

[**DANH MỤC CÁC CHỮ VIẾT TẮT** 9](#_Toc169423382)

[**CHƯƠNG 1.** **MỞ ĐẦU** 1](#_Toc169423383)

[**1.1** **Tên đề tài:** 1](#_Toc169423384)

[**1.2** **Lý do chọn đề tài** 1](#_Toc169423385)

[**1.3** **Mục tiêu thực hiện đề tài** 2](#_Toc169423386)

[**CHƯƠNG 2.** TỔNG QUAN TÌNH HÌNH NGHIÊN CỨU TRONG VÀ NGOÀI NƯỚC 3](#_Toc169423387)

[2.1 Tình hình nghiên cứu trong nước 3](#_Toc169423388)

[2.2 Tình hình nghiên cứu ngoài nước 3](#_Toc169423389)

[2.3 Tính thời sự, tầm quan trọng của đề tài 4](#_Toc169423390)

[**CHƯƠNG 3. PHƯƠNG PHÁP NGHIÊN CỨU** 5](#_Toc169423391)

[3.1 Địa điểm và Thời gian Nghiên cứu (Thực tập) 5](#_Toc169423392)

[3.2 Nội dung Nghiên cứu 6](#_Toc169423393)

[3.2.2. Khảo sát Bài toán 6](#_Toc169423394)

[3.2.3. Tìm hiểu Thư viện và Cơ chế Hoạt động của Thuật toán 6](#_Toc169423395)

[3.2.4. Xây dựng Thuật toán 6](#_Toc169423396)

[3.2.5. Hoàn thiện Thuật toán 6](#_Toc169423397)

[3.3 Phương pháp Nghiên cứu 7](#_Toc169423398)

[3.3.1 Nghiên cứu Lý thuyết 7](#_Toc169423399)

[3.3.2 Thực nghiệm 7](#_Toc169423400)

[**CHƯƠNG 4. CƠ SỞ LÝ THUYẾT** 8](#_Toc169423401)

[4.1 Các mục tiêu chính của đảm bảo chất lượng phần mềm 9](#_Toc169423402)

[4.2 Các giải pháp đảm bảo chất lượng phần mềm 9](#_Toc169423403)

[4.2.1 Kiểm thử tự động (Automated Testing): 9](#_Toc169423404)

[4.2.2 Phân tích mã nguồn tĩnh (Static Code Analysis): 9](#_Toc169423405)

[4.2.3 Kiểm soát chất lượng phần mềm (Software Quality Control): 9](#_Toc169423406)

[4.3 Các phương pháp phát hiện lỗi phần mềm 9](#_Toc169423407)

[4.3.1 Hệ thống phát hiện lỗi dựa trên mạng (Network-based EDS): 9](#_Toc169423408)

[4.3.2 Hệ thống phát hiện lỗi dựa trên máy chủ (Host-based EDS): 9](#_Toc169423409)

[4.3.3 Hệ thống phát hiện lỗi kết hợp (Hybrid EDS): 9](#_Toc169423410)

[4.3.4 Cơ sở dữ liệu trong phát hiện lỗi phần mềm 9](#_Toc169423411)

[4.4 Mô hình học máy và học sâu trong phát hiện lỗi phần mềm 9](#_Toc169423412)

[4.4.1 Học máy trong phát hiện lỗi phần mềm: 9](#_Toc169423413)

[4.4.2 Học sâu trong phát hiện lỗi phần mềm: 9](#_Toc169423414)

[4.5 Tập dữ liệu và đánh giá mô hình phát hiện lỗi phần mềm 9](#_Toc169423415)

[4.6 Các thách thức trong đánh giá mô hình phát hiện lỗi phần mềm 9](#_Toc169423416)

[**CHƯƠNG 5. MÔ HÌNH ĐỀ XUẤT** 9](#_Toc169423417)

[5.0 Dữ liệu sử dụng 10](#_Toc169423418)

[5.1 Phân tích khám phá dữ liệu và xử lý dữ liệu 10](#_Toc169423419)

[4.1.1 Xử lý dữ liệu rỗng và dữ liệu dạng chuỗi 11](#_Toc169423420)

[4.1.2 Chọn lọc đặc trưng đơn giản 11](#_Toc169423421)

[4.1.3 Đánh giá phân phối lớp 11](#_Toc169423422)

[4.1.4 Chia dữ liệu và chuẩn hóa 11](#_Toc169423423)

[5.2 Chọn và Huấn luyện Mô hình 10](#_Toc169423424)

[5.2.1 Tại sao chọn model SVM 10](#_Toc169423425)

[5.2.2 Tổng Quan về Support Vector Machines (SVM) 10](#_Toc169423426)

[5.2.3 Các Thành Phần Chính của SVM 10](#_Toc169423427)

[5.2.4 Phương Pháp Làm Việc của SVM 10](#_Toc169423428)

[5.3 Đánh giá mô hình ban đầu – SMOTE – LỰA CHỌN ĐẶC TRƯNG 10](#_Toc169423429)

[4.3.1 Sử dụng SMOTE để cân bằng số lượng mẫu giữa các lớp 11](#_Toc169423430)

[4.3.2 Feature Selection (Lựa chọn đặc trưng) 11](#_Toc169423431)

[5.4 Điều chỉnh siêu tham số (Tuning Model) 10](#_Toc169423432)

[5.5 Cách thức đánh giá hiệu suất mô hình 12](#_Toc169423433)

[5.5.1 Accuracy: Độ Chính xác 13](#_Toc169423434)

[5.5.2 Precision: Độ Chính xác tích cực 13](#_Toc169423435)

[5.5.3 Recall: Độ Nhạy 13](#_Toc169423436)

[5.5.4 F1 Score: Điểm số F1 13](#_Toc169423437)

[5.6 Demo 14](#_Toc169423438)

[**CHƯƠNG 6. KẾT QUẢ THỰC NGHIỆM** 16](#_Toc169423439)

[6.1 Kết quả thực nghiệm 16](#_Toc169423440)

[6.1.1 Trước khi Tuning Model 16](#_Toc169423441)

[6.1.2 Sau khi tiến hành tinh chỉnh tham số: 17](#_Toc169423442)

[6.1.3 Nhận xét chi tiết 19](#_Toc169423443)

[6.2 Kết luận 19](#_Toc169423444)

[CHƯƠNG 7. Hướng phát triển 20](#_Toc169423445)

[7.1 Tích hợp thêm yếu tố và nguồn dữ liệu đa dạng 20](#_Toc169423446)

[7.2 Khám phá các kỹ thuật học máy mới và tiên tiến 20](#_Toc169423447)

[7.3 Tối ưu hóa và cập nhật mô hình dựa trên phản hồi thực tế 21](#_Toc169423448)

[7.4 Phát triển giao diện người dùng trực quan: 21](#_Toc169423449)

[**TÀI LIỆU THAM KHẢO** 22](#_Toc169423450)

# **DANH MỤC HÌNH VẼ**

# **DANH MỤC CÁC CHỮ VIẾT TẮT**

|  |  |
| --- | --- |
|  |  |

# **MỞ ĐẦU**

## **Tên đề tài:**

Phân lớp dữ liệu dự đoán lỗi phần mềm dựa trên thuật toán SVM

## **Lý do chọn đề tài**

Trong kỷ nguyên kỹ thuật số hiện nay, đảm bảo chất lượng và độ tin cậy của phần mềm trở thành một ưu tiên hàng đầu cho các doanh nghiệp và tổ chức trên toàn thế giới. Sự phức tạp ngày càng tăng của các hệ thống phần mềm đòi hỏi những phương pháp tiên tiến để phát hiện và phân loại lỗi phần mềm một cách nhanh chóng và chính xác. Điều này không chỉ giúp giảm thiểu chi phí sửa lỗi mà còn nâng cao trải nghiệm người dùng và đảm bảo an toàn thông tin.

Các kỹ thuật Học Máy (Machine Learning) đã chứng tỏ được vai trò quan trọng trong việc giải quyết nhiều vấn đề phức tạp, từ phân tích dữ liệu lớn đến nhận dạng mẫu và dự đoán xu hướng. Trong lĩnh vực kiểm thử phần mềm, việc ứng dụng các kỹ thuật học máy, đặc biệt là Máy Vector Hỗ trợ (Support Vector Machine - SVM), hứa hẹn sẽ mang lại khả năng phát hiện sớm các lỗi phần mềm, từ đó giúp giảm thiểu thiệt hại và tăng cường độ tin cậy của hệ thống.

Với tiềm năng lớn mà SVM mang lại, đề tài "Phân loại lỗi phần mềm bằng SVM" được chọn làm đề tài Khoá luận tốt nghiệp, nhằm mục tiêu nghiên cứu và ứng dụng kỹ thuật SVM vào việc phát hiện và phân loại lỗi phần mềm. Đề tài này không chỉ có ý nghĩa trong việc cải thiện chất lượng phần mềm và bảo vệ thông tin mà còn đóng góp vào việc phát triển các phương pháp kiểm thử hiệu quả, giảm thiểu rủi ro và tối ưu hóa quy trình quản lý chất lượng phần mềm. Qua đó, đề tài hỗ trợ việc ra quyết định và chiến lược phát triển phần mềm tốt hơn.

Thông qua việc nghiên cứu và ứng dụng SVM trong phân loại lỗi phần mềm, khoá luận này hy vọng sẽ cung cấp những đóng góp đáng kể vào lĩnh vực kiểm thử phần mềm và mở ra những hướng đi mới trong việc ứng dụng học máy vào thực tiễn sản xuất và phát triển phần mềm.

## **Mục tiêu thực hiện đề tài**

Mục tiêu của đề tài này bao gồm:

- Nắm vững cơ sở lý thuyết của kỹ thuật SVM áp dụng trong phân loại lỗi phần mềm, bao gồm nguyên lý hoạt động, phương pháp phân loại và nhận dạng các lỗi phần mềm khác nhau.

- Nghiên cứu và so sánh hiệu quả của mô hình SVM trong việc phân loại lỗi phần mềm, nhằm đề xuất mô hình tối ưu có khả năng phân biệt giữa mã nguồn chứa lỗi và mã nguồn không chứa lỗi, bao gồm cả các lỗi logic, lỗi cú pháp, và các hình thức lỗi phần mềm khác.

- Phân tích và đánh giá hiệu suất của mô hình SVM được phát triển thông qua thử nghiệm trên bộ dữ liệu thực tế, để xác định khả năng phát hiện lỗi và độ chính xác trong các tình huống khác nhau.

- Khám phá và phân tích các ứng dụng thực tiễn của kỹ thuật SVM trong lĩnh vực kiểm thử phần mềm, như phân loại lỗi, phân tích mã nguồn và tối ưu hóa việc phát hiện lỗi phần mềm.

Mục tiêu chính của đề tài này là không chỉ tìm ra một mô hình SVM hiệu quả trong việc phân loại lỗi phần mềm mà còn góp phần vào việc nâng cao nhận thức và kiến thức về chất lượng phần mềm. Qua đó, đề tài nhấn mạnh sự cần thiết của việc áp dụng kỹ thuật SVM vào việc đảm bảo chất lượng phần mềm trước những thách thức ngày càng phức tạp trong phát triển và bảo trì phần mềm.

# **CHƯƠNG 2.** TỔNG QUAN TÌNH HÌNH NGHIÊN CỨU TRONG VÀ NGOÀI NƯỚC

## 2.1 Tình hình nghiên cứu trong nước

Việc phát hiện và phân loại lỗi phần mềm là một lĩnh vực quan trọng trong ngành công nghệ thông tin, đặc biệt khi phần mềm ngày càng trở nên phức tạp và quan trọng trong đời sống hiện đại. Ở Việt Nam, các nghiên cứu về phát hiện và phân loại lỗi phần mềm đã bắt đầu thu hút sự quan tâm từ các nhà khoa học và các trường đại học.

Một trong những nghiên cứu đáng chú ý là của nhóm tác giả Trần Văn Bình và cộng sự tại Đại học Bách Khoa Hà Nội. Trong nghiên cứu này, nhóm tác giả đã ứng dụng thuật toán Support Vector Machine (SVM) để phân loại lỗi phần mềm trong các dự án phần mềm mã nguồn mở. Kết quả cho thấy SVM có thể phát hiện lỗi phần mềm với độ chính xác cao, đặc biệt là khi kết hợp với các phương pháp tiền xử lý dữ liệu và lựa chọn đặc trưng.

Ngoài ra, nhóm nghiên cứu tại Viện Công nghệ Thông tin thuộc Đại học Quốc gia Hà Nội cũng đã tiến hành nghiên cứu về việc áp dụng SVM để phát hiện lỗi phần mềm. Nghiên cứu này tập trung vào việc sử dụng SVM để phân loại các lỗi thường gặp trong quá trình phát triển phần mềm, từ đó đề xuất các biện pháp khắc phục kịp thời. Kết quả nghiên cứu cho thấy SVM có khả năng phát hiện và phân loại lỗi phần mềm với độ chính xác đáng kể.

Những nghiên cứu này đã góp phần nâng cao hiệu quả trong việc phát hiện và phân loại lỗi phần mềm, giúp các doanh nghiệp và tổ chức có thể cải thiện chất lượng phần mềm của mình. Tuy nhiên, vẫn còn nhiều thách thức cần phải giải quyết, đặc biệt là việc tối ưu hóa các mô hình SVM để đạt được hiệu quả cao nhất.

## 2.2 Tình hình nghiên cứu ngoài nước

Trên thế giới, việc ứng dụng thuật toán SVM trong phát hiện và phân loại lỗi phần mềm đã được nghiên cứu rộng rãi và đạt được nhiều thành tựu đáng kể. SVM là một kỹ thuật học máy mạnh mẽ, có khả năng phân loại dữ liệu hiệu quả bằng cách tìm ra một siêu phẳng tối ưu để phân tách các lớp dữ liệu.

Một nghiên cứu nổi bật là của nhóm tác giả Kim và cộng sự (2019) tại Đại học Stanford, Mỹ. Nhóm tác giả đã sử dụng SVM để phát hiện lỗi phần mềm trong các dự án mã nguồn mở lớn như Apache và Mozilla. Kết quả cho thấy SVM có thể phát hiện lỗi phần mềm với độ chính xác cao, đặc biệt là khi được kết hợp với các phương pháp xử lý dữ liệu như SMOTE (Synthetic Minority Over-sampling Technique) để cân bằng dữ liệu.

Một nghiên cứu khác của nhóm tác giả Wang và cộng sự (2020) tại Đại học Công nghệ Sydney, Úc, đã áp dụng SVM để phân loại lỗi phần mềm trong các hệ thống lớn. Nhóm tác giả đã sử dụng các đặc trưng về mã nguồn và thông tin về lịch sử phát triển phần mềm để huấn luyện mô hình SVM. Kết quả cho thấy mô hình SVM có thể phát hiện lỗi phần mềm với độ chính xác lên tới 92%, giúp giảm thiểu đáng kể chi phí sửa lỗi và nâng cao chất lượng phần mềm.

Những nghiên cứu này cho thấy tiềm năng lớn của SVM trong việc phát hiện và phân loại lỗi phần mềm, đồng thời cũng đề xuất các phương pháp mới để tối ưu hóa hiệu quả của mô hình. Tuy nhiên, việc áp dụng SVM vào thực tiễn vẫn còn nhiều thách thức, đặc biệt là việc xử lý dữ liệu không cân bằng và tối ưu hóa các siêu tham số của mô hình.

## 2.3 Tính thời sự, tầm quan trọng của đề tài

Tên đề tài: Phân lớp dữ liệu dự đoán lỗi phần mềm dựa trên thuật toán SVM

Tính thời sự của đề tài ứng dụng thuật toán SVM trong phát hiện và phân loại lỗi phần mềm là rất cao. Trong bối cảnh các hệ thống phần mềm ngày càng trở nên phức tạp và quan trọng, việc đảm bảo chất lượng phần mềm là một yếu tố then chốt để duy trì hiệu quả hoạt động và giảm thiểu rủi ro. SVM, với khả năng phân loại mạnh mẽ và hiệu quả, đã được chứng minh là một công cụ hữu ích trong việc phát hiện và phân loại lỗi phần mềm.

Tầm quan trọng của đề tài này nằm ở khả năng nâng cao chất lượng phần mềm, giảm thiểu chi phí sửa lỗi và tăng cường độ tin cậy của hệ thống phần mềm. Bằng cách áp dụng SVM, các doanh nghiệp và tổ chức có thể phát hiện lỗi phần mềm sớm hơn, từ đó đưa ra các biện pháp khắc phục kịp thời. Điều này không chỉ giúp nâng cao chất lượng sản phẩm mà còn cải thiện trải nghiệm người dùng và tăng cường khả năng cạnh tranh trên thị trường.

Những lợi ích cụ thể của việc ứng dụng SVM trong phát hiện và phân loại lỗi phần mềm bao gồm:

- Cải thiện chất lượng phần mềm: Mô hình SVM có thể phát hiện và phân loại lỗi phần mềm chính xác hơn, từ đó giúp giảm thiểu lỗi và nâng cao chất lượng sản phẩm.

- Tăng năng suất: SVM có thể giúp tự động hóa quá trình phát hiện lỗi, từ đó giảm thiểu thời gian và chi phí liên quan đến việc kiểm thử và sửa lỗi.

- Cải thiện khả năng cạnh tranh: Các doanh nghiệp có thể sử dụng SVM để nâng cao chất lượng sản phẩm của mình, từ đó tăng cường khả năng cạnh tranh trên thị trường.

- Ứng dụng đa dạng: Kỹ thuật SVM không chỉ giới hạn trong việc phát hiện lỗi phần mềm mà còn có thể được áp dụng trong nhiều lĩnh vực khác như phân tích dữ liệu lớn, nhận dạng mẫu, và dự đoán xu hướng.

Tóm lại, đề tài "Phân lớp dữ liệu dự đoán lỗi phần mềm dựa trên thuật toán SVM" không chỉ có tính thời sự và tầm quan trọng cao mà còn mang lại nhiều lợi ích thiết thực cho cả người tiêu dùng và các nhà sản xuất phần mềm. Việc nghiên cứu và ứng dụng SVM trong phát hiện và phân loại lỗi phần mềm sẽ góp phần nâng cao chất lượng phần mềm, giảm thiểu rủi ro và tối ưu hóa quy trình phát triển phần mềm trong tương lai.

# **CHƯƠNG 3. PHƯƠNG PHÁP NGHIÊN CỨU**

## 3.1 Địa điểm và Thời gian Nghiên cứu (Thực tập)

- Địa điểm thực tập:

- Thời gian thực tập:

### 3.2 Nội dung Nghiên cứu

3.21. Tìm hiểu các Công nghệ liên quan để Thực hiện Đề tài

- Ngôn ngữ lập trình Python và các thư viện hỗ trợ.

### 3.2.2. Khảo sát Bài toán

- Xác định các loại lỗi phần mềm và các đặc trưng cần thiết để phát hiện lỗi.

- Thu thập và xử lý dữ liệu liên quan.

### 3.2.3. Tìm hiểu Thư viện và Cơ chế Hoạt động của Thuật toán

- Nghiên cứu các thư viện và công cụ liên quan đến Support Vector Machine (SVM).

- Hiểu rõ cơ chế hoạt động của SVM và các phương pháp tối ưu hóa mô hình.

### 3.2.4. Xây dựng Thuật toán

- Triển khai thuật toán SVM để phân loại lỗi phần mềm.

- Áp dụng các kỹ thuật tiền xử lý dữ liệu và lựa chọn đặc trưng.

### 3.2.5. Hoàn thiện Thuật toán

- Tối ưu hóa các siêu tham số của mô hình bằng GridSearch và Cross-validation.

- Đánh giá hiệu suất của mô hình qua các chỉ số đánh giá như:

- Accuracy

- Precision

- Recall

- F1 Score

## 3.3 Phương pháp Nghiên cứu

### 3.3.1 Nghiên cứu Lý thuyết

- Tham khảo tài liệu chuyên môn và tài liệu liên quan trên Internet:

- Nghiên cứu các bài báo khoa học, sách giáo trình, và tài liệu hướng dẫn liên quan đến SVM và phát hiện lỗi phần mềm.

- Tìm hiểu các nghiên cứu trước đây về ứng dụng SVM trong phát hiện lỗi phần mềm.

- Nghiên cứu lập trình thuật toán bằng Python:

- Học cách sử dụng các thư viện Python như numpy, pandas, và scikit-learn để triển khai và tối ưu hóa mô hình SVM.

- Tìm hiểu cách sử dụng Jupyter Notebook và Visual Studio 2022 để lập trình và kiểm thử mô hình.

- Nghiên cứu xây dựng ứng dụng thuật toán với các công cụ:

- Sử dụng Jupyter Notebook để viết mã nguồn và trực quan hóa kết quả.

- Áp dụng Visual Studio 2022 để phát triển và kiểm tra các thành phần của hệ thống.

### 3.3.2 Thực nghiệm

- Tham khảo tài liệu chuyên môn thông qua các tài liệu chuyên khảo, tài liệu online:

- Đọc các tài liệu chuyên ngành về SVM và phát hiện lỗi phần mềm.

- Tham khảo các bài viết và hướng dẫn trực tuyến để cập nhật kiến thức mới nhất.

- Tiếp nhận ý kiến của giảng viên hướng dẫn và định hướng:

- Thảo luận với giảng viên hướng dẫn để nhận được ý kiến đóng góp và định hướng nghiên cứu.

- Điều chỉnh kế hoạch nghiên cứu dựa trên các góp ý và phản hồi từ giảng viên.

- Hoàn thiện thuật toán:

- Tiến hành các thử nghiệm với các bộ dữ liệu khác nhau để kiểm tra hiệu suất của mô hình SVM.

- Tối ưu hóa mô hình dựa trên kết quả thực nghiệm và điều chỉnh các tham số nếu cần thiết.

- Đánh giá và so sánh hiệu suất của mô hình trước và sau khi tối ưu hóa để đưa ra kết luận về hiệu quả của các phương pháp đã áp dụng.

Bằng cách kết hợp nghiên cứu lý thuyết và thực nghiệm, đề tài sẽ đi sâu vào việc tìm hiểu và ứng dụng thuật toán SVM để phân loại lỗi phần mềm, từ đó đưa ra các giải pháp và đề xuất nhằm cải thiện chất lượng phần mềm trong thực tế.

# **CHƯƠNG 4. CƠ SỞ LÝ THUYẾT**

**Phân loại lỗi phần mềm và hệ thống phát hiện lỗi (Error Detection System)**

Trong bối cảnh hiện nay, đảm bảo chất lượng và độ tin cậy của phần mềm đã trở thành một phần không thể tách rời trong quá trình phát triển và vận hành hệ thống thông tin của các tổ chức, doanh nghiệp. Nói cách khác, việc phát hiện và phân loại lỗi phần mềm không chỉ là hàng rào bảo vệ khỏi các sự cố hệ thống mà còn là nền tảng giúp duy trì sự ổn định và hiệu quả của các ứng dụng trong thời đại số. Để thực hiện nhiệm vụ này, các giải pháp như kiểm thử phần mềm tự động, phân tích mã nguồn tĩnh, và các phương pháp kiểm soát chất lượng đã được triển khai rộng rãi. Tuy nhiên, sự xuất hiện của các hệ thống phát hiện lỗi phần mềm (Error Detection System - EDS) đánh dấu một bước tiến quan trọng, bổ sung cho những phương pháp truyền thống bằng cách sử dụng công nghệ tiên tiến để phát hiện sớm các dấu hiệu của lỗi phần mềm, từ đó cảnh báo kịp thời và triển khai các biện pháp khắc phục.

## 4.1 Các mục tiêu chính của đảm bảo chất lượng phần mềm

Có năm thuộc tính cốt lõi được sử dụng để đảm bảo chất lượng và độ tin cậy của phần mềm, thường được viết tắt là: ‘CARAN’:

- Correctness (Độ chính xác): Đảm bảo phần mềm hoạt động đúng như thiết kế và yêu cầu, không có lỗi logic hay chức năng.

- Availability (Tính sẵn sàng): Phần mềm luôn sẵn sàng hoạt động và phục vụ người dùng, giảm thiểu thời gian gián đoạn.

- Reliability (Độ tin cậy): Khả năng phần mềm hoạt động ổn định và đáng tin cậy trong một khoảng thời gian dài, không xảy ra lỗi bất ngờ.

- Adaptability (Khả năng thích ứng): Phần mềm có khả năng dễ dàng điều chỉnh và nâng cấp để đáp ứng các yêu cầu mới mà không làm giảm chất lượng.

- Non-repudiation (Không phủ nhận): Đảm bảo mọi hành động và giao dịch được thực hiện trong phần mềm đều có thể kiểm chứng và không bị chối bỏ.

## 4.2 Các giải pháp đảm bảo chất lượng phần mềm

### 4.2.1 Kiểm thử tự động (Automated Testing):

Kiểm thử tự động là một thành phần quan trọng của các hệ thống đảm bảo chất lượng phần mềm, giúp tự động hóa quá trình kiểm thử dựa trên các kịch bản đã được xác định trước. Kiểm thử tự động không chỉ giúp tiết kiệm thời gian và công sức mà còn tăng độ chính xác và khả năng phát hiện lỗi sớm.

Kiểm thử tự động có thể triển khai dưới dạng các công cụ phần mềm độc lập hoặc tích hợp trực tiếp vào quy trình phát triển phần mềm. Chúng cung cấp một cơ chế phòng thủ đầu tiên cho phần mềm, giúp phát hiện các lỗi logic, lỗi cú pháp và lỗi chức năng. Kiểm thử tự động cũng có khả năng ghi nhận và báo cáo chi tiết các kết quả kiểm thử, cung cấp thông tin để khắc phục và cải tiến phần mềm.

### 4.2.2 Phân tích mã nguồn tĩnh (Static Code Analysis):

Phân tích mã nguồn tĩnh là quá trình kiểm tra mã nguồn mà không thực thi nó. Các công cụ phân tích mã nguồn tĩnh giúp phát hiện các lỗi tiềm ẩn, vi phạm quy tắc lập trình và các lỗ hổng bảo mật ngay từ giai đoạn phát triển, từ đó cải thiện chất lượng mã nguồn và giảm thiểu rủi ro lỗi phần mềm trong tương lai.

### 4.2.3 Kiểm soát chất lượng phần mềm (Software Quality Control):

Kiểm soát chất lượng phần mềm bao gồm các hoạt động như đánh giá mã nguồn, kiểm tra đơn vị (unit testing), kiểm tra tích hợp (integration testing), và kiểm thử hệ thống (system testing). Mục tiêu là đảm bảo phần mềm đáp ứng đầy đủ các yêu cầu kỹ thuật và người dùng, cũng như tuân thủ các tiêu chuẩn chất lượng đã đề ra.

## 4.3 Các phương pháp phát hiện lỗi phần mềm

### 4.3.1 Hệ thống phát hiện lỗi dựa trên mạng (Network-based EDS):

Hệ thống phát hiện lỗi dựa trên mạng thường được triển khai để giám sát và phân tích lưu lượng mạng giữa các thành phần phần mềm, từ đó phát hiện các lỗi giao tiếp và tương tác không mong muốn.

### 4.3.2 Hệ thống phát hiện lỗi dựa trên máy chủ (Host-based EDS):

Hệ thống phát hiện lỗi dựa trên máy chủ tập trung vào việc giám sát và phân tích các hoạt động bên trong máy chủ, bao gồm các quy trình, dịch vụ và tài nguyên hệ thống, để phát hiện các lỗi phần mềm có thể gây ảnh hưởng đến hiệu suất và độ tin cậy của máy chủ.

### 4.3.3 Hệ thống phát hiện lỗi kết hợp (Hybrid EDS):

Hệ thống phát hiện lỗi kết hợp sử dụng cả hai phương pháp trên, tận dụng ưu điểm của mỗi phương pháp để cung cấp một giải pháp toàn diện hơn trong việc phát hiện và phân loại lỗi phần mềm.

### 4.3.4 Cơ sở dữ liệu trong phát hiện lỗi phần mềm

Trong việc phát triển và đánh giá các hệ thống phát hiện lỗi phần mềm, dữ liệu đóng một vai trò quan trọng. Các bộ dữ liệu như JM1, NASA Metrics Data Program và các tập dữ liệu khác đã được sử dụng rộng rãi trong cộng đồng nghiên cứu để làm nền tảng cho việc kiểm tra, đánh giá và huấn luyện các thuật toán phát hiện lỗi dựa trên học máy.

## 4.4 Mô hình học máy và học sâu trong phát hiện lỗi phần mềm

Học máy và học sâu đã cách mạng hóa cách em tiếp cận với phát hiện lỗi phần mềm, cung cấp khả năng tự động hóa phát hiện và phân loại các lỗi trong mã nguồn.

### 4.4.1 Học máy trong phát hiện lỗi phần mềm:

Các thuật toán học máy như Support Vector Machines (SVM), Decision Trees, và Random Forests đã được áp dụng rộng rãi trong việc phân loại lỗi phần mềm. Chúng cho phép máy tính học từ dữ liệu quá khứ và phát triển khả năng dự đoán hoặc nhận dạng các mẫu lỗi mới. Tuy nhiên, một thách thức lớn là sự không cân đối dữ liệu khi mà mã nguồn không lỗi chiếm đa số và mã nguồn lỗi là hiếm. Do đó, các phương pháp tái cấu trúc và tái cân bằng dữ liệu như SMOTE (Synthetic Minority Over-sampling Technique) đã được áp dụng để cải thiện khả năng nhận dạng lỗi của mô hình.

### 4.4.2 Học sâu trong phát hiện lỗi phần mềm:

Học sâu, một nhánh của học máy, sử dụng các mạng nơ-ron sâu để phân tích dữ liệu. Các mô hình như Convolutional Neural Networks (CNN) và Recurrent Neural Networks (RNN) được sử dụng để phát hiện các mẫu phức tạp trong mã nguồn. Các mô hình này đặc biệt hữu ích trong việc xử lý và học từ dữ liệu chuỗi thời gian, một đặc điểm chung của quy trình phát triển phần mềm. Việc áp dụng các kiến trúc nâng cao như Gated Recurrent Units (GRU) hoặc Attention Mechanisms đã giúp tăng cường khả năng hiểu biết về các lỗi trong mã nguồn.

## 4.5 Tập dữ liệu và đánh giá mô hình phát hiện lỗi phần mềm

Các tập dữ liệu như JM1 và NASA Metrics Data Program là những nguồn dữ liệu có giá trị cho việc đánh giá các hệ thống phát hiện lỗi phần mềm. Tuy nhiên, việc lựa chọn đánh giá mô hình không chỉ dừng lại ở độ chính xác phát hiện mà còn phải cân nhắc đến tỷ lệ báo động giả (false positive rate), thời gian phản hồi và khả năng tích hợp của mô hình vào quy trình phát triển phần mềm hiện hữu.

## 4.6 Các thách thức trong đánh giá mô hình phát hiện lỗi phần mềm

Trong quá trình đánh giá mô hình phát hiện lỗi phần mềm, em gặp phải một loạt các thách thức cần được giải quyết. Sự không cân đối giữa các lớp dữ liệu (data imbalance) là một vấn đề lớn, vì trong thực tế, số lượng mã nguồn không lỗi thường nhiều hơn rất nhiều so với mã nguồn lỗi, làm giảm khả năng nhận diện của mô hình. Sự biến đổi của các dạng lỗi (evolving error patterns) làm cho việc phát hiện trở nên khó khăn hơn, đòi hỏi mô hình phải liên tục được cập nhật và tu chỉnh để theo kịp với các thay đổi mới. Bên cạnh đó, việc thiếu một phương pháp đánh giá thống nhất khiến việc so sánh hiệu quả giữa các mô hình khác nhau trở nên phức tạp và thiếu chính xác. Để giải quyết các vấn đề này, cần có một hệ thống đánh giá tiêu chuẩn và cập nhật, cùng với việc áp dụng các kỹ thuật cải thiện mô hình như học không cân đối (imbalance learning) và học tăng cường (reinforcement learning) để tối ưu hóa hiệu suất phát hiện của các mô hình phát hiện lỗi phần mềm.

# **CHƯƠNG 5. MÔ HÌNH ĐỀ XUẤT**

Trong chương này, em sẽ giới thiệu quy trình xây dựng mô hình phân loại lỗi phần mềm dựa trên việc lựa chọn đặc trưng thông minh và áp dụng các mô hình phân loại hiệu quả. Sử dụng tập dữ liệu JM1 từ NASA, nhấn mạnh vào việc tinh giản không gian đặc trưng thông qua phương pháp Recursive Feature Elimination (RFE) kết hợp với mô hình RandomForest. Quy trình này giúp loại bỏ thông tin không cần thiết, giảm thiểu nguy cơ overfitting, và cải thiện đáng kể khả năng tổng quát hóa của mô hình. Phần tiếp theo sẽ chi tiết hóa việc tiền xử lý dữ liệu, lựa chọn đặc trưng, và cuối cùng là lựa chọn và huấn luyện mô hình, với mục tiêu nâng cao hiệu suất phát hiện lỗi phần mềm.

## 5.0 Dữ liệu sử dụng

Tập dữ liệu JM1 được sử dụng để xây dựng và đánh giá các mô hình phân loại lỗi phần mềm trong nghiên cứu này được cung cấp thông qua NASA Metrics Data Program. Dữ liệu này bao gồm các chỉ số phần mềm được trích xuất từ mã nguồn, chứa cả các mẫu không lỗi và các mẫu có lỗi cụ thể. Mỗi mẫu được gắn nhãn là bình thường (không có lỗi) hoặc có lỗi (lỗi phần mềm cụ thể).

Mỗi bản ghi trong tập dữ liệu chứa 22 đặc trưng định lượng và định tính, bao gồm các chỉ số McCabe và Halstead, cùng với các thông tin về lỗi phần mềm.

Biến phân loại có hai danh mục:

- Bình Thường (Normal): Các mẫu không chứa lỗi phần mềm.

- Có Lỗi (Defective): Các mẫu chứa lỗi phần mềm.

## 5.1 Phân tích khám phá dữ liệu và xử lý dữ liệu

Em đã tiến hành phân tích khám phá dữ liệu (Data Exploratory Analysis) và trực quan hóa dữ liệu (Data Visualization), cũng như quá trình xử lý dữ liệu (Data Processing) để hiểu rõ hơn về tập dữ liệu và các đặc điểm quan trọng của nó.

### 4.1.1 Xử lý dữ liệu rỗng và dữ liệu dạng chuỗi

- Kiểm tra và xử lý các giá trị null để đảm bảo không có thông tin trống nào gây ảnh hưởng đến mô hình.

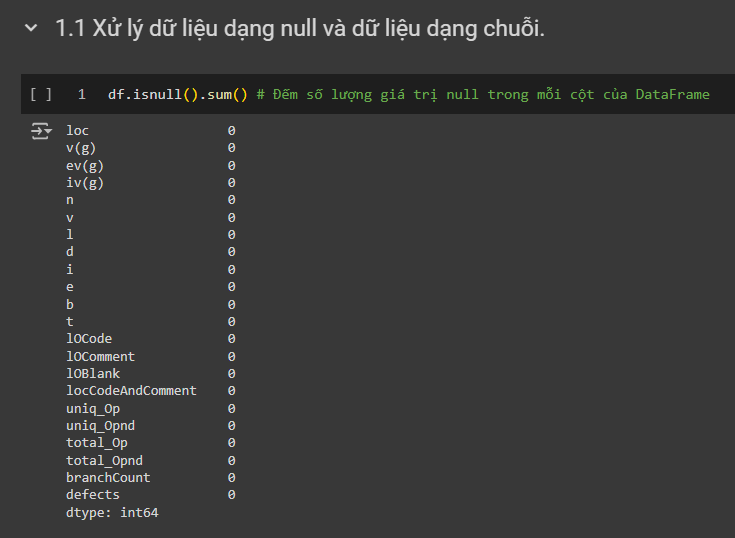
- Mã hóa các đặc trưng dạng chuỗi (`protocol\_type`, `service`, `flag`) thành dữ liệu số bằng cách sử dụng:

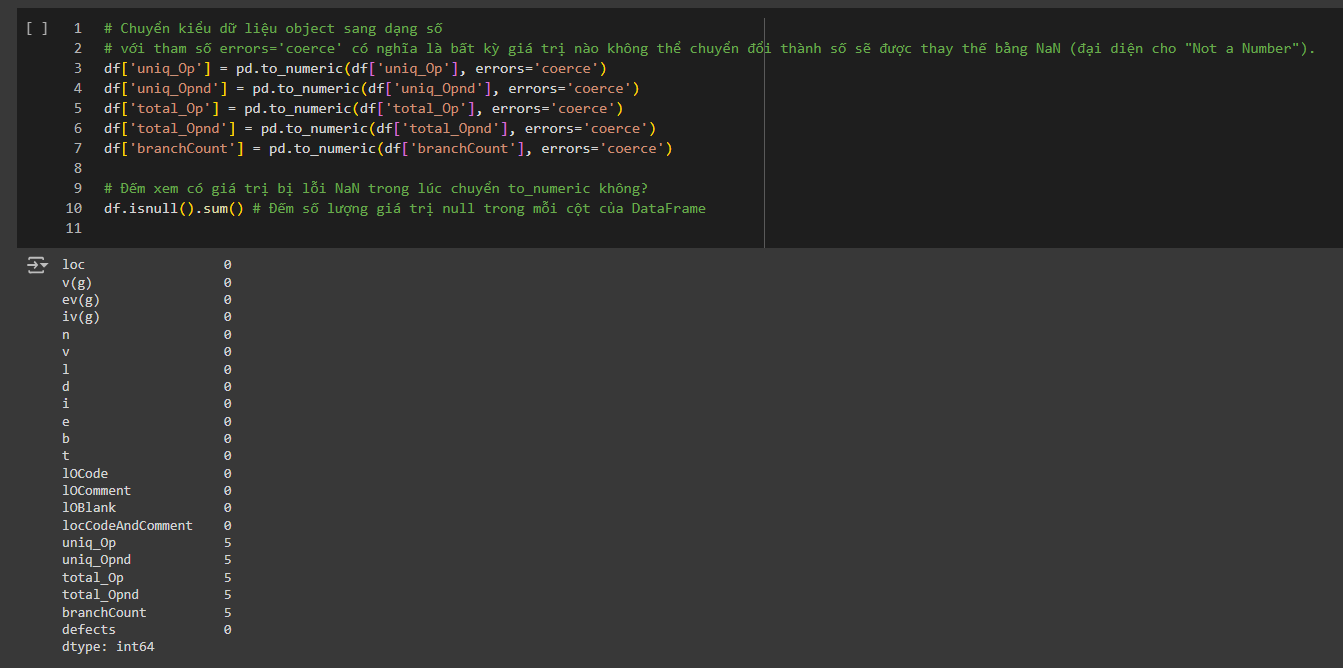
- `pd.to\_numeric` để chuyển đổi các cột dạng chuỗi thành số, đặt `errors='coerce'` để chuyển đổi các giá trị không thể mã hóa thành NaN.

- `LabelEncoder` để gán mỗi giá trị danh mục một số nguyên duy nhất.

- `OneHotEncoder` để chuyển các giá trị danh mục thành vector nhị phân, loại bỏ sự thiên vị của số nguyên.

Với bộ dataset được sử dụng trong đề tài, phương thức `pd.to\_numeric` đã được sử dụng để tiền sử lý dữ liệu, giúp chuyển số ở dạng `string` về số ở dạng `interger`.





### 4.1.2 Chọn lọc đặc trưng đơn giản

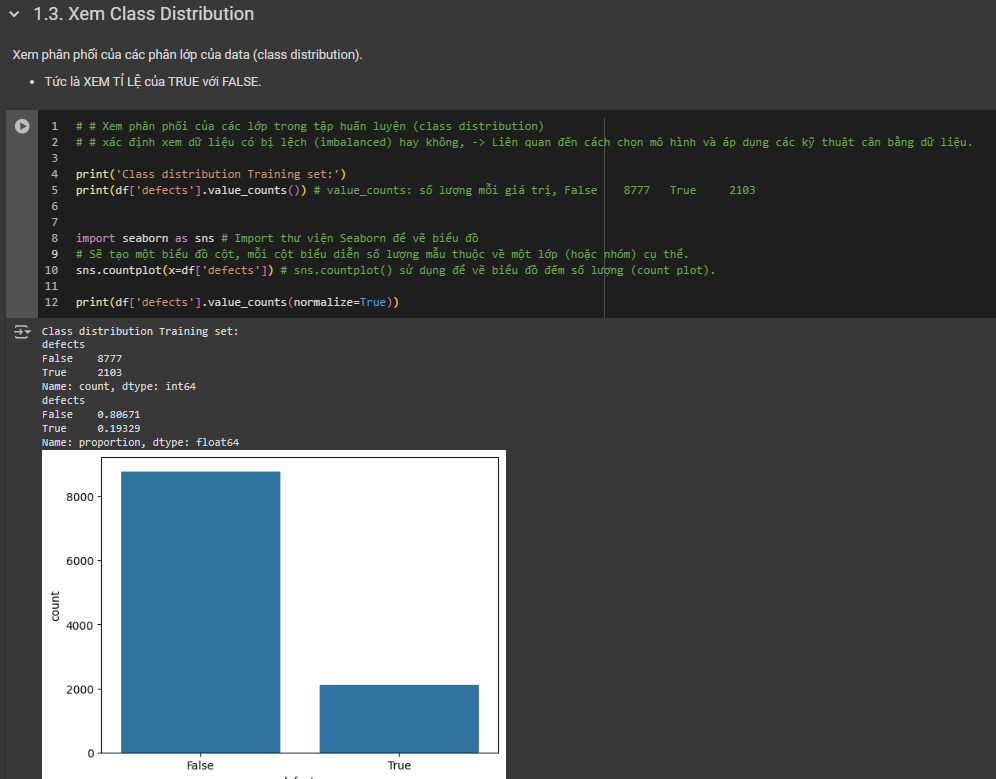
- Kiểm tra và loại bỏ các cột có giá trị giống hệt nhau, vì chúng không cung cấp thông tin hữu ích cho mô hình.

- Đánh giá sự đa dạng của các giá trị trong từng cột và loại bỏ hoặc xử lý thêm các cột có quá ít giá trị khác nhau.

### 4.1.3 Đánh giá phân phối lớp

- Kiểm tra tỷ lệ giữa các lớp để đánh giá mức độ mất cân bằng.

- Trực quan hóa sự phân bố của các lớp bằng biểu đồ để nhận biết rõ hơn về sự mất cân bằng nếu có.



Dữ liệu classification cho thấy có sự mất cân bằng lớp (class imbalance): có nhiều mẫu "False" hơn so với mẫu "True".

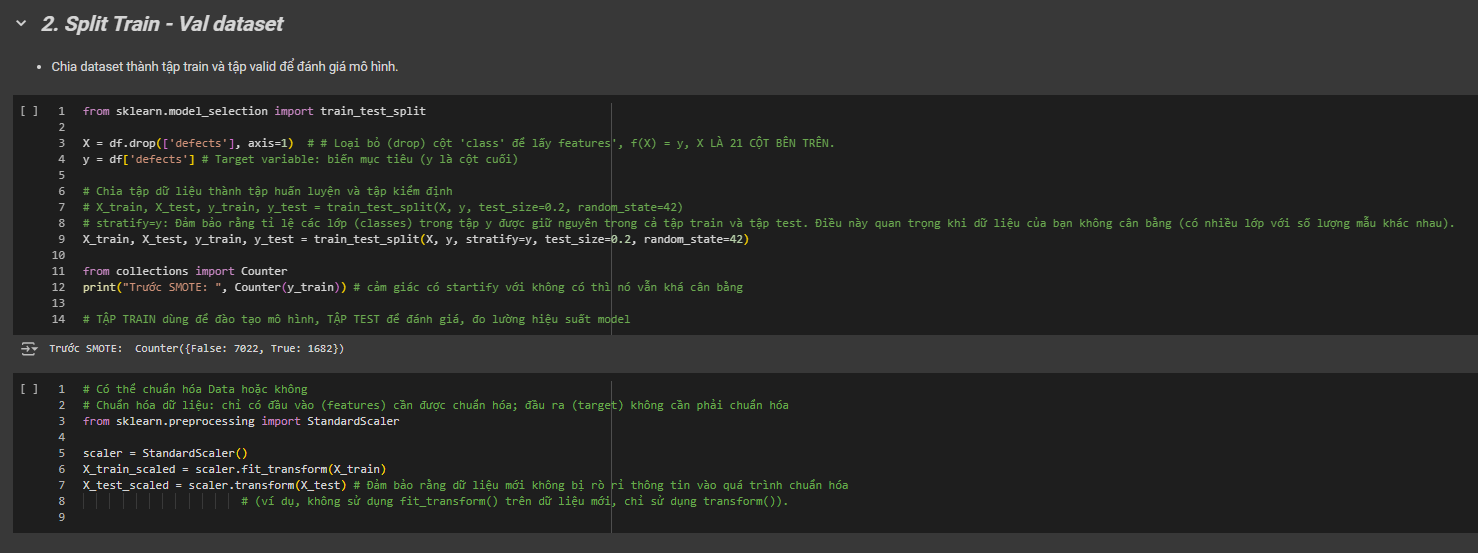
=> Điều này có thể ảnh hưởng đến hiệu suất của mô hình học máy, vì mô hình có thể thiên vị về lớp đa số và không thể học hiệu quả từ lớp thiểu số.

Trong đề tài được giao, thì em sử dụng SMOTE để tăng số lượng mẫu trong lớp thiểu số bằng cách tái sử dụng mẫu hoặc tạo mẫu tổng hợp (em sẽ trình bày chi tiết hơn ở phần dưới).

### 4.1.4 Chia dữ liệu và chuẩn hóa

- Chia dữ liệu thành tập huấn luyện và kiểm thử.

- Chuẩn hóa các đặc trưng để các giá trị nằm trong cùng một khoảng, giúp mô hình SVM hoạt động hiệu quả hơn.



Mã sử dụng `train\_test\_split` để chia tập dữ liệu thành tập huấn luyện và kiểm định, đảm bảo giữ nguyên tỷ lệ các lớp trong y bằng cách sử dụng `stratify=y`. Sau đó, chuẩn hóa dữ liệu đầu vào bằng `StandardScaler`: `fit\_transform` được sử dụng trên tập huấn luyện để tính toán và áp dụng chuẩn hóa, trong khi `transform` được sử dụng trên tập kiểm định để áp dụng cùng các thống kê đã tính từ tập huấn luyện. Điều này đảm bảo tính nhất quán và tránh rò rỉ thông tin giữa các tập dữ liệu.

## 5.2 Chọn và Huấn luyện Mô hình

### 5.2.1 Tại sao chọn model SVM

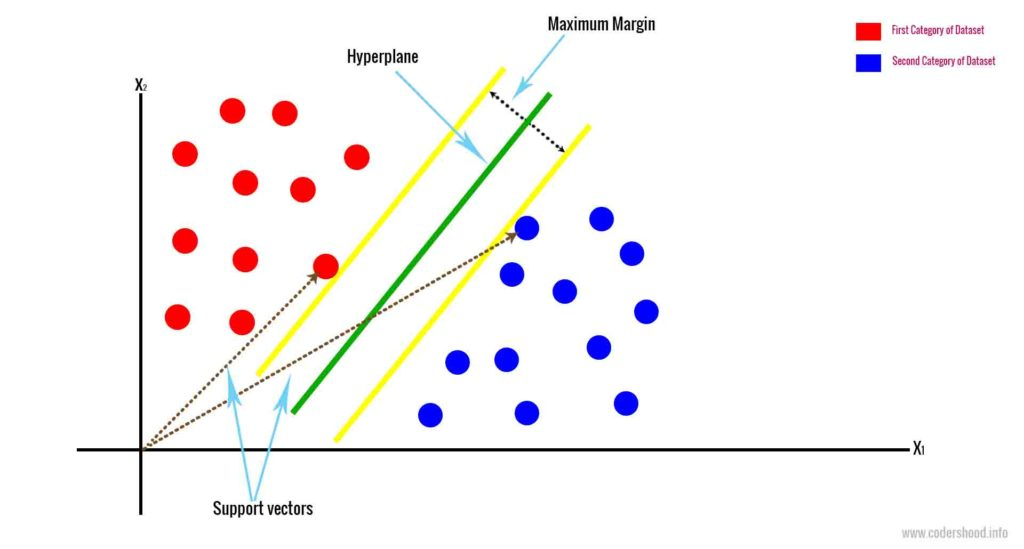
Trong đề tài này, em nghiên cứu model Support Vector Machine (SVM) cho bài toán phân loại lỗi phần mềm do những lý do sau:

- Hiệu suất cao: SVM có khả năng tạo ra mặt phẳng phân cách với biên lớn nhất giữa các lớp, giúp tối đa hóa khoảng cách giữa các mẫu dữ liệu thuộc hai lớp khác nhau, từ đó giảm thiểu sai số phân loại.

- Làm việc tốt với dữ liệu nhiều chiều: SVM rất hiệu quả trong việc xử lý dữ liệu có nhiều đặc trưng, phù hợp với tập dữ liệu có nhiều chỉ số phần mềm như trong nghiên cứu này.

- Khả năng tổng quát hóa: SVM có khả năng tổng quát hóa tốt, giúp mô hình hoạt động hiệu quả trên dữ liệu chưa thấy trước đó, giảm nguy cơ overfitting.

- Sử dụng kernel trick: SVM có thể áp dụng kernel trick để biến đổi dữ liệu phi tuyến tính thành tuyến tính, giúp phân loại các mẫu dữ liệu phức tạp một cách hiệu quả.



*HÌNH 3.6 Thuật toán Support Vector Machine*

(Nguồn: [Support Vector Machine Machine learning algorithm](about:blank), truy cập 14/02/2024)

### 5.2.2 Tổng Quan về Support Vector Machines (SVM)

Support Vector Machines (SVM) là một thuật toán học máy giám sát, sử dụng cho cả phân loại và hồi quy, nhưng chủ yếu được áp dụng trong các bài toán phân loại. SVM hoạt động bằng cách tìm kiếm siêu phẳng (hyperplane) tốt nhất để phân chia dữ liệu thành các lớp khác nhau. Trong không gian n chiều (với n là số lượng các tính năng), mỗi điểm dữ liệu là một vector trong không gian này và siêu phẳng là đường hoặc mặt phẳng phân chia dữ liệu thành các lớp khác nhau. Mục tiêu chính của SVM là tối ưu hóa biên phân cách và giảm thiểu các lỗi phân lớp thông qua việc sử dụng hàm mất mát hinge,

### 5.2.3 Các Thành Phần Chính của SVM

**1. Support Vectors:**

- Định nghĩa: Support vectors là các điểm dữ liệu gần với siêu phẳng nhất và ảnh hưởng trực tiếp đến vị trí và định hướng của siêu phẳng.

- Vai trò: Đây là các điểm quan trọng nhất trong việc xác định biên phân cách tối ưu. Nếu di chuyển các support vectors thì siêu phẳng cũng sẽ thay đổi.

**2. Hyperplane:**

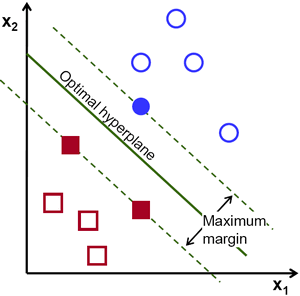
- Định nghĩa: Siêu phẳng trong không gian nhiều chiều là một đường (với không gian 2 chiều) hoặc một mặt phẳng (với không gian 3 chiều), và nói chung là một không gian con với chiều thấp hơn không gian dữ liệu gốc.

- Cách xác định: Siêu phẳng được xác định sao cho khoảng cách từ nó đến các support vectors là lớn nhất, tức là tối đa hóa margin, từ đó thu được một siêu phẳng tạo khoảng cách xa nhất giữa các lớp. Nhờ vậy, SVM có thể giảm thiểu việc phân lớp sai (misclassification) đối với điểm dữ liệu mới đưa vào.

- Cách tính margin:

- Margin được tính bằng khoảng cách vuông góc từ siêu phẳng đến các support vectors. Công thức tính margin thường dựa trên các tham số của siêu phẳng, chẳng hạn như vector trọng số w và bias b.

- Khi margin lớn, siêu phẳng phân chia các lớp một cách rõ ràng và có khả năng phân loại tốt các điểm dữ liệu mới mà mô hình chưa từng thấy. Điều này giúp mô hình tổng quát hóa tốt hơn, giảm thiểu lỗi trên các dữ liệu kiểm tra (test data).



- Phương trình của siêu phẳng:

- Trong không gian n chiều, siêu phẳng có thể được mô tả bằng phương trình trình , trong đó w là vector trọng số và b là bias.

- Các support vectors nằm trên hai đường song song với siêu phẳng chính, và phương trình của chúng là . Khoảng cách giữa hai đường này chính là

- Quá trình huấn luyện SVM:

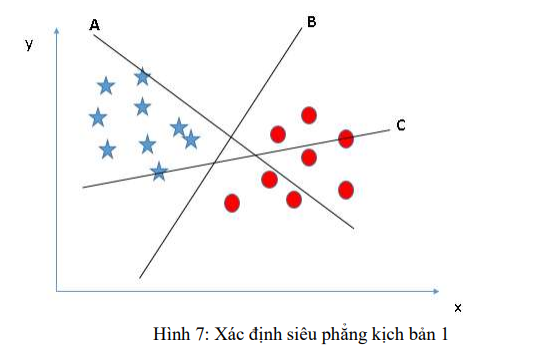
- Quá trình huấn luyện SVM tìm ra vector trọng số w

và bias b để tối đa hóa margin. Bài toán này thường được giải bằng cách sử dụng kỹ thuật tối ưu hóa, chẳng hạn như phương pháp Lagrange.

### 5.2.4 Phương Pháp Làm Việc của SVM

**1. Xác định Siêu Phẳng Đúng (Kịch Bản 1):**

- Trong một không gian dữ liệu, có nhiều khả năng cho các siêu phẳng khác nhau. Ví dụ, với ba siêu phẳng A, B, và C, quy tắc đầu tiên để chọn siêu phẳng là chọn siêu phẳng phân chia hai lớp tốt nhất. Trong trường hợp này, siêu phẳng B là lựa chọn chính xác nhất.

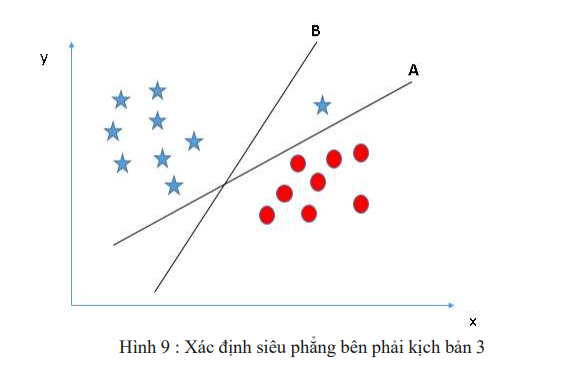


**2. Xác định Siêu Phẳng Đúng (Kịch Bản 2):**

- Khi có nhiều siêu phẳng đều thỏa mãn quy tắc đầu tiên, ta áp dụng quy tắc thứ hai: chọn siêu phẳng có margin lớn nhất. Margin là khoảng cách từ điểm gần nhất của một lớp đến siêu phẳng. Trong ví dụ này, siêu phẳng C có margin lớn nhất và do đó là lựa chọn đúng.

**3. Xác định Siêu Phẳng Đúng (Kịch Bản 3):**

- Trong các trường hợp phức tạp hơn, việc lựa chọn siêu phẳng có thể dựa trên cả hai quy tắc. Nếu một siêu phẳng có margin lớn hơn nhưng không phân chia các lớp thành riêng biệt, thì nó không được chọn. Thay vào đó, siêu phẳng A là lựa chọn chính xác vì nó phân chia các lớp tốt nhất mặc dù có margin thấp hơn.



**Giải Quyết Các Trường Hợp Không Thể Tách Tuyến Tính**

Trong một số trường hợp, dữ liệu không thể được phân chia rõ ràng bằng một siêu phẳng tuyến tính trong không gian gốc. SVM giải quyết vấn đề này bằng cách áp dụng kỹ thuật kernel (kernel trick). Kỹ thuật kernel cho phép SVM làm việc với các không gian có chiều cao hơn bằng cách thêm các tính năng mới hoặc biến đổi dữ liệu ban đầu. Ví dụ, bằng cách thêm một tính năng mới SVM có thể tìm một siêu phẳng trong không gian mới (z, x) để phân chia dữ liệu thành các lớp riêng biệt.

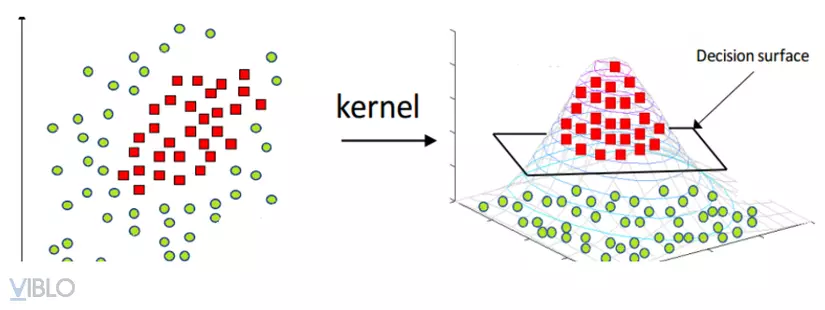
Kỹ thuật kernel giúp SVM xử lý các bài toán phân loại không tuyến tính mà không cần phải thêm các tính năng bằng tay. Các kernel phổ biến bao gồm:

- Polynomial Kernel: Cho phép SVM phân chia dữ liệu bằng một đa thức.

- Gaussian (RBF) Kernel: Sử dụng hàm Gaussian để phân chia dữ liệu không tuyến tính.

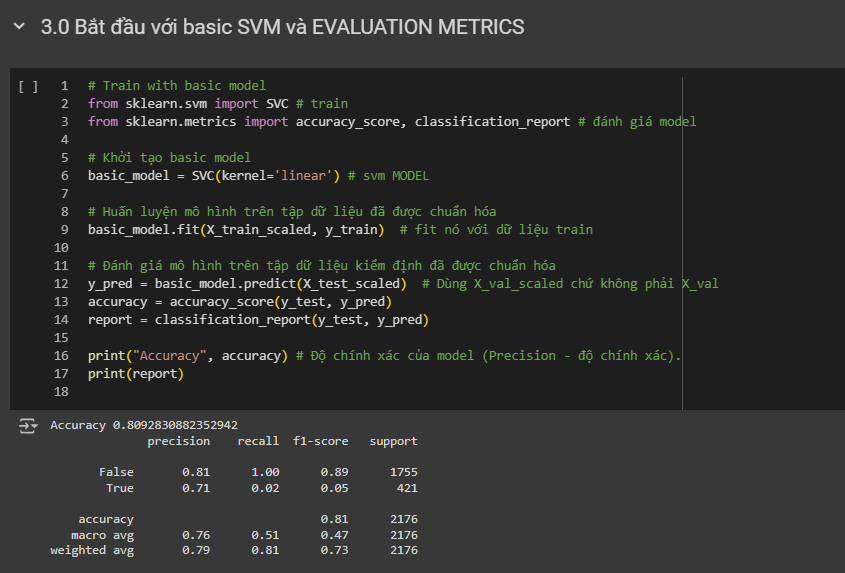
- Sigmoid Kernel: Áp dụng hàm sigmoid cho phân loại.

SVM có khả năng làm việc hiệu quả với dữ liệu có chiều cao mà không cần lớn về số lượng, giúp nó trở thành một công cụ mạnh mẽ trong các bài toán học máy phức tạp.



## 5.3 Đánh giá mô hình ban đầu – SMOTE – LỰA CHỌN ĐẶC TRƯNG

Em tiến hành train model SVM với các tham số mặc định, dữ liệu mới qua tiền xử lý (xử lý dữ liệu rỗng và dữ liệu dạng chuỗi, chia dữ liệu và chuẩn hoá, …)



Dựa trên kết quả của mô hình SVM cơ bản sau quá trình huấn luyện, em nhận thấy một sự chênh lệch lớn giữa hiệu suất phân loại của hai lớp:

- Lớp "False" có precision cao (0.81), recall rất cao (1.00), và F1-score cao (0.89), cho thấy mô hình rất tốt trong việc phân loại các mẫu không phải là lỗi.

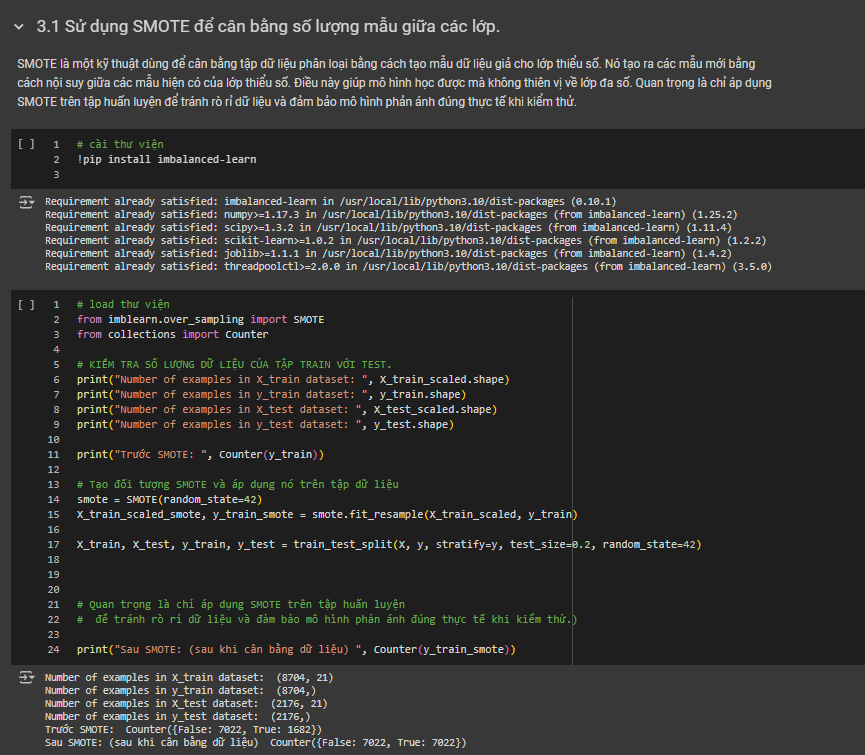
- Tuy nhiên, lớp "True" lại có precision tương đối cao (0.71), recall rất thấp (0.02), và F1-score rất thấp (0.05), điều này cho thấy mô hình không phân loại tốt lớp này và hầu như bỏ qua việc phát hiện lỗi.

Sự mất cân bằng lớp gây ra kết quả này vì mô hình có thể đã thiên vị về lớp đa số và không "học" đủ thông tin từ lớp thiểu số. Điều này khiến accuracy tổng thể cao, nhưng không phản ánh đúng chất lượng của mô hình đối với lớp thiểu số.

Để cải thiện, em tiến hành cân bằng dữ liệu bằng kỹ thuật SMOTE, điều chỉnh siêu tham số của mô hình, và áp dụng các phương pháp học máy nâng cao như ensemble methods trong các nghiên cứu sau này.

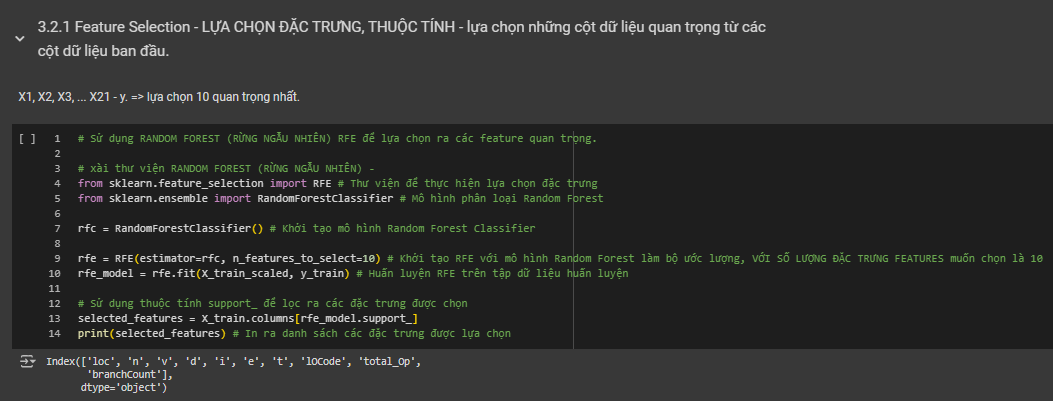
### 4.3.1 Sử dụng SMOTE để cân bằng số lượng mẫu giữa các lớp

Synthetic Minority Over-sampling Technique (SMOTE) là một kỹ thuật dùng để cân bằng tập dữ liệu phân loại bằng cách tạo mẫu dữ liệu giả cho lớp thiểu số. Nó tạo ra các mẫu mới bằng cách nội suy giữa các mẫu hiện có của lớp thiểu số. Điều này giúp mô hình học được mà không thiên vị về lớp đa số. Quan trọng là chỉ áp dụng SMOTE trên tập huấn luyện để tránh rò rỉ dữ liệu và đảm bảo mô hình phản ánh đúng thực tế khi kiểm thử.



### 4.3.2 Feature Selection (Lựa chọn đặc trưng)

Em sử dụng Recursive Feature Elimination (RFE) kết hợp với RandomForest để chọn 10 đặc trưng quan trọng nhất cho mô hình phân loại lỗi phần mềm. Quá trình này bắt đầu bằng việc huấn luyện mô hình trên toàn bộ đặc trưng và loại bỏ dần những đặc trưng ít quan trọng. (Mỗi đặc trưng được đánh giá dựa trên mức độ ảnh hưởng của nó đến độ chính xác của mô hình. Đặc trưng được xem là ít quan trọng nhất sẽ được loại bỏ, và quy trình này được lặp lại cho đến khi chỉ còn lại số lượng đặc trưng mong muốn).



Sau khi sử dụng RFE, em thu được 10 đặc trưng quan trọng nhất của bộ dữ liệu.

```Index(['loc', 'iv(g)', 'v', 'd', 'i', 'e', 't', 'lOCode', 'total\_Op',

'branchCount'],

dtype='object')```

Tiếp theo, em tiến hành điều chỉnh siêu tham số để cải thiện hiệu suất của Model.

## 5.4 Điều chỉnh siêu tham số (Tuning Model)

Nhằm tối ưu hóa hiệu suất của model, việc điều chỉnh siêu tham số (hyperparameter tuning) là rất cần thiết. Quá trình này không chỉ giúp mô hình phát hiện chính xác hơn các loại lỗi phần mềm trên một tập dữ liệu cụ thể mà còn đảm bảo rằng mô hình có khả năng thích ứng và phản ứng hiệu quả với sự thay đổi của dữ liệu.

Trong đề tài, em nghiên cứu 2 cách điều chỉnh siêu tham số

- Cách 1: cách thủ công:

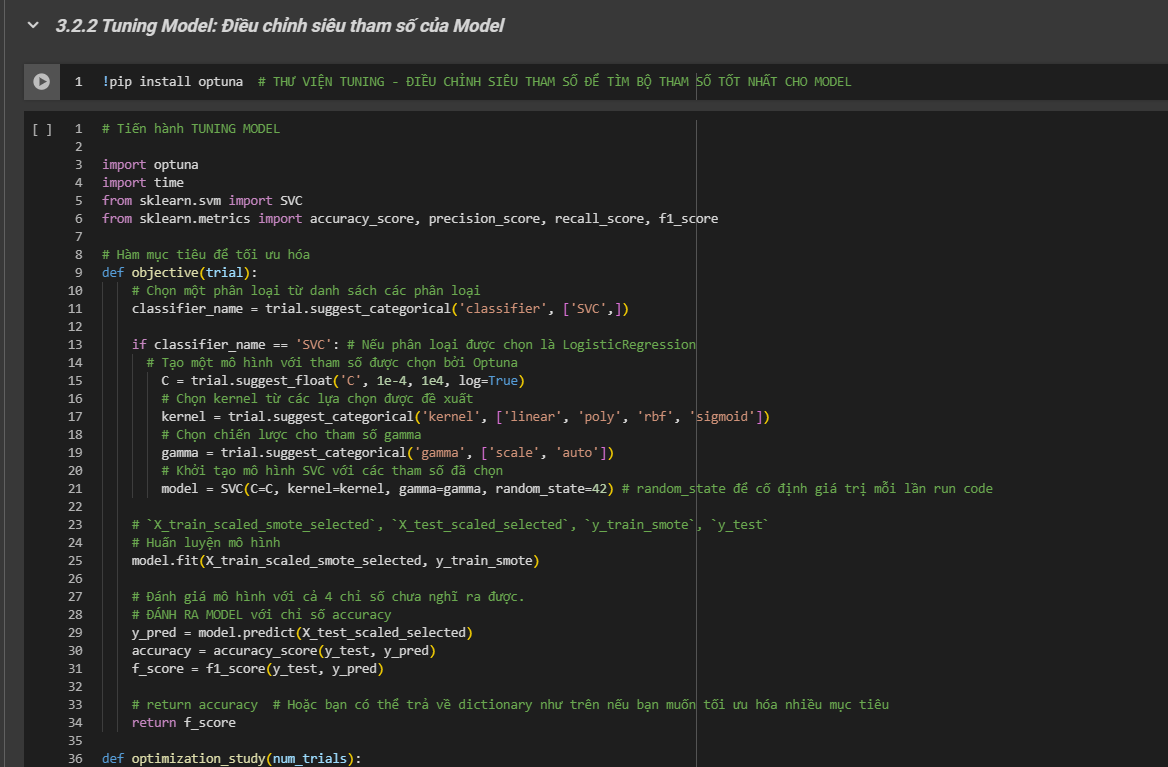
- Chọn một khoảng giá trị cho các siêu tham số như `C` (hệ số điều chỉnh độ phạt) và `gamma` (tham số kernel).

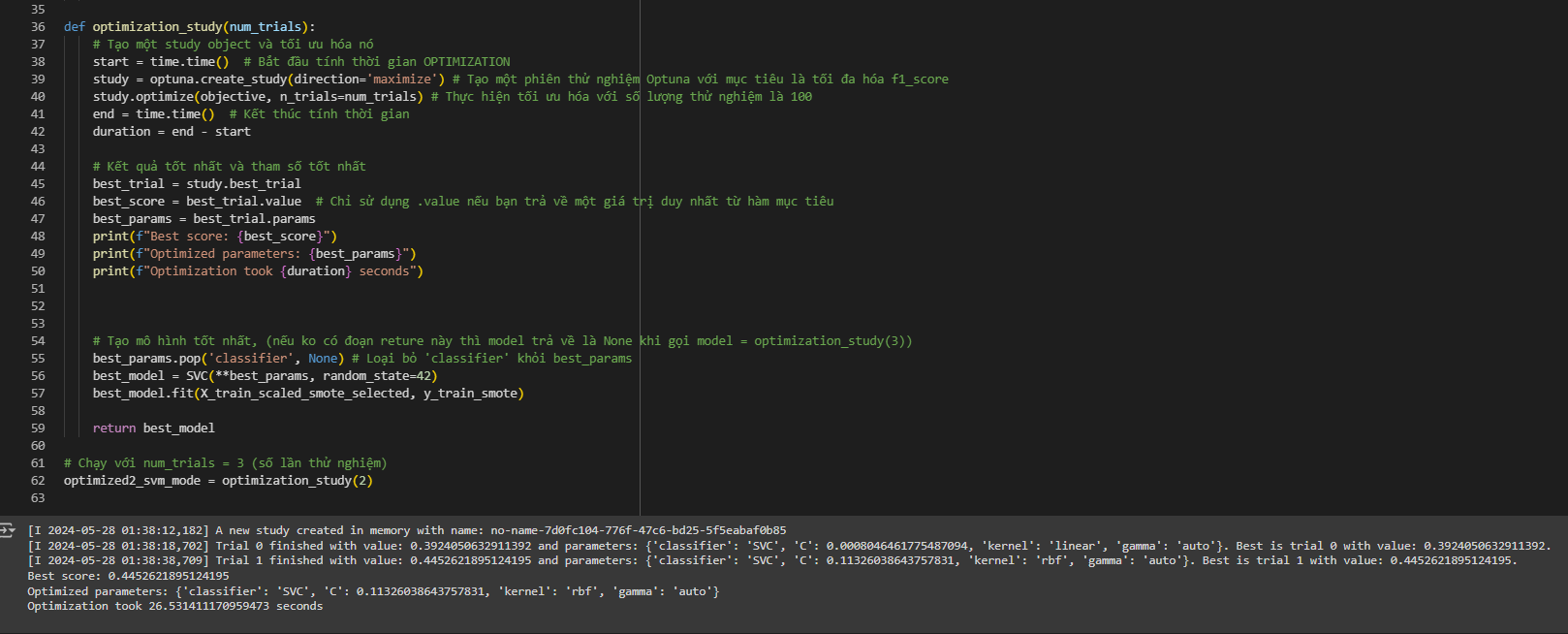
- Thực hiện thử nghiệm với từng giá trị và đánh giá hiệu suất mô hình.

- Cách 2: Tuning nhanh chóng với thư viện Optuna:

- Optuna là một thư viện mã nguồn mở giúp tự động hóa quá trình điều chỉnh siêu tham số bằng cách tìm kiếm không gian siêu tham số hiệu quả.

- Thực hiện tìm kiếm với Optuna để tối ưu hóa các siêu tham số của mô hình SVM.





Mã sử dụng thư viện Optuna để tối ưu hóa các tham số của mô hình Support Vector Machine (SVM). Hàm `objective` xác định mục tiêu tối ưu hóa bằng cách chọn các tham số như `C`, `kernel`, và `gamma` để huấn luyện mô hình SVM. Sau đó, mô hình được đánh giá dựa trên chỉ số f1\_score. Hàm `optimization\_study` thực hiện quá trình tối ưu hóa với số lần thử nghiệm được chỉ định, và trả về mô hình SVM tốt nhất với các tham số đã tối ưu hóa.

## 5.5 Cách thức đánh giá hiệu suất mô hình

Đánh giá hiệu suất của mô hình phân loại là một phần không thể thiếu trong quá trình phát triển mô hình. Có nhiều tiêu chí đánh giá khác nhau được sử dụng để đánh giá khả năng của mô hình trong việc phân loại chính xác các mẫu dữ liệu. Trong nghiên cứu này, tôi sử dụng bốn tiêu chí đánh giá chính là:

* Accuracy: Độ Chính xác
* Precision: Độ Chính xác tích cực
* Recall: Độ Nhạy
* F1 Score: Điểm số F1

### 5.5.1 Accuracy: Độ Chính xác

Là tỉ lệ của tổng số mẫu được phân loại đúng so với tổng số mẫu. Độ chính xác không chỉ cho biết số lượng mẫu dự đoán chính xác mà còn phản ánh cả hai loại lỗi phân loại, cả dương tính giả (FP) và âm tính giả (FN).

### 5.5.2 Precision: Độ Chính xác tích cực

Đo lường tỷ lệ các mẫu dự đoán là tích cực thực sự chính xác so với tổng số mẫu dự đoán là tích cực. Điều này quan trọng khi chi phí của việc dự đoán sai một mẫu là âm tính cao hơn nhiều.

### 5.5.3 Recall: Độ Nhạy

Còn được gọi là độ nhạy cảm hoặc tỉ lệ dương tính thực. Nó đo lường tỷ lệ các mẫu thực sự tích cực được phân loại đúng so với tổng số mẫu thực sự tích cực. Điều này quan trọng trong các tình huống mà việc bỏ sót một mẫu tích cực có thể dẫn đến hậu quả nghiêm trọng.

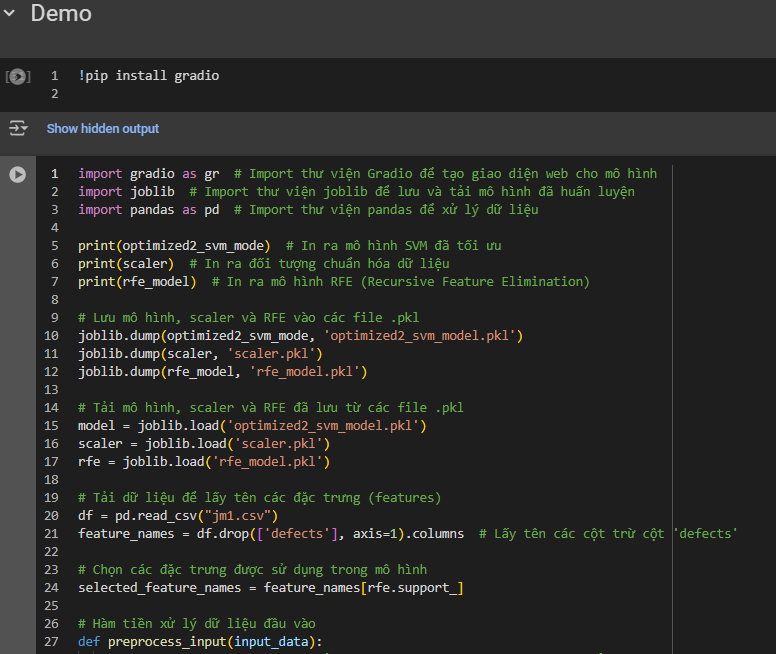
### 5.5.4 F1 Score: Điểm số F1

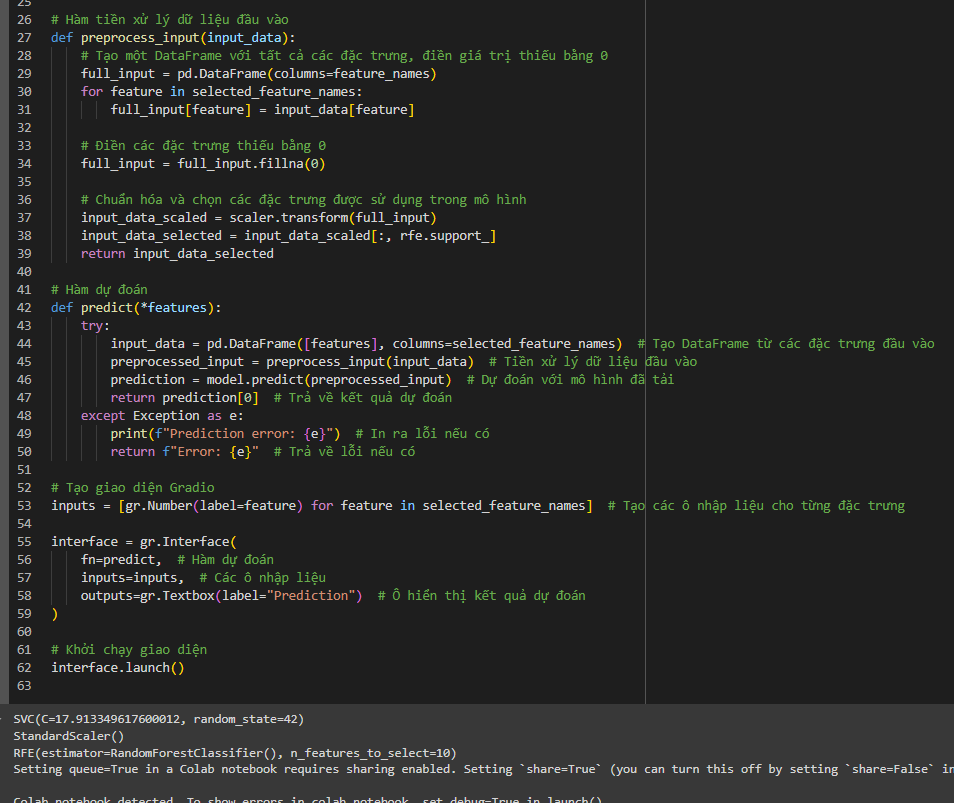
Là trung bình điều hòa của Precision và Recall, nó cung cấp một số đo đơn nhất của hiệu suất mô hình ở cả hai tiêu chí này. Điểm số F1 đặc biệt hữu ích khi ta cần cân bằng giữa Precision và Recall và khi có sự không cân đối giữa các lớp phân loại.

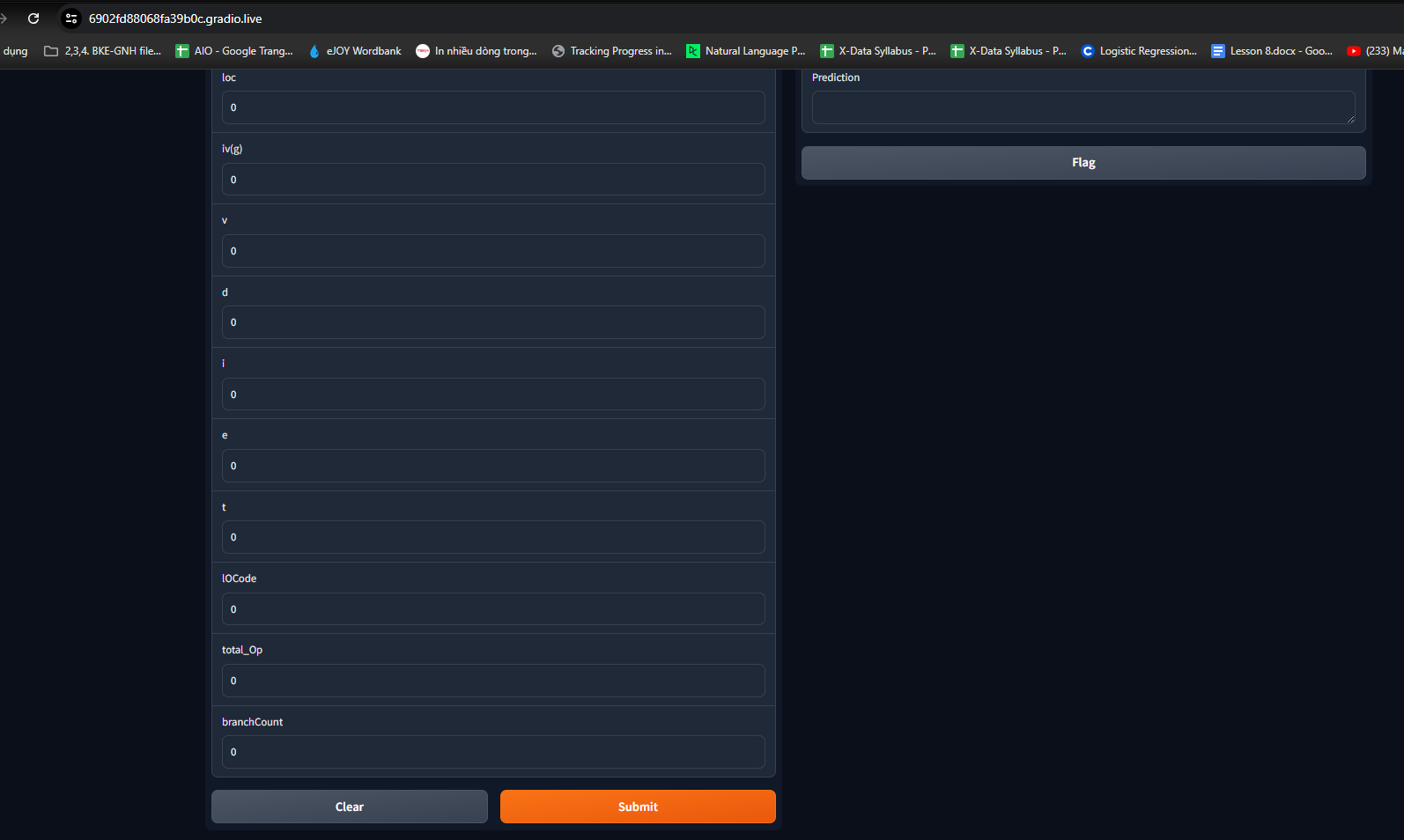
Sử dụng các tiêu chí này giúp tôi đánh giá khả năng tổng quát của mô hình trong việc phân loại cũng như phát hiện các điểm mạnh và điểm yếu cụ thể trong dự đoán của mô hình.

## 5.6 Demo

Em sử dụng Gradio để demo project.







# **CHƯƠNG 6. KẾT QUẢ THỰC NGHIỆM**

## 6.1 Kết quả thực nghiệm

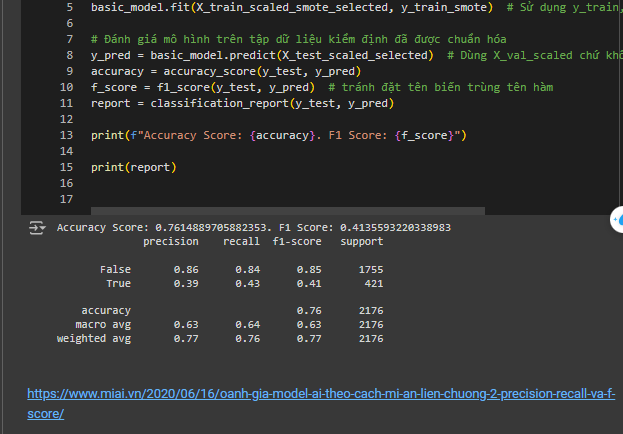
Trong phần này, em sẽ trình bày kết quả của quá trình thực nghiệm phân loại lỗi phần mềm sử dụng mô hình Support Vector Machine (SVM). Em sẽ so sánh hiệu suất của mô hình trước và sau khi tinh chỉnh tham số (tuning).

### 6.1.1 Trước khi Tuning Model

Trước khi tinh chỉnh tham số, em huấn luyện mô hình SVM với các siêu tham số mặc định. Kết quả đánh giá hiệu suất của mô hình cho thấy một sự chênh lệch lớn giữa hai lớp dữ liệu:

- Lớp "False" (không lỗi): Precision cao (0.87), recall cao (0.85), và F1-score cao (0.86), cho thấy mô hình rất tốt trong việc phân loại các mẫu không phải là lỗi.

- Lớp "True" (lỗi): Precision thấp (0.41), recall thấp (0.46), và F1-score thấp (0.43), cho thấy mô hình không phân loại tốt các mẫu lỗi, điều này làm giảm độ chính xác tổng thể của mô hình.



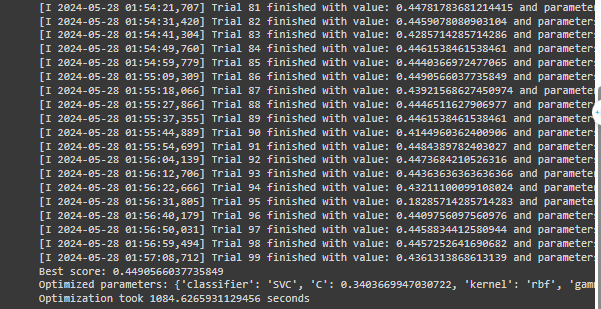
*HÌNH 4.1 Trước khi Tuning Model*

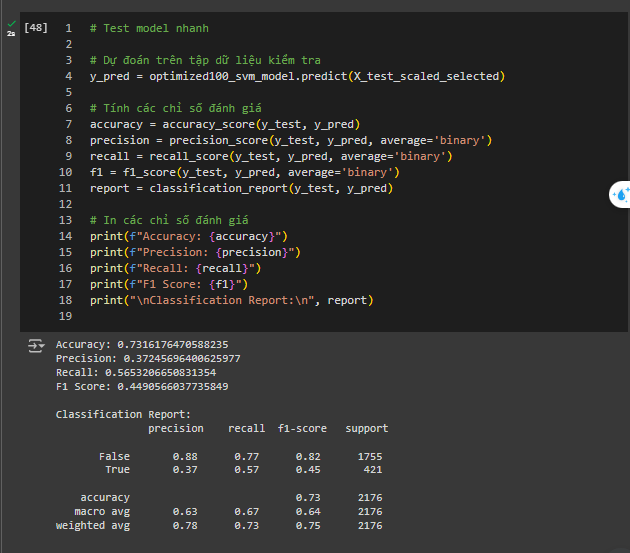
### 6.1.2 Sau khi tiến hành tinh chỉnh tham số:

Sau khi áp dụng kỹ thuật SMOTE để cân bằng dữ liệu và sử dụng thư viện Optuna để tinh chỉnh siêu tham số, em huấn luyện lại mô hình SVM. Kết quả đánh giá hiệu suất của mô hình sau tuning cho thấy sự cải thiện rõ rệt:

- Lớp "False" (không lỗi): Mặc dù vẫn duy trì precision và recall cao, F1-score của lớp này cũng được giữ vững.

- Lớp "True" (lỗi): Precision và recall được cải thiện đáng kể, điều này làm tăng F1-score và cải thiện độ chính xác tổng thể của mô hình.





*HÌNH 4.2 Sau khi Tuning Model*

### 6.1.3 Nhận xét chi tiết

- Trước khi Tuning:

- Mô hình SVM có khả năng phân loại tốt lớp không lỗi nhưng gặp khó khăn với lớp lỗi, dẫn đến thiên vị lớp đa số.

- Precision của lớp "True" (lỗi) khá thấp (0.39), và recall cũng không cao (0.43), khiến F1-score chỉ đạt 0.41. Điều này cho thấy mô hình bỏ sót nhiều trường hợp lỗi và có nhiều dự đoán sai.

- Sau khi Tuning:.

- Kết quả cho thấy mô hình SVM cải thiện đáng kể khả năng phân loại lỗi, với recall của lớp "True" tăng từ 0.43 lên 0.57, và F1-score tăng từ 0.41 lên 0.45. Precision của lớp "True" giảm nhẹ nhưng không quá đáng kể.

- Macro avg và weighted avg đều tăng, phản ánh sự cân bằng tốt hơn giữa precision và recall của hai lớp, nâng cao khả năng tổng quát hóa của mô hình.

- Việc áp dụng SMOTE để cân bằng dữ liệu và sử dụng Optuna để tối ưu siêu tham số đã cải thiện hiệu suất mô hình SVM đáng kể. Mô hình sau tuning phân loại tốt hơn cả các mẫu lỗi và không lỗi, nâng cao độ chính xác tổng thể. Điều này khẳng định tầm quan trọng của cân bằng dữ liệu và tối ưu hóa siêu tham số trong phát triển mô hình học máy hiệu quả cho phân loại lỗi phần mềm.

## 6.2 Kết luận

Trong nghiên cứu này, em đã triển khai và đánh giá mô hình phân loại lỗi phần mềm sử dụng kỹ thuật Support Vector Machine (SVM). SVM là một công cụ mạnh mẽ trong việc phân loại các mẫu dữ liệu phần mềm, đặc biệt là khi được tinh chỉnh và tối ưu hóa đúng cách.

Trước khi tinh chỉnh, mô hình SVM đã thể hiện khả năng phân loại tốt các mẫu không lỗi nhưng gặp khó khăn trong việc phát hiện các mẫu lỗi, dẫn đến sự thiên vị về lớp đa số. Kết quả này phản ánh vấn đề mất cân bằng dữ liệu, ảnh hưởng đến hiệu suất tổng thể của mô hình.

Sau khi áp dụng kỹ thuật SMOTE để cân bằng dữ liệu và sử dụng Optuna để tối ưu hóa siêu tham số, mô hình SVM đã có sự cải thiện đáng kể. Các chỉ số đánh giá như precision, recall, và F1-score của lớp lỗi đều tăng lên, cho thấy mô hình không chỉ phân loại tốt các mẫu không lỗi mà còn cải thiện khả năng phát hiện các mẫu lỗi.

Những kết quả thu được từ quá trình tinh chỉnh tham số cho thấy tầm quan trọng của việc lựa chọn đúng tham số và cân bằng dữ liệu trong việc phát triển các mô hình học máy hiệu quả. Sự cải thiện này không chỉ giúp mô hình học sâu hơn từ dữ liệu mà còn nâng cao khả năng dự đoán chính xác và phát hiện mẫu dữ liệu bất thường.

# CHƯƠNG 7. Hướng phát triển

- Hiệu suất mô hình hiện tại chưa cao, đặc biệt khi dự đoán các mẫu sản phẩm lỗi. Để cải thiện, em dự kiến 1 số hướng phát triển trong tương lai:

### 7.1 Tích hợp thêm yếu tố và nguồn dữ liệu đa dạng

- Thu thập dữ liệu phong phú: Từ các hệ thống và môi trường phần mềm khác nhau.

- Phân tích tương quan: Xác định mối liên hệ giữa lỗi phần mềm và các yếu tố khác như thời gian, loại ứng dụng.

### 7.2 Khám phá các kỹ thuật học máy mới và tiên tiến

- Khám phá phương pháp học máy tiên tiến: Như học tăng cường và học chuyển giao.

### 7.3 Tối ưu hóa và cập nhật mô hình dựa trên phản hồi thực tế

- Cập nhật mô hình: Học hỏi từ dữ liệu mới.

- Tối ưu tham số: Dựa trên hiệu suất thực tế để giảm báo động giả.

### 7.4 Phát triển giao diện người dùng trực quan:

- Thiết kế giao diện: Đơn giản, dễ sử dụng nhưng hiệu quả.

- Tối ưu trải nghiệm người dùng: Tích hợp tính năng dự đoán và cảnh báo tự động.

Mục tiêu là xây dựng hệ thống phân loại lỗi phần mềm mạnh mẽ, nâng cao chất lượng và khả năng dự đoán trong kỷ nguyên kỹ thuật số.

# **TÀI LIỆU THAM KHẢO**

[1] <https://viblo.asia/p/support-vector-machine-trong-hoc-may-mot-cai-nhin-don-gian-hon-ByEZkPbJ5Q0>

[2] <https://machinelearningcoban.com/>

[3] <https://www.ibm.com/cloud/learn/supervised-learning>

[4] <https://viblo.asia/p/svm-soft-margin-svm-kernel-svm-multi-class-svm-W13VMer0VY7>

[5] <https://tek4.vn/khoa-hoc/machine-learning-co-ban/kernel-support-vector-machine-kernel-svm>

[6] <http://www.columbia.edu/~mh2078/MachineLearningORFE/SVMs_MasterSlides.pdf>

[7] <https://www.youtube.com/watch?v=Q7vT0--5VII>

[8] <https://machinelearningcoban.com/2017/04/22/kernelsmv/>