Một chương trình AutoML hoàn chỉnh cho việc triển khai các mô hình máy học cổ điển trên vi điều khiển ESP32 và Arduino

Nguyễn Hữu Phước
(Can Tho University, Can Tho, Vietnam
https://orcid.org/0009-0006-5822-4121, huuphuoc.research@gmail.com)

Tóm tắt: Bài báo này trình bày P-ML, một khung AutoML đầu cuối dành cho việc triển khai các mô hình máy học cổ điển trên các vi điều khiển có giới hạn bộ nhớ. Khung này tự động hóa toàn bộ quy trình, bao gồm chia dữ liệu, lựa chọn mô hình, tối ưu siêu tham số và sinh các thư viện C++ tương thích với Arduino đã được tối ưu. P-ML tích hợp cơ chế tối ưu siêu tham số dựa trên Optuna với phương pháp chia dữ liệu phân tầng sử dụng thuật toán SPXY nhằm đảm bảo việc lựa chọn mô hình có độ tin cậy và tính ổn định cao. Khung đã được triển khai và kiểm chứng thành công, cho phép tự động tạo thư viện cho nhiều nhóm mô hình máy học cổ điển, bao gồm các biến thể của SVM (linear, RBF, polynomial, sigmoid, NuSVC, LinearSVC), SVM dựa trên RFF, các mô hình tổ hợp dựa trên cây quyết định, mạng perceptron đa lớp, k láng giềng gần nhất, phân tích phân biệt và các bộ phân loại Naive Bayes. Các thư viện được sinh ra có kích thước nhỏ gọn, hiệu quả và có thể triển khai trực tiếp trên các nền tảng Arduino Uno, Arduino Nano và ESP32 thông qua Arduino IDE.

Từ khóa: AutoML, máy học nhúng, hệ thống có tài nguyên hạn chế, sinh mã, khung phần mềm
Phân loại: B.4.2 (Hệ thống nhúng), C.3.3 (Hệ thống thời gian thực và hệ thống nhúng), I.2.6 (Học máy)
DOI: 10.5281/zenodo.18130146

1. Giới thiệu

Việc triển khai ngày càng rộng rãi các hệ thống Internet of Things IoT đã thúc đẩy nhu cầu lớn đối với suy luận máy học trực tiếp trên các thiết bị vi điều khiển như Arduino và ESP32. Các nền tảng này được sử dụng phổ biến trong mạng cảm biến, hệ thống đeo được và các ứng dụng trí tuệ biên, tuy nhiên chúng hoạt động dưới những ràng buộc nghiêm ngặt về bộ nhớ và năng lực tính toán. Trong những môi trường như vậy, các mô hình máy học cổ điển như Support Vector Machines, cây quyết định, các phương pháp tổ hợp và k láng giềng gần nhất thường cho hiệu năng tốt hơn các phương pháp học sâu đối với các tập dữ liệu cảm biến có kích thước nhỏ và trung bình, đồng thời yêu cầu ít tài nguyên tính toán hơn đáng kể.
Mặc dù có những ưu điểm này, việc triển khai các mô hình máy học cổ điển trên vi điều khiển vẫn còn nhiều thách thức. Các giải pháp máy học nhúng hiện nay либо dựa vào các môi trường thực thi mức cao gây ra chi phí lớn về bộ nhớ và hiệu năng, либо tập trung chủ yếu vào các mô hình học sâu, dẫn đến quy trình triển khai phức tạp và khả năng áp dụng hạn chế đối với các thiết bị có tài nguyên hạn chế. Bên cạnh đó, các công cụ hiện tại còn thiếu một quy trình tự động và có hệ thống nhằm kết nối việc phát triển máy học tiêu chuẩn trong

Python với triển khai C C plus plus gốc hiệu quả trên các nền tảng tương thích Arduino. Để khắc phục những hạn chế này, bài báo giới thiệu P-ML, một chương trình AutoML đầu cuối được thiết kế chuyên biệt cho việc triển khai các mô hình máy học cổ điển trên các hệ thống nhúng có giới hạn bộ nhớ. P ML tự động hóa toàn bộ quy trình, bao gồm chia dữ liệu, lựa chọn mô hình, tối ưu siêu tham số và sinh các thư viện C plus plus tương thích Arduino có kích thước nhỏ gọn.
Chương trình P-ML tích hợp cơ chế tối ưu siêu tham số dựa trên Optuna với phương pháp chia dữ liệu phân tầng sử dụng thuật toán SPXY nhằm đảm bảo việc đánh giá mô hình có độ tin cậy và phù hợp với triển khai thực tế. Một phạm vi rộng các thuật toán máy học cổ điển được hỗ trợ, và các mô hình đã huấn luyện được tự động chuyển đổi thành mã C plus plus tối ưu, phù hợp cho các nền tảng Arduino Uno, Arduino Nano và ESP32. Toàn bộ chu trình được thống nhất trong một giao diện người dùng trực quan, giúp loại bỏ nhu cầu cấu hình thủ công hoặc kiến thức chuyên sâu về hệ thống nhúng.

2. Kiến trúc của chương trình P-ML

2.1 Thuật toán chia dữ liệu

Chương trình P-ML áp dụng chiến lược chia dữ liệu dạng hold out dựa trên thuật toán SPXY cho quá trình huấn luyện và đánh giá mô hình. Phương pháp này được thiết kế nhằm tạo ra các tập huấn luyện và kiểm tra độc lập, đồng thời vẫn bảo toàn phân bố lớp và tối đa hóa tính đa dạng của dữ liệu trong tập huấn luyện.

Với SPXY, tập dữ liệu được chia thành các tập con huấn luyện và kiểm tra theo một tỷ lệ xác định trước, ví dụ như 70 phần trăm cho huấn luyện và 30 phần trăm cho kiểm tra. Thuật toán được áp dụng độc lập trong từng lớp nhằm duy trì sự cân bằng nhãn. Các mẫu huấn luyện được lựa chọn sao cho có khoảng cách lớn nhất với nhau trong không gian đặc trưng, từ đó tăng mức độ bao phủ của miền đầu vào và cải thiện khả năng khái quát hóa của mô hình.

Ví dụ, xét một tập dữ liệu gồm 150 mẫu thuộc 3 lớp, mỗi lớp có 50 mẫu, với tỷ lệ huấn luyện là 70 phần trăm. Thuật toán SPXY sẽ chọn 35 mẫu mỗi lớp cho tập huấn luyện và 15 mẫu mỗi lớp cho tập kiểm tra, kết quả thu được là 105 mẫu huấn luyện và 45 mẫu kiểm tra độc lập. Mô hình được huấn luyện một lần duy nhất trên tập huấn luyện và được đánh giá trên tập kiểm tra, qua đó phản ánh sát điều kiện triển khai thực tế khi mô hình phải xử lý các dữ liệu chưa từng xuất hiện trước đó.

Để tránh hiện tượng rò rỉ thông tin, quá trình chuẩn hóa đặc trưng chỉ được thực hiện trên tập huấn luyện. Các thống kê thu được như giá trị trung bình và hệ số tỷ lệ sau đó được áp dụng cho tập kiểm tra và được nhúng trực tiếp vào thư viện vi điều khiển được sinh ra. Cách làm này đảm bảo tính nhất quán giữa các giai đoạn huấn luyện, đánh giá và suy luận trực tiếp trên thiết bị.

So với phương pháp chia hold out ngẫu nhiên, SPXY cung cấp một tập huấn luyện mang tính đại diện và đa dạng hơn, đặc biệt đối với các tập dữ liệu có kích thước nhỏ đến trung bình. Mặc dù phương pháp này không tái sử dụng các

mẫu kiểm tra cho huấn luyện, nhưng nó mang lại một quy trình đánh giá rõ ràng và định hướng triển khai, phù hợp với các ứng dụng nhúng và IoT, nơi các mô hình thường chỉ được huấn luyện một lần và sau đó được triển khai mà không cần huấn luyện lại.

2.2 Tối ưu siêu tham số

Chương trình P-ML sử dụng Optuna để tự động tìm kiếm và lựa chọn các siêu tham số tối ưu cho từng mô hình máy học. Thay vì kiểm thử thủ công một tập giá trị định trước với số lượng hạn chế, Optuna đánh giá hàng chục tổ hợp tham số theo cách có định hướng và xác định cấu hình mang lại hiệu năng tốt nhất.

Ví dụ, đối với mô hình SVM, thay vì cố định các tham số như C = 1.0 và kernel = rbf, hệ thống sẽ khảo sát nhiều giá trị khác nhau, chẳng hạn như C ∈ {0.1, 1, 10, 50}, các loại kernel khác nhau và các thiết lập gamma tương ứng, sau đó lựa chọn tổ hợp tham số tối đa hóa độ chính xác. Trong thực tế, cách tiếp cận này thường mang lại cải thiện hiệu năng đáng kể so với việc tinh chỉnh thủ công, ví dụ nâng độ chính xác từ khoảng 80 phần trăm lên mức 85 đến 90 phần trăm.

Optuna hoạt động thông qua các lần thử, trong đó mỗi lần thử tương ứng với một cấu hình siêu tham số cụ thể. Tất cả các lần thử được quản lý trong một nghiên cứu, nơi kết quả được ghi nhận và so sánh nhằm xác định bộ tham số có hiệu năng tốt nhất. Quá trình tối ưu được dẫn dắt bởi bộ lấy mẫu Tree structured Parzen Estimator, có khả năng học từ các lần thử trước đó để ưu tiên những vùng tiềm năng của không gian tìm kiếm, thay vì dựa vào tìm kiếm ngẫu nhiên hoặc tìm kiếm lưới toàn diện. So với tìm kiếm ngẫu nhiên vốn chậm và thiếu định hướng, cũng như tìm kiếm lưới vốn tốn kém về tính toán và mang tính thô, TPE cho tốc độ hội tụ nhanh hơn và hiệu quả cao hơn.

Để giảm thời gian huấn luyện, chương trình tích hợp cơ chế Median Pruning, cho phép kết thúc sớm các lần thử có hiệu năng kém. Trong quá trình đánh giá chéo, nếu hiệu năng của một lần thử sau các lần chia ban đầu kém hơn đáng kể so với giá trị trung vị của các lần thử trước đó, lần thử này sẽ bị loại bỏ và quá trình huấn luyện được dừng lại. Ví dụ, nếu hầu hết các cấu hình đạt độ chính xác khoảng 75 đến 80 phần trăm, thì một lần thử chỉ đạt 45 phần trăm sau lần chia đầu tiên sẽ bị loại bỏ ngay lập tức nhằm tránh các phép tính không cần thiết.

Mỗi lần thử được đánh giá bằng phương pháp đánh giá chéo k lần chia, ví dụ k bằng 5. Đối với một tập tham số được đề xuất, mô hình được huấn luyện và đánh giá trên năm lần chia khác nhau, và điểm số cuối cùng của lần thử được tính bằng giá trị trung bình của độ chính xác trên các lần chia đó. Nếu một lần thử bị loại bỏ sớm, điểm số của nó chỉ dựa trên các lần chia đã hoàn thành. Chẳng hạn, một lần thử có thể đạt các độ chính xác lần lượt là 80 phần trăm, 82 phần trăm, 85 phần trăm, 83 phần trăm và 84 phần trăm, cho giá trị trung bình là 82.8 phần trăm, trong khi một lần thử khác có thể bị dừng sau lần chia đầu tiên do hiệu năng kém.

So với việc lựa chọn tham số thủ công, cách tiếp cận này mang tính khách quan, tự động và hiệu quả hơn, đồng thời giảm nguy cơ bỏ sót các tổ hợp tham số tiềm năng. So với tìm kiếm lưới, Optuna làm giảm đáng kể số lần huấn luyện cần thiết thông qua chiến lược TPE và cơ chế loại bỏ sớm. Hạn chế chính của phương pháp này là chi phí tính toán ban đầu cao hơn so với huấn luyện với tham số cố định, do mỗi mô hình phải được huấn luyện nhiều lần.

Trong cấu hình hiện tại, mỗi mô hình được tối ưu bằng 50 lần thử kết hợp với đánh giá chéo 5 lần chia, các tham số này có thể được người dùng cấu hình. Điều này tương ứng với tối đa 250 lần huấn luyện. Tuy nhiên, thời gian chạy thực tế được giảm đáng kể nhờ cơ chế loại bỏ sớm và các ràng buộc về thời gian. Kết quả cuối cùng của quá trình này là tập siêu tham số tốt nhất cùng với độ chính xác cao nhất, sau đó được sử dụng để huấn luyện lại mô hình và sinh thư viện C cộng cộng tối ưu cho triển khai trên vi điều khiển.

2.3 Sinh thư viện cho vi điều khiển

Quá trình chuyển đổi các mô hình scikit learn trong Python sang các thư viện C hoặc C cộng cộng cho ESP32 và Arduino Mega được thực hiện bằng cách trích xuất trực tiếp các tham số bên trong của mô hình đã huấn luyện và ánh xạ chúng sang các cấu trúc dữ liệu tĩnh trong C hoặc C cộng cộng. Mục tiêu của bước này là đảm bảo quá trình suy luận trên vi điều khiển tái hiện chính xác các phép toán toán học giống như mô hình gốc trong Python.
Đối với tất cả các mô hình, bước đầu tiên là trích xuất các tham số của StandardScaler đã được huấn luyện trên tập huấn luyện. Cụ thể, các mảng scaler.mean_ và scaler.scale_ được xuất ra dưới dạng các mảng số thực hằng trong C hoặc C cộng cộng. Trong quá trình suy luận, mỗi vector đặc trưng đầu vào từ cảm biến được chuẩn hóa bằng cùng công thức như trong Python:

$$x' = (x - \mu) / \sigma$$

Trong đó μ và σ lần lượt là các giá trị trung bình và hệ số tỷ lệ đã được lưu trữ. Điều này đảm bảo rằng phân bố dữ liệu đầu vào trong giai đoạn triển khai phù hợp với giai đoạn huấn luyện. Ngoài ra, danh sách các lớp, tên lớp và thứ tự đặc trưng cũng được mã hóa cứng trong thư viện nhằm đảm bảo ánh xạ nhãn chính xác và tránh sai lệch dữ liệu đầu vào.

Đối với các mô hình SVM sử dụng kernel như SVC và NuSVC, chương trình trích xuất các thuộc tính quan trọng từ mô hình scikit learn, bao gồm support_vectors_, dual_coef_, intercept_ và các tham số kernel như kernel, gamma, degree và coef0. Các vector hỗ trợ được lưu trữ dưới dạng các mảng hai chiều hoặc mảng phẳng trong C hoặc C cộng cộng, trong khi các hệ số song song và giá trị chệch được lưu trữ dưới dạng các trọng số và độ lệch tương ứng cho từng bộ phân loại one vs one. Trong quá trình suy luận, hàm dự đoán trên thiết bị nhúng tái hiện phép tính trong Python, trong đó đối với mỗi vector hỗ trợ, hàm kernel được tính toán, ví dụ như kernel RBF với biểu thức exp(−γ nhân chuẩn Euclid bình phương giữa x và x_i), sau đó được nhân với hệ số song song, cộng với giá trị chệch và tổng hợp thông qua cơ chế bỏ phiếu để lựa chọn lớp có điểm số cao nhất.

Đối với LinearSVC, quá trình đơn giản hơn do chỉ cần trích xuất các tham số coef_ và intercept_. Mã C hoặc C cộng cộng được sinh ra sẽ tính tích vô hướng giữa vector đầu vào và các trọng số tuyến tính, tạo ra một mô hình gọn nhẹ và tiết kiệm bộ nhớ, phù hợp với các thiết bị có tài nguyên hạn chế.

Trong các mô hình cây quyết định, các thuộc tính như tree_.feature, tree_.threshold, tree_.children_left, tree_.children_right và tree_.value được trích xuất trực tiếp từ scikit learn. Các mảng này được chuyển đổi thành các cấu trúc dữ liệu hằng trong C hoặc C cộng cộng và được sử dụng để tái hiện logic duyệt cây giống như trong Python. Trong quá trình suy luận, chương trình trên vi điều khiển bắt đầu từ nút gốc, so sánh đặc trưng tương ứng với ngưỡng, đi theo nhánh trái hoặc phải tương ứng và dừng lại tại nút lá, nơi lớp có tần suất hoặc xác suất cao nhất trong tree_.value được chọn.

Đối với MLP hay mạng nơ ron, chương trình trích xuất toàn bộ danh sách các ma trận coefs_ và các vector intercepts_, tương ứng với trọng số và độ lệch của từng lớp trong mạng. Các ma trận này được lưu trữ dưới dạng mảng C hoặc C cộng cộng theo đúng thứ tự các lớp. Loại hàm kích hoạt như relu, tanh hoặc logistic cũng được mã hóa cứng. Trong quá trình suy luận, mỗi lớp thực hiện phép nhân ma trận vector, cộng độ lệch và áp dụng hàm kích hoạt tương ứng. Việc bảo toàn đúng thứ tự các lớp và hàm kích hoạt đảm bảo rằng kết quả trên vi điều khiển khớp với mô hình Python.

Đối với KNN, do mô hình không học tham số, chương trình trích xuất trực tiếp tập X_train và y_train hoặc một tập con đã được rút gọn từ Python và lưu trữ trong thư viện C hoặc C cộng cộng. Trong quá trình suy luận, chương trình tính khoảng cách, thường là khoảng cách Euclid, giữa mẫu đầu vào và các mẫu đã lưu, lựa chọn k láng giềng gần nhất và áp dụng cơ chế bỏ phiếu đa số. Trên Arduino Mega, số lượng mẫu được lưu có thể bị giới hạn để phù hợp với ràng buộc bộ nhớ, trong khi ESP32 thường có thể lưu trữ toàn bộ tập dữ liệu.

Đối với Naive Bayes, tất cả các tham số thống kê được trích xuất từ Python, bao gồm theta_ là giá trị trung bình, var_ là phương sai và class_prior_ đối với GaussianNB, hoặc các bảng log xác suất đối với BernoulliNB. Các giá trị này được ánh xạ sang các mảng hằng trong C hoặc C cộng cộng, và quá trình suy luận bao gồm việc tính toán log likelihood theo các công thức thống kê tiêu chuẩn và lựa chọn lớp có xác suất hậu nghiệm cao nhất. Tương tự, đối với LDA và QDA, các giá trị trung bình theo lớp, xác suất tiên nghiệm và các ma trận hiệp phương sai, chung cho LDA và riêng cho từng lớp đối với QDA, được trích xuất và sử dụng để tính toán các hàm phân biệt tuyến tính hoặc bậc hai trong chương trình trên thiết bị.

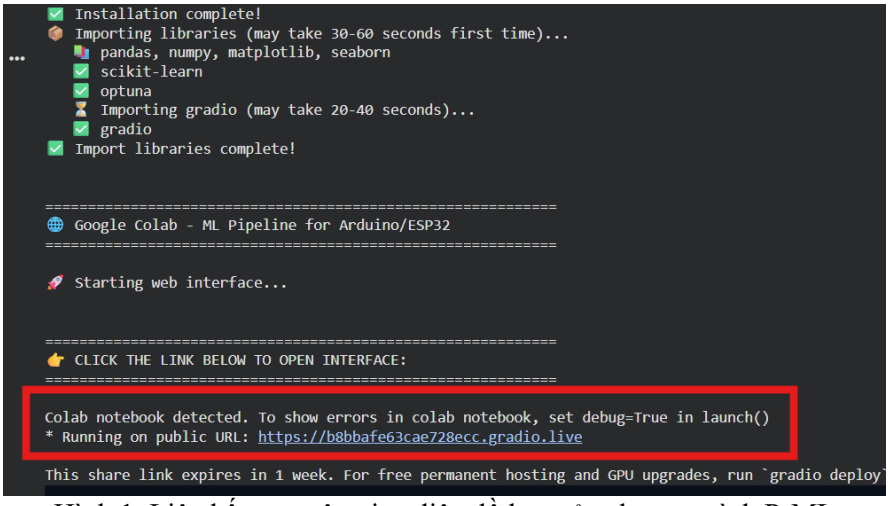
Tóm lại, toàn bộ quá trình chuyển đổi không dựa trên xấp xỉ hay huấn luyện lại. Thay vào đó, chương trình thực hiện việc chuyển giao có kiểm soát các tham số toán học từ Python sang C hoặc C cộng cộng và tái hiện trung thực các phép toán suy luận ban đầu. Do đó, các thư viện được sinh ra cho ESP32 và Arduino Mega thể hiện hành vi dự đoán nhất quán với các mô hình scikit learn gốc, đồng

thời vẫn cho phép tối ưu bộ nhớ và hiệu năng phù hợp với từng nền tảng phần cứng.

3. Hướng dẫn sử dụng

3.1 Chạy chương trình trên Google Colab

Để bắt đầu, hãy truy cập mã nguồn của chương trình P-ML thông qua liên kết Google Colab được cung cấp [https://colab.research.google.com/drive/1G3PQG_6mmQgMDnm88vhXh-duST0GOgO2].
Sau khi mở, hãy lưu một bản sao cá nhân của notebook vào Google Drive của bạn để có thể chỉnh sửa và thực thi chương trình.
Sau khi lưu, chỉ cần chạy lần lượt các ô lệnh có sẵn. Notebook sẽ tự động cài đặt tất cả các thư viện phụ thuộc cần thiết. Khi quá trình thiết lập hoàn tất, một đường dẫn truy cập đến giao diện người dùng đồ họa sẽ được hiển thị trong phần kết quả. Nhấp vào liên kết này để mở giao diện web của P-ML trong một thẻ trình duyệt mới và bắt đầu làm việc với hệ thống.



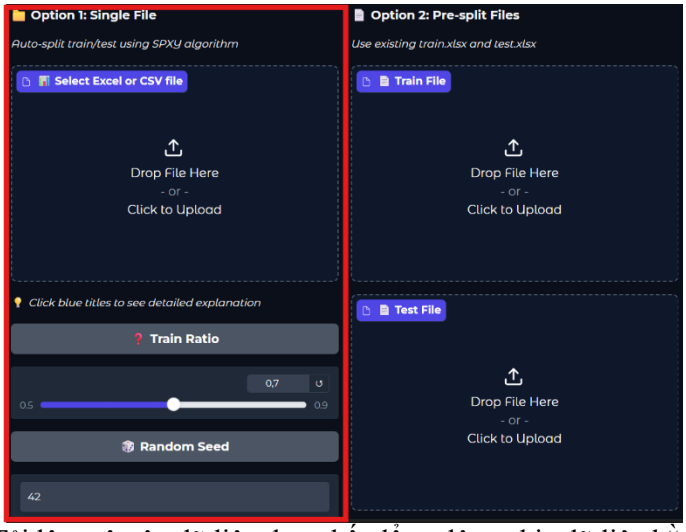
Hình 1. Liên kết truy cập giao diện đồ họa của chương trình P-ML

3.2 Tải lên tập dữ liệu

Yêu cầu đối với tệp dữ liệu
Tập dữ liệu được tải lên phải thỏa mãn các điều kiện sau:
Tệp dữ liệu phải ở định dạng Excel (.xlsx).
Hàng đầu tiên phải chứa tên các cột.
Mỗi đặc trưng phải được lưu trữ trong một cột riêng biệt.
Bắt buộc phải có một cột có tên là Class, cột này chứa nhãn chuẩn của dữ liệu.
Nếu cột này bị thiếu, hệ thống sẽ báo lỗi.
Tên các cột đặc trưng có thể tùy ý và do người dùng tự định nghĩa.
Một tệp dữ liệu mẫu được cung cấp tại liên kết tham chiếu

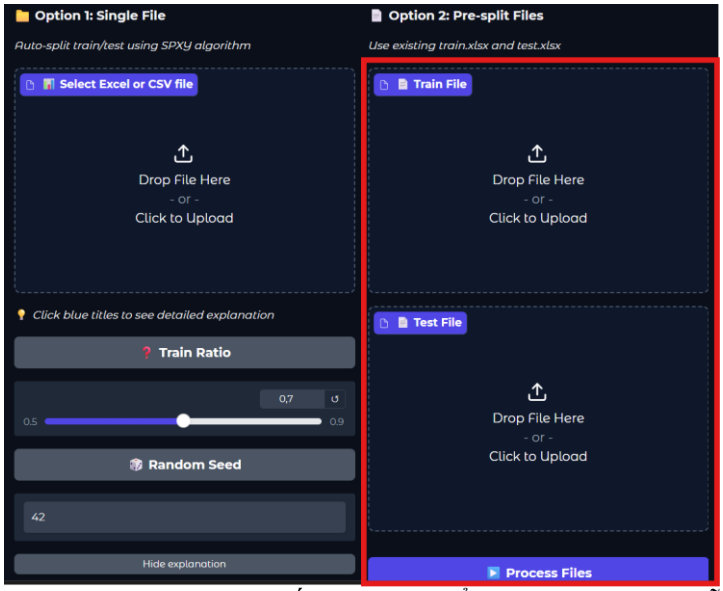
[https://raw.githubusercontent.com/HuuPhuoc2411/P-ML/main/DATA_SAMPLE.xlsx].

Các tùy chọn tải dữ liệu
Chương trình P-ML hỗ trợ hai kịch bản tải dữ liệu khác nhau.



Hình 2. Tải lên một tập dữ liệu duy nhất để tự động chia dữ liệu bằng SPXY

Nếu dữ liệu của bạn chưa được chia sẵn, hãy tải lên một tệp Excel duy nhất chứa toàn bộ tập dữ liệu. Hệ thống sẽ tự động áp dụng thuật toán SPXY để chia dữ liệu thành tập huấn luyện và tập kiểm tra với tỷ lệ mặc định là 70 phần trăm cho huấn luyện và 30 phần trăm cho kiểm tra, tỷ lệ này có thể được người dùng điều chỉnh.
Các tập dữ liệu huấn luyện và kiểm tra được tạo ra cũng sẽ được lưu dưới dạng các tệp Excel riêng biệt, cho phép tái sử dụng trong các thí nghiệm trong tương lai.



Hình 3. Tải lên các tệp huấn luyện và kiểm tra đã được chia sẵn

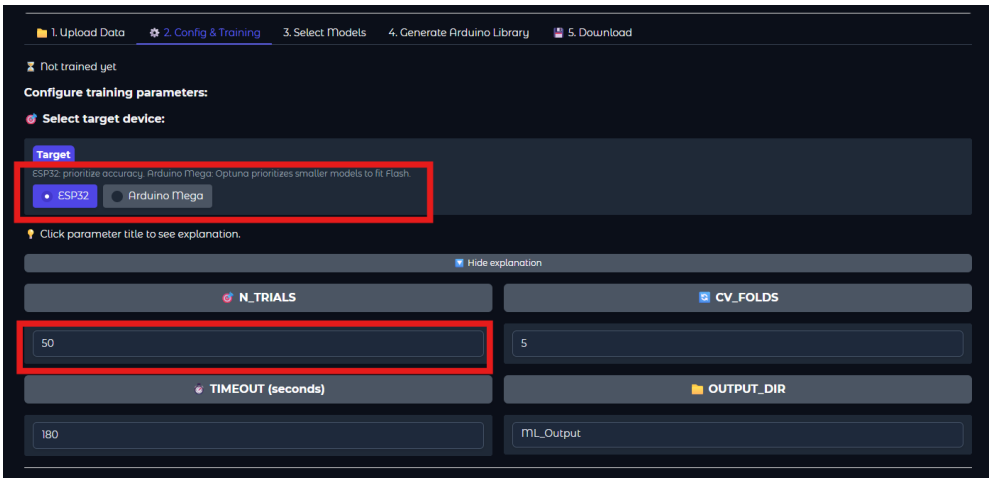
Nếu bạn đã chuẩn bị sẵn các tệp huấn luyện và kiểm tra riêng biệt, hãy tải chúng trực tiếp bằng các trường tương ứng trong Tùy chọn 2 là các tệp đã được chia sẵn.
Sau khi tải lên, hệ thống sẽ hiển thị các biểu đồ phân bố lớp cho cả tập huấn luyện và tập kiểm tra. Các biểu đồ này cho phép người dùng kiểm tra trực quan xem tỷ lệ các lớp có được cân bằng giữa hai tập hay không. Lý tưởng nhất, các phân bố tương tự nhau cho thấy quá trình chia dữ liệu là thành công và đáng tin cậy.
Sau khi hoàn tất bước này, người dùng cần chủ động chuyển sang giai đoạn tiếp theo để lựa chọn và huấn luyện các mô hình.

3.3 Lựa chọn mô hình và huấn luyện

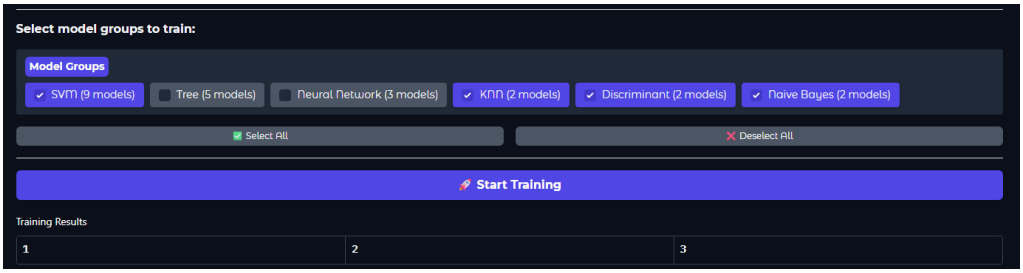
Lựa chọn cấu hình đầu tiên là xác định nền tảng đích để sinh thư viện, bao gồm ESP32 hoặc Arduino. Lựa chọn này sẽ quyết định các ràng buộc về bộ nhớ cũng như các chiến lược tối ưu được áp dụng ở các bước tiếp theo của quy trình.

- Tiếp theo, người dùng cấu hình các tham số huấn luyện gồm: Số lần thử cho mỗi mô hình, tức là số lần mỗi mô hình được huấn luyện với các cấu hình siêu tham số khác nhau.
- Thời gian giới hạn, là thời gian huấn luyện tối đa cho một mô hình. Nếu đạt đến giới hạn này trước khi hoàn thành tất cả các lần thử, quá trình sẽ dừng lại và bộ tham số tốt nhất tìm được cho đến thời điểm đó sẽ được giữ lại.
- Số lần chia dữ liệu dùng nội bộ trong quá trình huấn luyện để ước lượng hiệu năng.
- Tên thư mục đầu ra, là tên thư mục nơi tất cả các kết quả và tệp được sinh ra sẽ được lưu trữ.

Sau đó, người dùng lựa chọn các mô hình cần huấn luyện trong mục lựa chọn nhóm mô hình. Có thể chọn toàn bộ các nhóm mô hình có sẵn hoặc chỉ chọn một số thuật toán cụ thể tùy theo nhu cầu.
Cuối cùng, quá trình huấn luyện được khởi động bằng cách nhấn nút Train. Tùy thuộc vào kích thước tập dữ liệu, số lượng mô hình được chọn và số lần huấn luyện, quá trình này có thể mất một khoảng thời gian đáng kể để hoàn thành.



Hình 4. Giao diện cấu hình huấn luyện



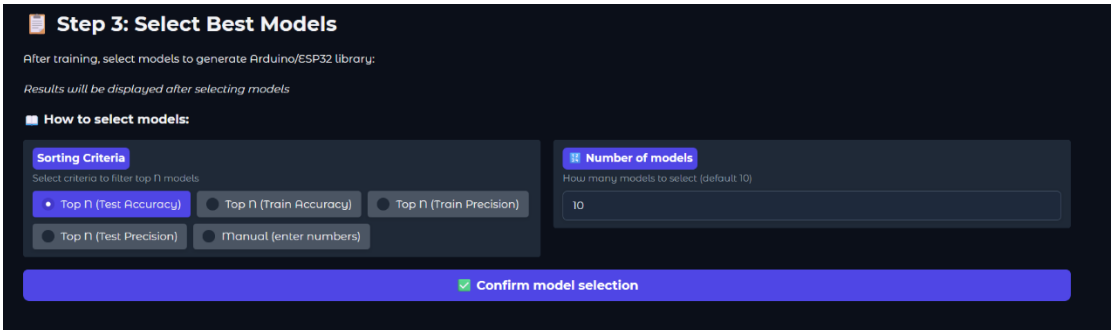
Hình 5. Giao diện lựa chọn nhóm mô hình

3.4 Lựa chọn các mô hình tốt nhất để sinh thư viện

Sau khi quá trình huấn luyện hoàn tất, hệ thống sẽ hiển thị danh sách các mô hình đã được huấn luyện kèm theo các chỉ số hiệu năng tương ứng. Tại giai đoạn này, người dùng có thể lựa chọn những mô hình sẽ được xuất ra dưới dạng thư viện nhúng.

Hệ thống cung cấp bảy tiêu chí lọc khác nhau nhằm hỗ trợ việc xác định các mô hình có hiệu năng tốt nhất. Ngoài ra, người dùng có thể chỉ định số lượng mô hình cần lựa chọn, với giá trị mặc định là mười mô hình.

Bên cạnh đó, một tùy chọn lựa chọn thủ công cũng được cung cấp. Khi kích hoạt, người dùng có thể chủ động chọn các mô hình cụ thể bằng cách nhập chỉ số tương ứng của chúng trong danh sách hiển thị, bỏ qua cơ chế xếp hạng tự động dựa trên hiệu năng. Tùy chọn này đặc biệt hữu ích trong các trường hợp người dùng ưu tiên một số mô hình nhất định vì lý do thực tiễn hoặc mục đích thí nghiệm.



Hình 6. Giao diện lọc mô hình và lựa chọn thủ công

3.5 Tạo thư viện nhúng

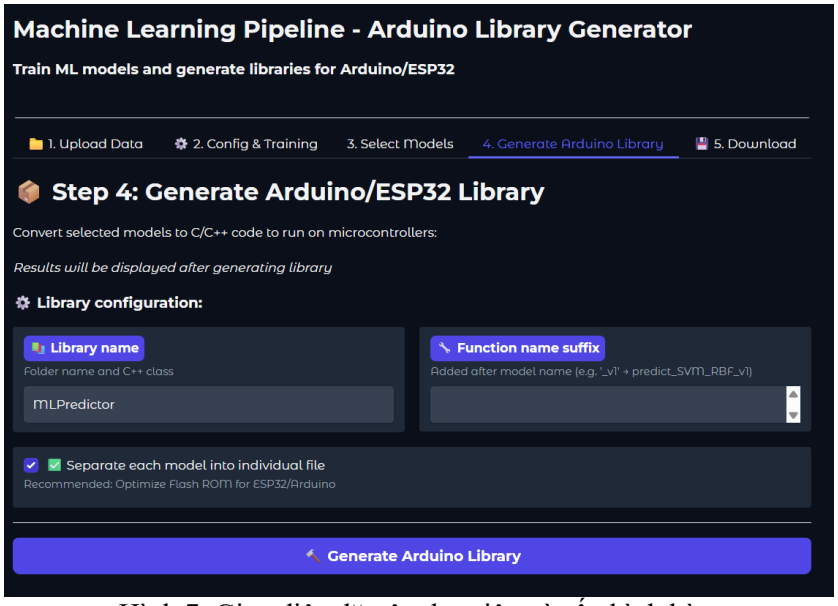
Sau khi các mô hình mong muốn đã được lựa chọn, người dùng chuyển sang trang Generate Arduino Library.

Tại đây, người dùng cần chỉ định các thông tin sau:

Tên thư viện.

Hậu tố tùy chỉnh cho các hàm dự đoán.

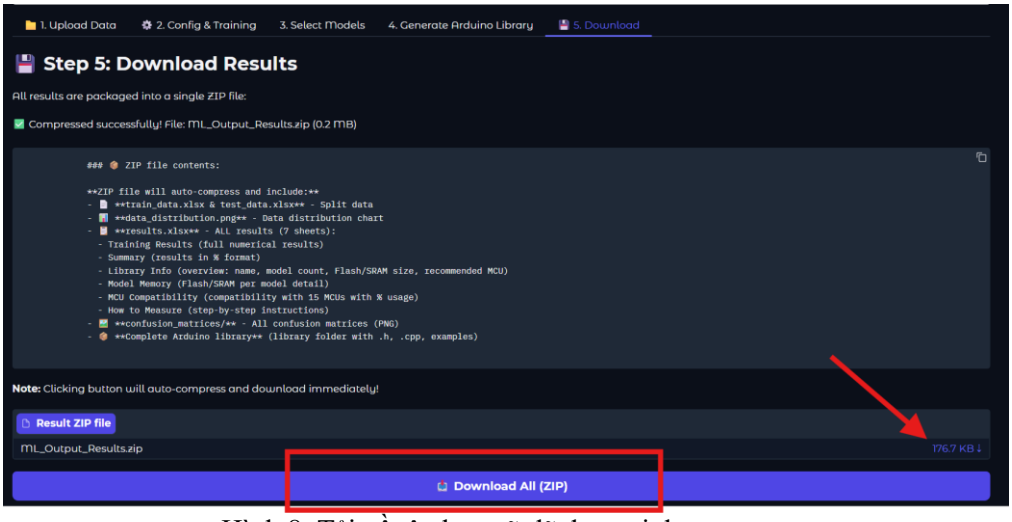
Mỗi hàm dự đoán được sinh ra sẽ tự động được đặt tên dựa trên tên mô hình gốc kết hợp với hậu tố do người dùng định nghĩa, nhằm đảm bảo tính rõ ràng và tránh xung đột tên.
Sau khi hoàn tất cấu hình, nhấn nút Generate để khởi động quá trình sinh thư viện. Quá trình này sẽ chuyển đổi các mô hình đã chọn thành mã C hoặc C cộng cộng được tối ưu, phù hợp với nền tảng vi điều khiển đã được lựa chọn.



Hình 7. Giao diện đặt tên thư viện và cấu hình hàm

3.6 Tải về kết quả

Ở bước cuối cùng, người dùng có thể tải về toàn bộ các kết quả đã được sinh ra. Khi nhấn nút Download, toàn bộ thư mục kết quả sẽ được nén lại thành một tệp lưu trữ duy nhất ở định dạng RAR.
Do đặc thù của giao diện Google Colab, người dùng có thể cần nhấp lại vào vị trí tải xuống như được hiển thị trên giao diện để hoàn tất quá trình chuyển tệp về máy tính cá nhân.



Hình 8. Tải về tệp lưu trữ đã được sinh ra

3.7 Cấu trúc tệp đầu ra

Tệp lưu trữ được tải về bao gồm các thành phần sau:

- Một tệp Excel tóm tắt kết quả huấn luyện và hiệu năng của các mô hình.
- Một tệp ảnh PNG thể hiện phân bố của tập dữ liệu.
- Một thư mục chứa các ảnh ma trận nhầm lẫn cho tất cả các mô hình đã được huấn luyện.
- Một thư mục chứa các thư viện nhúng đã được sinh ra cùng với mã ví dụ cho Arduino IDE.
- Nếu phương pháp chia dữ liệu SPXY được sử dụng, sẽ có thêm hai tệp Excel chứa tập dữ liệu huấn luyện và tập dữ liệu kiểm tra.

Cấu trúc đầu ra này cho phép người dùng dễ dàng kiểm tra, tái sử dụng và triển khai các mô hình đã huấn luyện trên các nền tảng nhúng khác nhau.

4. Thảo luận và hạn chế

Kết quả thực nghiệm cho thấy các thư viện được sinh ra cho nền tảng ESP32 luôn đạt độ chính xác dự đoán cao hơn so với các thư viện triển khai trên các vi điều khiển dựa trên Arduino. Sự khác biệt về hiệu năng này chủ yếu xuất phát từ các ràng buộc phần cứng. Các thiết bị ESP32 cung cấp dung lượng bộ nhớ lớn hơn đáng kể, cho phép các mô hình được triển khai với đầy đủ tập tham số và không cần đơn giản hóa mạnh. Ngược lại, các nền tảng Arduino, đặc biệt là Arduino Mega, chịu những giới hạn nghiêm ngặt về cả bộ nhớ SRAM và Flash. Do đó, nhiều mô hình buộc phải được đơn giản hóa thông qua việc giảm số lượng tham số, cắt tỉa hoặc tối ưu cấu trúc để có thể phù hợp với tài nguyên sẵn có, điều này có thể dẫn đến sự suy giảm đáng kể về hiệu năng dự đoán.

Bên cạnh các ràng buộc về bộ nhớ, hiệu năng tính toán cũng là một hạn chế quan trọng trên các nền tảng Arduino. Cụ thể, các biến thể SVM tuyến tính như SVC_linear hoặc SVM_linear, mặc dù có dấu chân bộ nhớ tương đối nhỏ, nhưng thường cho tốc độ suy luận rất chậm trên các vi điều khiển Arduino. Nguyên nhân là do việc phải tính toán tích vô hướng giữa vector đầu vào và các

vector trọng số có số chiều cao cho từng lớp, điều này trở nên tốn kém về mặt tính toán trên các bộ xử lý có tần số thấp và không có tăng tốc phần cứng. Do đó, ngay cả những mô hình đáp ứng được yêu cầu về bộ nhớ cũng có thể không thỏa mãn được các yêu cầu suy luận thời gian thực trên các thiết bị Arduino.
Để giảm thiểu những hạn chế này trên các nền tảng Arduino bị giới hạn về bộ nhớ và khả năng tính toán, khuyến nghị không nên phụ thuộc vào đầu ra của một mô hình duy nhất. Thay vào đó, có thể áp dụng chiến lược bỏ phiếu theo kiểu tổ hợp, trong đó nhiều mô hình nhẹ được triển khai song song và quyết định cuối cùng được xác định dựa trên kết quả bỏ phiếu đa số từ các dự đoán của chúng. Ví dụ, việc triển khai một tập gồm mười mô hình đã được đơn giản hóa và lựa chọn lớp được dự đoán nhiều nhất có thể cải thiện đáng kể độ ổn định và độ chính xác tổng thể, qua đó bù đắp cho sự suy giảm hiệu năng do việc đơn giản hóa từng mô hình riêng lẻ. Mặc dù cách tiếp cận này chủ yếu xuất phát từ các ràng buộc của Arduino, nó cũng hoàn toàn có thể áp dụng cho các nền tảng ESP32 và góp phần nâng cao tính ổn định và độ tin cậy trong các triển khai IoT thực tế.
Cuối cùng, cần lưu ý rằng kích thước của thư viện nhúng được sinh ra không phải là cố định. Dung lượng bộ nhớ sử dụng tăng theo cả số lượng mẫu huấn luyện, chẳng hạn như các vector hỗ trợ trong SVM hoặc các mẫu được lưu trữ trong KNN, và số lượng lớp đầu ra, đặc biệt đối với các bộ phân loại đa lớp sử dụng chiến lược one vs one hoặc one vs rest. Do đó, người dùng cần cân nhắc cẩn thận kích thước tập dữ liệu, số lượng lớp và độ phức tạp của mô hình khi hướng tới các nền tảng có tài nguyên rất hạn chế như Arduino Mega, bởi việc vượt quá các giới hạn này có thể dẫn đến lỗi biên dịch, suy giảm hiệu năng khi chạy hoặc yêu cầu phải tiếp tục đơn giản hóa mô hình.

5. Giấy phép và tuyên bố bản quyền

Chương trình được đề xuất cùng với các thư viện Arduino và ESP32 được sinh ra chỉ được phát hành cho mục đích học thuật, nghiên cứu và giáo dục.
Người dùng được phép sử dụng, chỉnh sửa và sao chép mã nguồn cũng như các thư viện được sinh ra cho mục đích phi thương mại, với điều kiện phải trích dẫn đầy đủ công trình này trong mọi ấn phẩm, báo cáo kỹ thuật hoặc nghiên cứu phái sinh có liên quan.
Mọi hình thức sử dụng cho mục đích thương mại, phân phối vì lợi nhuận, cấp phép lại hoặc tích hợp vào các hệ thống độc quyền hay thương mại đều bị nghiêm cấm nếu không có sự cho phép bằng văn bản trước từ các tác giả.
Công trình này được lưu trữ và công bố công khai với định danh bền vững DOI: https://doi.org/10.5281/zenodo.18130146

Các thư viện hoặc phần phụ thuộc của bên thứ ba được sử dụng trong chương trình này vẫn tuân theo các giấy phép tương ứng của chúng.