TRƯỜNG ĐẠI HỌC XÂY DỰNG HÀ NỘI

# KHOA CÔNG NGHỆ THÔNG TIN

## BỘ MÔN HỌC MÁY



BÁO CÁO BÀI TẬP LỚN

# Đề tài: Phát hiện mã độc

**Giảng viên hướng dẫn:** Đào Xuân Trường

**Lớp học phần:** 66PM5

**Nhóm thực hiện:** Nhóm 7

Phạm Công Quân 0201366

Nguyễn Anh Tú 0207566

An Đăng Vinh 0209266

Trần Văn Long 0194866

Nguyễn Thành Trung 0206966

Hà Nội, ngày 21 tháng 10 năm 2024

# Mục lục

1. Giới thiệu

2. Mô tả dữ liệu

2.1. Các thuộc tính

2.2. Phân tích sơ bộ dữ liệu

3. Phân tích dữ liệu

4. Mô hình hóa

4.1 Tiền xử lý dữ liệu

4.2 Chia tập Dữ liệu

4.3 Chọn và đánh giá mô hình

5. Dự đoán dữ liệu kiểm tra và gửi kết quả

6. Ưu điểm

7. Nhược điểm

8. Kết luận

9. Tài liệu tham khảo

# Giới thiệu

* Ngày nay, với sự phát triển mạnh mẽ của công nghệ thông tin và internet, mã độc (malware) đã trở thành một trong những mối đe dọa nghiêm trọng đối với an ninh mạng. Mã độc có thể xâm nhập và phá hoại hệ thống, gây mất dữ liệu, rò rỉ thông tin nhạy cảm, và gây ra những thiệt hại lớn về tài chính và uy tín cho các cá nhân và tổ chức. Do đó, việc phát hiện và ngăn chặn mã độc kịp thời là vô cùng quan trọng.
* Trong lĩnh vực học máy, phát hiện mã độc dựa trên phân tích dữ liệu đã trở thành một hướng nghiên cứu phổ biến và đầy triển vọng. Với khả năng học hỏi từ dữ liệu lịch sử, các mô hình học máy có thể tự động phân loại các tệp hoặc hành vi có khả năng là mã độc và đưa ra cảnh báo kịp thời. Các kỹ thuật này không chỉ giúp tăng độ chính xác trong việc phát hiện mã độc mà còn cải thiện hiệu suất hoạt động của các hệ thống bảo mật.
* Đề tài "Phát hiện mã độc" nhằm mục tiêu áp dụng các thuật toán học máy để xây dựng mô hình phân loại mã độc và mã lành, từ đó giúp tăng cường khả năng phát hiện các mối đe dọa tiềm ẩn. Báo cáo này sẽ trình bày chi tiết về quá trình xử lý dữ liệu, lựa chọn mô hình, đánh giá hiệu suất, và so sánh giữa các phương pháp để tìm ra mô hình tối ưu cho bài toán phát hiện mã độc.

# Mô tả dữ liệu

* Dữ liệu TUNADROMD là một tập hợp phong phú các đặc trưng phần mềm ứng dụng được thu thập với mục tiêu chính là phục vụ cho việc phân loại mã độc và phần mềm an toàn. Tập dữ liệu này được xây dựng từ các ứng dụng thực tế và có thể chứa những đặc điểm quan trọng mà các thuật toán phân loại cần phân tích để đưa ra quyết định chính xác về việc xác định mã độc hay không.
* Tổng cộng, tập dữ liệu này chứa 4,465 mẫu, mỗi mẫu được đại diện bởi 241 thuộc tính. Trong số này, 240 thuộc tính là độc lập và mô tả các quyền truy cập cũng như các tính năng của ứng dụng. Các thuộc tính này có thể bao gồm thông tin về quyền truy cập vào mạng, lưu trữ, hệ thống, và các tính năng khác mà ứng dụng yêu cầu để hoạt động. Những thuộc tính này rất quan trọng trong việc phân tích hành vi của ứng dụng, từ đó giúp xác định được nguy cơ mà nó có thể gây ra.
* Thuộc tính cuối cùng là thuộc tính nhãn, đóng vai trò như mục tiêu cho quá trình phân loại. Nhãn này có giá trị nhị phân, trong đó giá trị 1 được chỉ định cho các mẫu mã độc (malware) và giá trị 0 được chỉ định cho các mẫu không phải mã độc. Việc phân loại chính xác giữa hai loại này không chỉ giúp bảo vệ người dùng khỏi các mối đe dọa tiềm tàng mà còn hỗ trợ các nhà phát triển phần mềm trong việc cải thiện tính bảo mật của ứng dụng của họ.
* Nhờ vào cấu trúc dữ liệu chi tiết và phong phú, tập dữ liệu TUNADROMD là một nguồn tài nguyên quý giá cho các nhà nghiên cứu và phát triển trong lĩnh vực an toàn thông tin, góp phần nâng cao khả năng phát hiện và phân loại mã độc bằng các thuật toán học máy hiện đại. Việc sử dụng dữ liệu này trong các mô hình phân loại giúp cải thiện độ chính xác và hiệu suất của các hệ thống an ninh mạng, từ đó tạo ra một môi trường sử dụng an toàn hơn cho người dùng.

## 2.1. Các thuộc tính

* **Các thuộc tính có thể ảnh hưởng đến việc phát hiện malware:**

1. ACCESS\_ALL\_DOWNLOADS: Quyền truy cập tất cả các tệp đã tải xuống.
2. ACCESS\_CACHE\_FILESYSTEM: Quyền truy cập hệ thống tệp bộ nhớ đệm.
3. ACCESS\_NETWORK\_STATE: Quyền truy cập trạng thái mạng.
4. ACCESS\_COARSE\_LOCATION: Quyền truy cập vị trí tương đối của thiết bị.
5. ACCESS\_FINE\_LOCATION: Quyền truy cập vị trí chính xác của thiết bị.
6. ACCESS\_WIFI\_STATE: Quyền truy cập trạng thái kết nối Wi-Fi.
7. RECEIVE\_SMS: Quyền nhận tin nhắn SMS.
8. SEND\_SMS: Quyền gửi tin nhắn SMS.
9. READ\_SMS: Quyền đọc tin nhắn SMS.
10. READ\_LOGS: Quyền đọc nhật ký hệ thống.
11. READ\_CONTACTS: Quyền đọc danh bạ trên thiết bị.
12. WRITE\_EXTERNAL\_STORAGE: Quyền ghi dữ liệu vào bộ nhớ ngoài.
13. INSTALL\_SHORTCUT: Quyền cài đặt lối tắt trên màn hình chính.
14. READ\_PHONE\_STATE: Quyền truy cập trạng thái điện thoại.
15. WRITE\_SETTINGS: Quyền thay đổi cài đặt hệ thống.
16. VIBRATE: Quyền điều khiển chế độ rung của thiết bị.
17. GET\_ACCOUNTS: Quyền truy cập danh sách các tài khoản đã đăng nhập trên thiết bị.
18. MODIFY\_AUDIO\_SETTINGS: Quyền thay đổi cài đặt âm thanh.
19. SYSTEM\_ALERT\_WINDOW: Quyền hiển thị các cửa sổ cảnh báo trên các ứng dụng khác.
20. RECEIVE\_BOOT\_COMPLETED: Quyền nhận thông báo khi thiết bị khởi động lại.
21. BIND\_ACCESSIBILITY\_SERVICE: Quyền truy cập dịch vụ hỗ trợ đặc biệt.
22. WAKE\_LOCK: Quyền giữ thiết bị ở chế độ thức.
23. CAMERA: Quyền truy cập máy ảnh của thiết bị.
24. INTERNET: Quyền truy cập Internet của ứng dụng.
25. Các thuộc tính đáng chú ý khác trong phát hiện malware
26. ALLOW\_MOCK\_LOCATION: Cho phép ứng dụng giả mạo vị trí, có thể được sử dụng để lừa đảo người dùng.
27. USE\_FINGERPRINT: Quyền truy cập vào cảm biến vân tay, có thể bị lợi dụng để lấy cắp thông tin.
28. ACCESS\_NOTIFICATION\_POLICY: Quyền thay đổi chính sách thông báo, có thể dùng để kiểm soát thông báo của thiết bị.
29. REQUEST\_IGNORE\_BATTERY\_OPTIMIZATIONS: Quyền yêu cầu bỏ qua tối ưu hóa pin, cho phép ứng dụng chạy liên tục và có thể sử dụng tài nguyên mà không bị gián đoạn.
30. FOREGROUND\_SERVICE: Quyền cho phép ứng dụng chạy trong nền mà không bị tắt.
31. ACCESS\_MEDIA\_LOCATION: Quyền truy cập vào vị trí của các tệp phương tiện.
32. GET\_TASKS: Quyền truy cập vào danh sách các tác vụ đang chạy, có thể bị lợi dụng để theo dõi hoạt động của người dùng.

* **Nhãn mục tiêu:**

1. Label: Thuộc tính mục tiêu, có giá trị 1 nếu phần mềm là malware và 0 nếu là goodware.

* **Các thuộc tính đáng chú ý nhất**
  1. Trong số tất cả các thuộc tính này, những thuộc tính sau thường được coi là quan trọng nhất trong việc phát hiện malware:
  2. ACCESS\_FINE\_LOCATION
  3. SEND\_SMS
  4. READ\_CONTACTS
  5. WRITE\_SETTINGS
  6. SYSTEM\_ALERT\_WINDOW
  7. RECEIVE\_BOOT\_COMPLETED
  8. ALLOW\_MOCK\_LOCATION

=> Các thuộc tính này thường cho phép phần mềm độc hại thực hiện các hành động có hại, đánh cắp dữ liệu cá nhân hoặc gây rối cho người dùng. Tập dữ liệu của bạn có thể chứa nhiều thuộc tính hơn, vì vậy cần phân tích chi tiết để xác định các yếu tố quan trọng trong việc phát hiện malware.

## 2.2. Phân tích sơ bộ dữ liệu

* Trước khi tiến hành xây dựng mô hình phân loại, tôi đã thực hiện một số phân tích sơ bộ để hiểu rõ hơn về tập dữ liệu TUNADROMD. Những bước phân tích này bao gồm:

1. Kiểm tra tên các cột dữ liệu và số lượng thuộc tính: Tôi đã xác định rõ ràng các tên cột trong tập dữ liệu, cũng như số lượng thuộc tính mà tập dữ liệu chứa, giúp tôi có cái nhìn tổng quát về cấu trúc của dữ liệu. Việc này cũng cho phép tôi nhận diện các thuộc tính nào có thể đóng vai trò quan trọng trong quá trình phân loại.
2. Kiểm tra kiểu dữ liệu: Tôi đã xem xét các kiểu dữ liệu của từng thuộc tính để đảm bảo rằng chúng phù hợp cho quá trình phân tích và mô hình hóa. Điều này bao gồm việc xác định các thuộc tính nào là số, phân loại, hoặc nhị phân, từ đó quyết định các phương pháp xử lý thích hợp cho mỗi loại dữ liệu.
3. Tỷ lệ phân bổ của các nhãn: Để hiểu rõ hơn về tính cân bằng của dữ liệu, tôi đã phân tích tỷ lệ phân bổ giữa hai nhãn (1 cho mã độc và 0 cho không phải mã độc). Việc này giúp tôi xác định xem có sự thiên lệch nào trong dữ liệu hay không, từ đó đưa ra các giải pháp phù hợp để xử lý.
4. Kiểm tra dữ liệu trong file TUANDROMD.csv: Tôi đã xem xét dữ liệu trong file chính cùng với hai file chú thích đi kèm để nắm rõ các nhãn hợp lệ của các thuộc tính. Việc này không chỉ giúp nhận diện các nhãn hợp lệ mà còn hỗ trợ trong việc loại trừ các nhãn không hợp lệ có thể xuất hiện trong bộ dữ liệu, từ đó cải thiện độ chính xác của mô hình.
5. Phân phối của các thuộc tính: Tôi đã kiểm tra sự phân phối của từng thuộc tính để xác định có thuộc tính nào có giá trị không được quy định hay không. Điều này quan trọng nhằm đảm bảo rằng các thuộc tính được chọn đều có giá trị thông tin cao và có thể hỗ trợ tốt cho quá trình phân loại.
6. Phân tích thống kê mô tả: Đối với các thuộc tính định lượng, tôi đã thực hiện phân tích thống kê mô tả để hiểu rõ hơn về các đặc điểm như trung bình, độ lệch chuẩn và các giá trị ngoại lệ. Những thông tin này không chỉ giúp tôi nắm bắt được sự phân bố của dữ liệu mà còn hỗ trợ trong việc lựa chọn các phương pháp xử lý dữ liệu phù hợp. Chẳng hạn, việc phát hiện các giá trị ngoại lệ có thể yêu cầu tôi phải xử lý chúng trước khi đưa vào mô hình, nhằm cải thiện độ chính xác và độ tin cậy của kết quả phân loại.
7. Thông qua các bước phân tích sơ bộ này, tôi đã có cái nhìn tổng quan về dữ liệu, từ đó chuẩn bị tốt hơn cho quá trình xây dựng và đánh giá mô hình phân loại mã độc. Việc hiểu rõ đặc điểm của tập dữ liệu không chỉ giúp nâng cao hiệu suất mô hình mà còn tăng cường khả năng đưa ra các quyết định chính xác trong phân loại mã độc và phần mềm an toàn.

# Phân tích dữ liệu

* Phân tích dữ liệu là một bước cực kỳ quan trọng trong bất kỳ dự án học máy nào, đặc biệt là khi làm việc với các bộ dữ liệu phức tạp như TUANDROMD.csv. Qua quá trình phân tích này, tôi có thể hiểu rõ hơn về các thuộc tính trong tập dữ liệu cũng như mối quan hệ giữa chúng. Việc nắm vững cấu trúc và nội dung của dữ liệu không chỉ giúp tôi lựa chọn mô hình phù hợp mà còn cải thiện khả năng dự đoán, từ đó nâng cao độ chính xác của kết quả.
* Đầu tiên, tôi tiến hành đọc tập tin CSV chứa dữ liệu để thu thập thông tin tổng quan. Tập tin TUANDROMD.csv bao gồm nhiều cột, mỗi cột đại diện cho một thuộc tính của ứng dụng phần mềm. Một số thuộc tính quan trọng trong tập dữ liệu này bao gồm các quyền truy cập như ACCESS\_ALL\_DOWNLOADS, ACCESS\_NETWORK\_STATE, và INSTALL\_SHORTCUT, cùng với thuộc tính nhãn Label, cho biết ứng dụng đó có phải là malware (1) hay không (0).
* Khi đã nắm bắt được cấu trúc của tập tin CSV, tôi tiếp tục kiểm tra xem có bất kỳ giá trị nào bị thiếu hay không. Việc kiểm tra này rất quan trọng vì các giá trị thiếu có thể gây ra nhiều vấn đề trong quá trình phân tích và xây dựng mô hình. Nếu không được xử lý kịp thời, những giá trị này có thể dẫn đến những kết luận sai lầm và ảnh hưởng đến khả năng dự đoán của mô hình.
* Sau khi xác định được các giá trị thiếu, tôi tiến hành phân tích rủi ro liên quan đến chúng. Mỗi thuộc tính trong tập dữ liệu đều có thể ảnh hưởng đáng kể đến khả năng phân loại phần mềm độc hại. Chẳng hạn, các thuộc tính như ACCESS\_ALL\_DOWNLOADS, ACCESS\_NETWORK\_STATE, và INSTALL\_SHORTCUT có thể là những yếu tố quan trọng để xác định tính độc hại của một ứng dụng. Nếu những thuộc tính này không có đủ dữ liệu, độ tin cậy của mô hình sẽ bị ảnh hưởng. Ví dụ, nếu các giá trị về quyền truy cập mạng không đầy đủ, mô hình có thể không nhận diện được những ứng dụng nguy hiểm, dẫn đến việc phát hiện phần mềm độc hại không chính xác hoặc bỏ sót các phần mềm độc hại quan trọng. Do đó, việc xử lý các giá trị thiếu không chỉ là một kỹ thuật, mà còn là cách tôi hiểu và quản lý rủi ro trong quá trình phân tích dữ liệu.
* Tiếp theo, tôi đã xem xét sự phân bố của từng thuộc tính thông qua các biểu đồ trực quan. Việc phân tích này giúp tôi nhận diện các mẫu và xu hướng trong dữ liệu. Chẳng hạn, nếu thuộc tính ACCESS\_FINE\_LOCATION có sự phân bố không đồng đều, điều này có thể ảnh hưởng đến khả năng phân loại của mô hình. Một thuộc tính như READ\_LOGS có thể có phân bố tập trung nhiều ở các ứng dụng độc hại, trong khi các ứng dụng an toàn ít sử dụng quyền này hơn. Phân tích phân bố này giúp tôi xác định các thuộc tính nào có thể là dấu hiệu nhận biết rõ ràng giữa malware và goodware.
* Bên cạnh đó, tôi cũng phân tích các mối tương quan giữa các thuộc tính. Việc này không chỉ giúp tôi phát hiện các thuộc tính có liên quan chặt chẽ đến nhãn phân loại mà còn cung cấp cái nhìn rõ hơn về những thuộc tính nào có thể là yếu tố quyết định trong việc phân loại malware. Ví dụ, các thuộc tính như ACCESS\_WIFI\_STATE và ACCESS\_NETWORK\_STATE có thể có mối tương quan mạnh với nhau, cho thấy rằng những ứng dụng yêu cầu quyền truy cập mạng có khả năng cao hơn bị đánh giá là phần mềm độc hại. Phân tích các mối tương quan này sẽ hỗ trợ tôi trong việc chọn lọc và tối ưu hóa các thuộc tính cần thiết cho mô hình phân loại.
* Cuối cùng, sau khi hoàn tất các bước phân tích và đánh giá, tôi đã sẵn sàng để tiến đến giai đoạn tiếp theo của dự án: mô hình hóa. Giai đoạn này sẽ tập trung vào việc xây dựng và đánh giá các mô hình học máy, nhằm tối ưu hóa quy trình phát hiện phần mềm độc hại. Việc chuyển từ phân tích dữ liệu sang mô hình hóa sẽ cho phép tôi áp dụng những kiến thức đã thu thập được để cải thiện hiệu quả của các mô hình trong việc phát hiện và ngăn chặn phần mềm độc hại một cách chính xác và hiệu quả hơn.

# Mô hình hóa

## 4.1 Tiền xử lý dữ liệu

* Trước khi tiến hành xây dựng mô hình, việc tiền xử lý dữ liệu là rất quan trọng để đảm bảo rằng dữ liệu đầu vào có chất lượng cao và phù hợp cho các thuật toán học máy. Tôi đã thực hiện một số bước chính trong quy trình tiền xử lý dữ liệu như sau:
* Kiểm tra và xử lý các giá trị thiếu: Đầu tiên, tôi đã kiểm tra dữ liệu để xác định các cột có giá trị thiếu. Việc này được thực hiện thông qua hàm isnull() kết hợp với sum() để đếm số lượng giá trị thiếu trong mỗi cột. Sau khi xác định được các cột có giá trị thiếu, tôi đã quyết định sử dụng phương pháp thay thế các giá trị thiếu bằng giá trị trung bình của cột tương ứng. Phương pháp này đơn giản và hiệu quả cho những cột mà dữ liệu phân bố đồng đều.

A screen shot of a computer code

Description automatically generated

Hình 1: Xử lý giá trị còn thiếu

A bar code with black text

Description automatically generated

Hình 1.1: Biểu đồ các giá trị còn thiếu

* Chuẩn hóa dữ liệu: Để nâng cao hiệu suất của các mô hình học máy, tôi đã thực hiện chuẩn hóa dữ liệu. Quy trình này bao gồm việc chuẩn hóa các thuộc tính về khoảng giá trị (min-max scaling), nhằm đảm bảo rằng tất cả các thuộc tính có thể so sánh được và không làm lệch trọng số trong quá trình học.

A screenshot of a computer program

Description automatically generated

Hình 2: Chuẩn hóa dữ liệu

A screenshot of a graph

Description automatically generated

Hình 2.1: Hình ảnh trước khi chuẩn hóa

A diagram of a graph

Description automatically generated with medium confidence

Hình 2.2: Hình ảnh sau khi chuẩn hóa

A screenshot of a computer screen

Description automatically generated

Hình 2.3: Hình ảnh trước khi chuẩn hóa theo dạng bitmap

A screenshot of a computer screen

Description automatically generated

Hình 2.4: Hình ảnh sau khi chuẩn hóa dạng bitmap

* Tạo các đặc trưng mới (Feature Engineering): chia dữ liệu thành các đặc điểm và nhãn để chuẩn bị cho việc xây dựng mô hình. Trong đoạn mã dưới đây, tôi đã sử dụng phương pháp loại bỏ cột 'Label' (là nhãn phân loại) để lấy các đặc điểm của dữ liệu, trong khi đó, nhãn (các giá trị 1 và 0 biểu thị cho malware và không phải malware) được lưu riêng.

## 4.2 Chia tập Dữ liệu

* Tôi đã tiến hành chia dữ liệu thành hai phần: tập huấn luyện và tập kiểm tra với tỷ lệ 80/20. Điều này giúp đảm bảo rằng mô hình sẽ được huấn luyện trên một tập dữ liệu đủ lớn, đồng thời cho phép kiểm tra độc lập trên các dữ liệu chưa thấy nhằm đánh giá khả năng tổng quát hóa của mô hình.

A screen shot of a computer code

Description automatically generated

Hình 3: Hình ảnh chia tập dữ liệu

* Việc tách biệt các tập dữ liệu như vậy sẽ giúp đảm bảo mô hình có thể học tốt từ các đặc điểm có ý nghĩa, đồng thời tối ưu hóa hiệu suất và độ chính xác trong các dự đoán khi áp dụng trên các mẫu mới.
* Ngoài ra, tôi cũng đã đảm bảo rằng việc chia dữ liệu được thực hiện ngẫu nhiên và đồng đều, nhằm giữ nguyên tính phân phối của các thuộc tính trong cả hai tập huấn luyện và kiểm tra. Điều này giúp mô hình có khả năng học từ các mẫu đại diện cho toàn bộ tập dữ liệu, từ đó tăng cường độ chính xác và độ tin cậy trong việc dự đoán phần mềm độc hại. Việc phân chia dữ liệu một cách cẩn thận là bước quan trọng trong quy trình xây dựng mô hình, đảm bảo tối ưu hóa hiệu suất và nâng cao chất lượng các dự đoán sau này.

## 4.3 Chọn và đánh giá mô hình

## 4.3.1. Lựa chọn mô hình

* Trong dự án phát hiện mã độc này, tôi đã lựa chọn mô hình Random Forest (Rừng ngẫu nhiên) để phân loại dữ liệu. Random Forest là một mô hình học máy mạnh mẽ, có khả năng giải quyết tốt các bài toán phân loại và hồi quy. Điểm mạnh của Random Forest nằm ở khả năng kết hợp nhiều cây quyết định độc lập để tạo ra một dự đoán chính xác hơn, từ đó giảm thiểu hiện tượng quá khớp (overfitting) so với các mô hình cây quyết định đơn lẻ.
* Lý do tôi lựa chọn mô hình Random Forest trong bài toán phát hiện mã độc bao gồm:

1. Khả năng tổng quát hóa tốt: Mô hình Random Forest không dễ bị overfitting nhờ việc tạo ra nhiều cây quyết định khác nhau từ các mẫu ngẫu nhiên của dữ liệu. Điều này giúp tăng độ tin cậy và khả năng tổng quát hóa của mô hình.
2. Khả năng xử lý dữ liệu có nhiều đặc trưng: Dữ liệu phát hiện mã độc thường có nhiều đặc trưng liên quan đến các quyền truy cập và thông tin hệ thống, do đó Random Forest có khả năng xử lý hiệu quả những dữ liệu phức tạp này.
3. Hiệu suất dự đoán cao: Random Forest đã được chứng minh là một mô hình có độ chính xác cao trong nhiều bài toán phân loại, đặc biệt là các bài toán phân loại nhị phân như phân loại mã độc và không phải mã độc.

## 4.3.2. Triển khai mô hình

* Để giải thích rõ hơn về mô hình Random Forest mà tôi đã xây dựng, chúng ta cần nêu ra các khái niệm, công thức và đại lượng quan trọng liên quan. Phần này bao gồm các khái niệm nền tảng về cây quyết định (Decision Tree) và rừng ngẫu nhiên (Random Forest).
* **Cây quyết định (Decision Tree):** Cây quyết định là một cấu trúc dữ liệu dạng cây, trong đó mỗi nút trong cây đại diện cho một thuộc tính (feature), và các cạnh phân nhánh của cây dựa trên các điều kiện phân tách (threshold) của thuộc tính đó.
  1. Điều kiện dừng: Quá trình xây dựng cây sẽ dừng khi một trong các điều kiện sau được thỏa mãn:

1. Tất cả các mẫu đều thuộc cùng một lớp (lý tưởng).
2. Không còn thuộc tính nào để chia tiếp.
3. Độ sâu tối đa của cây đã đạt đến giới hạn được chỉ định (max\_depth).
4. **Gini index**: Là một thước đo mức độ hỗn hợp (impurity) của một tập hợp các mẫu. Gini index được tính dựa trên xác suất của các nhãn trong tập dữ liệu. Công thức Gini index cho một tập dữ liệu có hai nhãn (giả sử là 0 và 1) được tính như sau:

**Gini(y) = 1 – P02 – P 12**

* Trong đó:
* P02: là tỉ lệ mẫu thuộc lớp 0.
* P 12: là tỉ lệ mẫu thuộc lớp 1.

1. Gini index càng nhỏ, tập dữ liệu càng "thuần".

* Quá trình phân chia (split): Tại mỗi nút, thuật toán sẽ chọn ra thuộc tính và giá trị ngưỡng sao cho impurity (Gini index) của dữ liệu sau khi phân chia là nhỏ nhất. Điều này giúp phân biệt các lớp nhãn một cách rõ ràng hơn.
* **Rừng ngẫu nhiên (Random Forest):** Rừng ngẫu nhiên là một mô hình học máy gồm nhiều cây quyết định hoạt động cùng nhau. Mỗi cây trong rừng được xây dựng dựa trên các mẫu ngẫu nhiên từ dữ liệu ban đầu (bootstrap sampling) và chỉ sử dụng một tập con ngẫu nhiên của các thuộc tính tại mỗi bước phân chia. Điều này giúp giảm thiểu overfitting và cải thiện độ chính xác của mô hình.
  1. Các thành phần chính:
     1. Bootstrap sampling: Đây là phương pháp chọn ngẫu nhiên mẫu từ tập dữ liệu ban đầu, cho phép lặp lại (sampling with replacement). Điều này đảm bảo rằng mỗi cây trong rừng được huấn luyện trên các mẫu khác nhau và do đó sẽ có sự khác biệt trong các cây.
     2. Số lượng cây (n\_trees): Số lượng cây trong rừng quyết định mức độ chính xác của mô hình. Số lượng cây càng lớn thì kết quả dự đoán càng ổn định.

## 4.3.3: Cách vận hành

1. **Cây quyết định:**

* **Khởi Tạo Cây Quyết Định:** Khi khởi tạo một cây quyết định, tham số max\_depth được chỉ định để xác định độ sâu tối đa mà cây có thể đạt được. Biến self.tree được khởi tạo với giá trị None, dùng để lưu trữ cấu trúc cây sau khi nó được xây dựng.
* **Huấn Luyện Mô Hình:** Trong phương thức fit, mô hình sẽ được huấn luyện trên dữ liệu đầu vào X (ma trận đặc trưng) và y (mảng nhãn). Phương thức \_build\_tree được gọi để xây dựng cây quyết định từ dữ liệu này.
* **Xây Dựng Cây Quyết Định:** Phương thức \_build\_tree là phần quan trọng trong việc xây dựng cây quyết định. Nó thực hiện các bước sau:
  1. Kiểm Tra Điều Kiện Dừng: Nếu số lượng mẫu là 0, độ sâu hiện tại bằng với max\_depth, hoặc tất cả các nhãn giống nhau (chỉ có một lớp), thì cây dừng lại và trả về giá trị trung bình của nhãn (đối với bài toán hồi quy).
  2. Tìm Split Tốt Nhất: Phương thức \_best\_split được gọi để tìm ra thuộc tính và ngưỡng (threshold) tốt nhất cho việc chia dữ liệu. Nó sẽ kiểm tra tất cả các thuộc tính và tìm ra ngưỡng tối ưu mà tại đó sẽ chia dữ liệu thành hai nhánh.
  3. Chia Dữ Liệu: Dữ liệu được chia thành hai nhánh dựa trên thuộc tính và ngưỡng tốt nhất tìm được. Các chỉ số (indices) của mẫu cho nhánh bên trái (khi giá trị thuộc tính nhỏ hơn ngưỡng) và nhánh bên phải (khi giá trị lớn hơn hoặc bằng ngưỡng) sẽ được xác định.
  4. Gọi Đệ Quy: Phương thức \_build\_tree được gọi đệ quy cho cả hai nhánh (trái và phải) với dữ liệu đã chia, đồng thời tăng độ sâu lên một đơn vị. Cuối cùng, thông tin về thuộc tính tốt nhất, ngưỡng và các cây con được trả về để tạo thành cấu trúc cây hoàn chỉnh.
* **Tìm Split Tốt Nhất:** Phương thức \_best\_split được sử dụng để tìm thuộc tính và ngưỡng tốt nhất để chia dữ liệu. Nó thực hiện các bước sau:

1. Duyệt Qua Các Thuộc Tính:
   * Đối với mỗi thuộc tính trong dữ liệu, nó xác định các ngưỡng khác nhau dựa trên các giá trị duy nhất của thuộc tính đó.
2. Chia Dữ Liệu:
   * Với mỗi ngưỡng, dữ liệu được chia thành hai nhánh, và nếu một nhánh rỗng, việc chia này sẽ bị bỏ qua.
3. Tính Toán Impurity:
   * Phương thức \_gini\_index được gọi để tính toán mức độ impurity (không thuần khiết) cho việc chia này. Nếu mức độ impurity thấp hơn mức tốt nhất đã tìm thấy trước đó, nó sẽ cập nhật thông tin về thuộc tính và ngưỡng tốt nhất.

* **Tính Toán Gini Index:** Phương thức \_gini\_index được sử dụng để tính toán chỉ số Gini cho hai nhánh sau khi chia. Nó thực hiện các bước sau:

1. Tính Toán Tổng Số Mẫu:
   * Tổng số mẫu trong cả hai nhánh được xác định.
2. Tính Gini Có Trọng Số:
   * Tính toán chỉ số Gini cho mỗi nhánh và trả về giá trị gini index có trọng số cho cả hai nhánh.

* **Dự Đoán Nhãn:** Phương thức predict nhận ma trận đặc trưng cho các mẫu mới và trả về mảng nhãn dự đoán. Nó gọi phương thức \_predict\_sample cho từng mẫu trong X.
* **Dự Đoán Một Mẫu Đơn:** Phương thức \_predict\_sample thực hiện dự đoán cho một mẫu đơn. Nếu cây hiện tại là một tuple (có nghĩa là nó vẫn đang trong quá trình phân chia), nó sẽ kiểm tra thuộc tính và ngưỡng, từ đó quyết định đi tiếp vào nhánh nào (trái hoặc phải). Nếu cây đã đạt đến một nút lá (leaf node), nó sẽ trả về nhãn dự đoán dựa trên giá trị trung bình.

1. **Random forest**

* **Khởi Tạo Mô Hình Rừng Ngẫu Nhiên:** Khi khởi tạo một mô hình rừng ngẫu nhiên, hai tham số chính được chỉ định là n\_trees (số lượng cây trong rừng) và max\_depth (độ sâu tối đa cho mỗi cây). Biến self.trees được khởi tạo như một danh sách rỗng để lưu trữ các cây quyết định sẽ được tạo ra trong quá trình huấn luyện.
* **Huấn Luyện Mô Hình:** Trong phương thức fit, mô hình sẽ được huấn luyện trên dữ liệu đầu vào X (ma trận đặc trưng) và y (mảng nhãn). Dưới đây là các bước cụ thể trong quá trình huấn luyện:

1. Xác Định Số Lượng Mẫu:
   * Số lượng mẫu trong tập dữ liệu được xác định thông qua X.shape[0].
2. Tạo Các Cây Quyết Định:
   * Một vòng lặp được chạy n\_trees lần. Trong mỗi lần lặp:
   * Chọn Mẫu Ngẫu Nhiên: Sử dụng phương pháp bootstrap sampling, các chỉ số mẫu được chọn ngẫu nhiên từ tập dữ liệu gốc với việc cho phép lặp lại. Điều này có nghĩa là cùng một mẫu có thể được chọn nhiều lần.
   * Tạo Mẫu Ngẫu Nhiên: Tạo ra ma trận đặc trưng X\_sample và mảng nhãn y\_sample dựa trên các chỉ số đã chọn.
   * Tạo Cây Quyết Định: Một cây quyết định mới được tạo ra với độ sâu tối đa được chỉ định và sau đó được huấn luyện trên mẫu ngẫu nhiên.
   * Lưu Cây: Cây vừa huấn luyện được thêm vào danh sách self.trees.

* **Dự Đoán Nhãn:** Phương thức predict được sử dụng để dự đoán nhãn cho các mẫu mới. Các bước trong phương thức này như sau:

1. Dự Đoán từ Tất Cả Các Cây:
   * Đối với mỗi cây trong danh sách self.trees, phương thức predict của cây được gọi với dữ liệu đầu vào X. Kết quả là một ma trận tree\_preds, trong đó mỗi hàng chứa các dự đoán từ một cây.
2. Tính Toán Nhãn Cuối Cùng:
   * Để xác định nhãn cuối cùng cho mỗi mẫu, phương thức sử dụng np.bincount để đếm số lần mỗi nhãn xuất hiện trong các dự đoán của cây. Nhãn nào xuất hiện nhiều nhất sẽ được chọn là nhãn dự đoán cho mẫu đó.

A black rectangular object with text

Description automatically generated

Hình 4: Hình ảnh huấn luyện mô hình

# Dự đoán dữ liệu kiểm tra và gửi kết quả

## 5.1. dữ liệu kiểm tra và gửi kết quả

* Trong giai đoạn này, chúng tôi sẽ thực hiện việc dự đoán nhãn cho tập dữ liệu kiểm tra bằng cách sử dụng mô hình Random Forest đã được huấn luyện ở phần trước. Cụ thể, chúng tôi sẽ sử dụng phương thức predict của lớp Random Forest để dự đoán nhãn cho các mẫu trong tập kiểm tra và tính toán độ chính xác của mô hình.
* Trước tiên, chúng tôi sẽ gọi phương thức predict để lấy ra các nhãn dự đoán cho tập kiểm tra. Nhãn dự đoán này sẽ được lưu trữ trong biến y\_pred. Sau đó, chúng tôi sẽ so sánh nhãn dự đoán với nhãn thực tế y\_test để xác định độ chính xác của mô hình. Độ chính xác được tính toán bằng cách tính tỉ lệ giữa số lượng nhãn dự đoán đúng và tổng số mẫu trong tập kiểm tra.
* Cuối cùng, chúng tôi sẽ in ra độ chính xác của mô hình dưới dạng phần trăm để đánh giá hiệu suất của mô hình trong việc phân loại các mẫu vào các nhãn tương ứng. Điều này giúp chúng tôi hiểu rõ hơn về khả năng của mô hình trong việc dự đoán các nhãn của dữ liệu chưa thấy.

A screenshot of a computer program

Description automatically generated

Hình 5: Hình ảnh tập kiểm tra và độ chính xác

* Với mô hình tôi xây dựng ở trên thì độ chính xác là **98.43%**

## 5.2. Biểu đồ mô tả kết quả

* **Số lượng thực tế và số lượng đoán**

A graph with red and blue squares

Description automatically generated

Hình 6: Hình ảnh số lượng thực tế và dự đoán

* **Ma trận nhầm lẫn**

**A blue and white squares with black numbers

Description automatically generated**

Hình 7: Hình ảnh ma trận nhầm lẫn

# Ưu điểm

* Khả năng xử lý dữ liệu lớn: Random Forest có khả năng xử lý và phân tích hiệu quả với các tập dữ liệu lớn, nhiều đặc trưng mà không cần giảm số lượng đặc trưng.
* Giảm thiểu overfitting: Bằng cách kết hợp nhiều cây quyết định (decision trees) và sử dụng phương pháp voting, mô hình giúp giảm thiểu khả năng overfitting mà thường xảy ra với các cây quyết định đơn lẻ.
* Độ chính xác cao: Random Forest thường cho kết quả dự đoán chính xác hơn so với các mô hình đơn lẻ khác, nhờ vào việc kết hợp nhiều mô hình nhỏ để đưa ra một kết quả tổng quát hơn.
* Khả năng đo lường tầm quan trọng của đặc trưng: Mô hình cung cấp thông tin về độ quan trọng của từng đặc trưng, giúp trong việc lựa chọn và tối ưu hóa đặc trưng.
* Khả năng hoạt động với dữ liệu không hoàn hảo: Random Forest có khả năng hoạt động tốt ngay cả khi dữ liệu chứa nhiều giá trị thiếu (missing values).

# Nhược điểm

* Thời gian huấn luyện lâu: Do phải xây dựng nhiều cây quyết định, thời gian huấn luyện mô hình Random Forest có thể lâu hơn so với các mô hình đơn giản khác.
* Khó giải thích: Mặc dù mỗi cây quyết định trong Random Forest có thể được giải thích, việc giải thích mô hình tổng thể trở nên phức tạp do sự kết hợp của nhiều cây.
* Tốn bộ nhớ: Random Forest có thể tiêu tốn nhiều bộ nhớ do phải lưu trữ nhiều cây quyết định, đặc biệt khi số lượng cây (n\_trees) lớn.
* Kém hiệu quả với dữ liệu rất lớn: Trong một số trường hợp, nếu kích thước tập dữ liệu quá lớn, Random Forest có thể trở nên kém hiệu quả về mặt tính toán.
* Khả năng không tốt với dữ liệu phân loại không cân bằng: Random Forest có thể gặp khó khăn khi làm việc với các tập dữ liệu mà các lớp không cân bằng, dẫn đến kết quả dự đoán thiên lệch.

# Kết luận

* Trong báo cáo này, chúng tôi đã trình bày quy trình xây dựng và đánh giá mô hình Random Forest nhằm phát hiện mã độc từ tập dữ liệu đã cho. Mô hình Random Forest được chọn lựa vì khả năng xử lý hiệu quả các đặc trưng phức tạp và tính chính xác cao trong việc dự đoán.
* Quá trình xây dựng mô hình bao gồm các bước như chuẩn bị dữ liệu, huấn luyện mô hình, và đánh giá hiệu suất. Thông qua việc áp dụng phương pháp bootstrap sampling và kết hợp nhiều cây quyết định, mô hình đã cho thấy khả năng phân loại tốt với độ chính xác cao. Việc đánh giá hiệu suất của mô hình được thực hiện thông qua các chỉ số như độ chính xác, độ nhạy, độ đặc hiệu và ma trận nhầm lẫn, cho thấy khả năng phát hiện mã độc đạt yêu cầu.
* Bên cạnh những ưu điểm nổi bật như giảm thiểu overfitting và khả năng đo lường tầm quan trọng của đặc trưng, mô hình Random Forest cũng tồn tại một số nhược điểm như thời gian huấn luyện lâu và độ khó trong việc giải thích kết quả tổng thể. Tuy nhiên, những nhược điểm này không ảnh hưởng đáng kể đến khả năng ứng dụng của mô hình trong thực tế.
* Cuối cùng, chúng tôi đề xuất các hướng phát triển tiếp theo cho mô hình này, bao gồm việc tối ưu hóa các tham số, thử nghiệm với các phương pháp học máy khác, và áp dụng các kỹ thuật tăng cường dữ liệu để cải thiện hiệu suất dự đoán. Hy vọng rằng những kết quả đạt được từ mô hình Random Forest sẽ đóng góp tích cực cho công tác phát hiện mã độc và bảo mật thông tin.