**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

---------

**BÁO CÁO ĐỒ ÁN**

**TRIỂN KHAI PHÂN LOẠI MÃ ĐỘC ANDROID BẰNG NLP**

**Sinh viên thực hiện:**

1. Nguyễn Thị Nhàn MSSV: 21522405

2. Nguyễn Thị Kim Ngân MSSV: 21521174

Thành phố Hồ Chí Minh, tháng 11 năm 2023

**MỤC LỤC**

Nội dung

[ABSTRACT 1](#_Toc152137814)

[I. INTRODUCTION 1](#_Toc152137815)

[II. CÁC BÀI BÁO LIÊN QUAN 3](#_Toc152137816)

[1. Phân loại mã độc sử dụng phương pháp Deep Learning 3](#_Toc152137817)

[2. Phân loại mã độc sử dụng NLP 3](#_Toc152137818)

[3. Phần mềm độc hại Android dựa trên chuỗi kết hợp phát hiện bằng xử lý ngôn ngữ tự nhiên 4](#_Toc152137819)

[4. Phân loại mã độc trên Android với Deep Learning sử dụng RNN từ chuỗi Opcode 4](#_Toc152137820)

[5. Phân loại mã độc sử dụng lời gọi hệ thống 4](#_Toc152137821)

[III. PHÂN TÍCH TẬP DỮ LIỆU 5](#_Toc152137822)

[1. Mô tả tập dữ liệu 5](#_Toc152137823)

[2. Tiền xử lý tập dữ liệu 6](#_Toc152137824)

[3. Chia tỷ lệ tính năng 7](#_Toc152137825)

[4. Tách tập dữ liệu 7](#_Toc152137826)

[IV. CÁC THUẬT TOÁN PHÂN LOẠI MÃ ĐỘC ANDROID 8](#_Toc152137827)

[A. Naive Bayes 8](#_Toc152137828)

[B. Logistic Regression 9](#_Toc152137829)

[C. Random Forest 9](#_Toc152137830)

[D. Support Vector Machines 10](#_Toc152137831)

[E. K-Nearest neighbor (K-NN) 11](#_Toc152137832)

[V. BẢNG SO SÁNH CÁC THUẬT TOÁN 12](#_Toc152137833)

[VI. ĐÀO TẠO VÀ THỬ NGHIỆM 14](#_Toc152137834)

[1. Naive Bayes: 14](#_Toc152137835)

[2. Random Forest 15](#_Toc152137836)

[3. Logistic Regression: 16](#_Toc152137837)

[4. Support Vector Machine 17](#_Toc152137838)

[5. K-Nearest Neighbors 17](#_Toc152137839)

[VII. PHÂN TÍCH SO SÁNH 18](#_Toc152137840)

[VIII. KẾT LUẬN 20](#_Toc152137841)

[TÀI LIỆU THAM KHẢO 21](#_Toc152137842)

# **ABSTRACT**

Hiện nay, việc sử dụng các thiết bị điện thoại thông minh kết nối internet đang gia tăng ổn định và số lượng người dùng ứng dụng Android cũng đang tăng lên. Các thiết bị di động được sử dụng trong các hoạt động hàng ngày như điều khiển ánh sáng từ xa, thanh toán hàng hóa và bãi đỗ xe thông qua thông tin chi tiết của thẻ tín dụng đã được lưu trữ. Do sự tăng trưởng của người dùng ứng dụng Android, một số kẻ xâm nhập tạo ra các ứng dụng Android độc hại nhằm đánh cắp dữ liệu nhạy cảm và tiến hành lừa đảo hoặc giả mạo danh tính. Hiện có nhiều công cụ và phần mềm phát hiện ứng dụng độc hại có sẵn. Tuy nhiên, cần có các công cụ phát hiện ứng dụng độc hại hiệu quả và hiệu suất để xử lý các ứng dụng mới phức tạp do kẻ xâm nhập tạo ra.

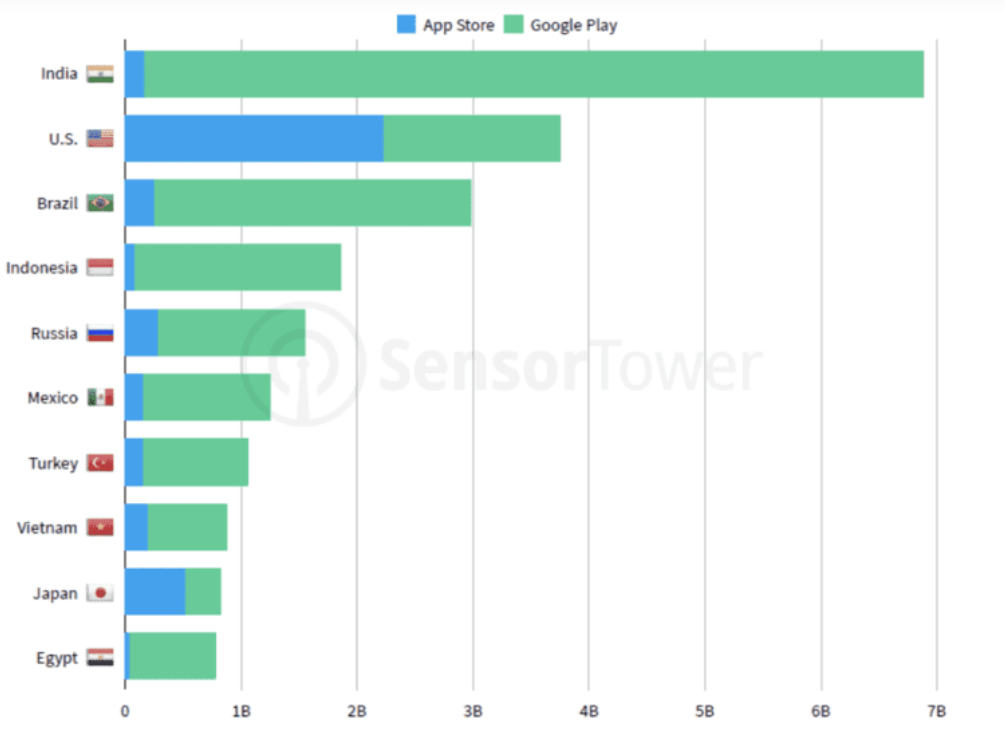
Trong bài báo cáo này, sử dụng các kỹ thuật xử lý ngôn ngữ tự nhiên (NLP) kết hợp với các phương pháp học máy được sử dụng để phát hiện ứng dụng Android độc hại. Ở đây, kỹ thuật tiền xử lý dữ liệu và trích xuất đặc trưng được sử dụng để tăng tốc tính toán của thuật toán. Tiền xử lý dữ liệu được áp dụng cho các đặc trưng của dữ liệu. Nó giúp chuẩn hóa thông tin trong một phạm vi cụ thể. Các kỹ thuật trích xuất đặc trưng như cắt tỉa dữ liệu được sử dụng.

Kỹ thuật xử lý ngôn ngữ tự nhiên (NLP) có thể tăng cường phân tích ứng dụng Android độc hại. NLP có thể được sử dụng để xử lý dữ liệu văn bản liên quan đến các ứng dụng, chẳng hạn như mô tả ứng dụng, đánh giá của người dùng và các quyền được yêu cầu. Bằng cách áp dụng các thuật toán NLP, các nhà nghiên cứu có thể trích xuất các đặc trưng có ý nghĩa từ dữ liệu văn bản, phát hiện các chỉ báo tiềm năng về hành vi độc hại hoặc các mô hình nghi ngờ. Sự kết hợp này giữa NLP và các phương pháp học máy củng cố hệ thống phát hiện phần mềm độc hại tổng thể bằng cách tích hợp phân tích ngôn ngữ và hiểu nội dung văn bản của ứng dụng.

# **I. INTRODUCTION**

Phần mềm độc hại được định nghĩa là phần mềm được thiết kế để tác động và can thiệp vào các ứng dụng di động, lấy đi thông tin chi tiết. Một số phần mềm độc hại chỉ được sử dụng để làm phiền người dùng. Phần mềm độc hại rất nguy hiểm, có thể gây hại máy tính và đánh cắp thông tin tài khoản ngân hàng của chúng ta. Để làm rõ vấn đề này, chúng ta bắt đầu với một bình luận về các loại phần mềm độc hại như virus, worm, Trojan, adware, crime ware, rootkit và spyware.

Điện thoại di động hiện đang là thiết bị điện tử được sử dụng rộng rãi ở khắp mọi nơi. Trên điện thoại di động, chúng ta sử dụng phần mềm Android. Android hiện tại là nền tảng thiết bị di động được sử dụng nhiều nhất trên toàn cầu. Các ứng dụng Android được thực thi thông qua Google Play Store, nơi hàng tỷ ứng dụng được tải xuống.



Hình 1:Số lượt tải xuống các ứng dụng

Trong khảo sát năm 2019 được thực hiện bởi các nhà nghiên cứu tại Check Point, họ phát hiện thú vị rằng số lượng cuộc tấn công mạng nhắm vào điện thoại thông minh và các thiết bị khác đã tăng 50% so với năm trước.

Phần mềm độc hại trên Android có thể có nhiều hình dạng và mối đe dọa khác nhau. Từ phần mềm quảng cáo phiền nhiễu nhưng không gây hại đến phần mềm độc hại tinh vi có khả năng chiếm đoạt thiết bị di động và truy cập vào dữ liệu cá nhân. Trong những năm gần đây, đã tăng đáng kể số lượng phần mềm độc hại được lợi nhuận, chủ yếu dựa trên việc gửi tin nhắn cao cấp bởi các ứng dụng mà không có ý chí và nhận thức của người dùng thiết bị.

Các loại phần mềm độc hại trên Android chủ yếu có thể được phân loại thành tải xuống ngầm, đóng gói lại và tấn công cập nhật. Tuy nhiên, chúng không hoàn toàn độc lập với nhau, vì phần mềm độc hại có thể sử dụng các kỹ thuật khác nhau để lừa người dùng tải xuống. Hơn 80% phần mềm độc hại trên Android là các ứng dụng đã được đóng gói lại. Tấn công cập nhật là một bước tiến trong sự tinh vi của phần mềm độc hại.

Thay vì gắn toàn bộ tải trọng độc hại vào mã ứng dụng, khi ứng dụng đang chạy, toàn bộ tải trọng độc hại sẽ được tải xuống để bổ sung một thành phần cập nhật. Điều này làm cho việc phát hiện phần mềm độc hại trở nên khó khăn hơn, vì quá trình quét tĩnh sẽ không phát hiện được tải trọng độc hại. Cuối cùng, tải xuống ngầm là một kỹ thuật kỹ thuật xã hội cổ điển được áp dụng vào lĩnh vực mới, các thiết bị di động. Nó bao gồm việc lôi kéo người dùng tải xuống các ứng dụng thú vị, nhưng những ứng dụng này sẽ thực hiện những hành động khác ngoài dự đoán.

Phần mềm diệt virus di động chỉ có thể phát hiện được hơn 70% phần mềm độc hại và trong một số trường hợp chỉ 20% trong số chúng. Việc phát triển các giải pháp chống phần mềm độc hại di động mới là một nhu cầu cấp bách.

Vì vậy để giải quyết vấn đề trên nên trong bài báo cáo này, chúng em đề xuất một vài thuật toán có sự kết hợp giữa NLP và Machine Learning để giúp phân loại mã độc Android như: RF (Random Forest) algorithm, LR (Logistic Regression), NB (Naive Bayes), SVM (Support Vector Machines), and K-NN (K\_nearest neighbors). CICMaldroid đã được sử dụng cho công việc này, đây là một dataset tương đối mới và bao gồm các mẫu phần mềm độc hại mới nhất mà có thể phát hiện thành công. Ngoài ra, công việc này mạnh mẽ trong việc xử lý dữ liệu chiều cao, làm cho nó hoạt động hiệu quả khi được sử dụng như một bộ dữ liệu độc lập mới và tương ứng.

# **II. CÁC BÀI BÁO LIÊN QUAN**

## **1. Phân loại mã độc sử dụng phương pháp Deep Learning**

Droid-sec sử dụng Deep Belief Network (DBN) để triển khai công cụ phát hiện phần mềm độc hại trên Android sử dụng phân tích kết hợp. Và Droid-sec đạt được tỷ lệ chính xác 96%.

DroidDeep bắt đầu bằng cách trích xuất năm loại chức năng tĩnh khác nhau từ một số các ứng dụng khác nhau, cả phần mềm tốt và phần mềm độc hại. Mô hình DBN được sử dụng để học các tính năng, cuối cùng, thuật toán máy vectơ hỗ trợ (SVM) được sử dụng để phân loại các mẫu đó.Theo kết quả, DroidDeep đạt được điểm chính xác 99,4%. Hiệu suất phát hiện là được cải thiện trong phiên bản mới của DroidDeep vì sử dụng Deep Learning để trích xuất các tính năng hành vi đáng chú ý từ các ứng dụng. DeepRefiner là một framework dựa trên ngữ nghĩa tự động có thể trích xuất các tính năng liên quan từ mã byte cấp phương thức bằng LSTM với nhiều lớp ẩn. Theo kết quả, độ chính xác của DeepRefiner lên tới 97,74%. Kim và cộng sự giới thiệu phương pháp tiếp cận đa phương thức đầu tiên để phát hiện phần mềm độc hại trên Android bằng cách sử dụng Deep Learning. Theo các phương pháp trích xuất tính năng, họ đã đưa ra hai tính năng mới: dựa trên sự tồn tại và dựa trên sự tương đồng.

## **2. Phân loại mã độc sử dụng NLP**

McLaughlin và cộng sự sử dụng các chuỗi opcode thô có nguồn gốc từ các ứng dụng Android làm tính năng, mạng thần kinh truyền thống đã được sử dụng để phát hiện phần mềm độc hại trên Android. MalDozer đã qua sử dụng trình tự của phương thức API thô làm tính năng tĩnh và phương thức DL được sử dụng để nhận dạng mẫu độc hại và quy kết gia đình của họ. Các thí nghiệm trên quy mô lớn cho thấy MalDozer hoạt động hiệu quả và hiệu quả trong việc phát hiện phần mềm độc hại.

MalDy, báo cáo hành vi của Phần mềm độc hại trên Android được tạo ra bởi công cụ Droidbox và sau đó các tác giả sử dụng túi từ (BoW) để mô hình hóa các báo cáo này thành một chuỗi các từ. MalDy xây dựng một dàn nhạc mô hình phát hiện và phân bổ phần mềm độc hại.

TextDroid là một ứng dụng Android tự động và hiệu quả hệ thống phát hiện phần mềm độc hại tận dụng các kỹ thuật NLP và ML. Phân đoạn từ và mô hình n-gram được sử dụng để mô hình hóa lưu lượng truy cập di động theo chuỗi n-gram và TextDroid áp dụng thuật toán lựa chọn tính năng để nắm bắt các tính năng quan trọng. Sau đó, các tác giả đã phát triển một mô hình phát hiện với thuật toán SVM đạt được hiệu suất tốt trong Tập kiểm tra.

DySign sử dụng phương pháp NLP nâng cao để tự động tạo dấu vân tay từ các hành vi động của phần mềm độc hại Android. DySign đạt được phần mềm độc hại Android nâng cao nhiệm vụ phát hiện với phân loại họ và thực hiện hiệu suất phát hiện tốt với khả năng mở rộng cao trong việc đánh giá tập dữ liệu thực tế.

Nhìn chung, hầu hết NLP hiện có các giải pháp dựa trên sử dụng các phương pháp ML truyền thống, cần có kỹ thuật tính năng. Phương pháp của chúng tôi, mặt khác, sử dụng tính năng dựa trên trình tự lai không yêu cầu tính năng kỹ thuật và sử dụng thuật toán DL tiên tiến. Kết quả là phương pháp của chúng tôi có thể nắm bắt được các tính năng nhạy cảm hơn và do đó đạt được hiệu suất tốt hơn.

## **3. Phần mềm độc hại Android dựa trên chuỗi kết hợp phát hiện bằng xử lý ngôn ngữ tự nhiên**

CoDroid là một hệ thống phân tích kết hợp dựa trên trình tự tự động cho phần mềm độc hại phát hiện trên Android bằng DL được đề xuất. Để tăng cường phát hiện phần mềm độc hại cho nền tảng Android hệ thống, người thực hiện đã trích xuất các tính năng dựa trên trình tự kết hợp các chuỗi opcode với hệ thống trình tự cuộc gọi. Trong nghiên cứu này, người thực hiện sử dụng mô hình CNN–BiLSTM–Attention để tự động tìm hiểu các tính năng nhạy cảm từ ứng dụng Android. Kết quả đánh giá vượt trội hơn các phương pháp tương tự khác và có thể phân biệt phần mềm độc hại với độ chính xác hợp lý.

## **4.** **Phân loại mã độc trên Android với Deep Learning sử dụng RNN từ chuỗi Opcode**

Mô hình đề xuất nhằm mục đích nhận dạng các chuỗi đặc trưng tĩnh. Các ứng dụng Android được phát triển bằng java. Ứng dụng Android có sẵn ở dạng nén dưới dạng tệp zip có phần mở rộng apk. Việc trích xuất tệp zip dẫn đến một số tệp bao gồm meta-inf, res, tệp XML của tệp kê khai android, tệp dex, nội dung. Một tệp dex có thể được dịch ngược để tạo ra một tệp smali. thông tin opcodes được trích xuất từ ​​các tập tin smali. Các tác giả đã sử dụng công cụ python có tên “Androguard” để trích xuất các tính năng [18]. Androguard cung cấp một số lệnh để hoạt động với apk Android. Lệnh “androguard dịch ngược” tạo biểu đồ luồng điều khiển cho ứng dụng Android nhất định. Nó cũng tạo ra các tệp .ag (định dạng giống như smali) cho tất cả các phương thức trong các lớp được dịch ngược (tệp .java). Các tệp .ag được sử dụng để trích xuất các chuỗi opcode. Sau khi thu được các chuỗi opcode Dalvik, mô hình Mạng thần kinh tái phát được áp dụng để học sâu từ các chuỗi này.

## **5.** **Phân loại mã độc sử dụng lời gọi hệ thống**

Một phương pháp sử dụng các chuỗi lệnh gọi hệ thống làm tính năng đã được đề xuất trong bài báo. Sử dụng một bộ sưu tập dấu vết hệ thống thực thi đang được đào tạo, họ sử dụng Machine Learning để xây dựng dấu vân tay. Họ đã thử nghiệm hệ thống của họ trên các thiết bị thực và nhận thấy nó có hiệu quả 97% trong việc phát hiện phần mềm độc hại.

SWORD nắm bắt các lệnh gọi hệ thống từ ứng dụng và sử dụng chuỗi Markov để xây dựng đồ thị lời gọi hệ thống tuần tự. SWORD đạt được độ chính xác 94,2% trong các thử nghiệm trên tập dữ liệu chứa 2000 mẫu Android từ nhiều nguồn khác nhau.

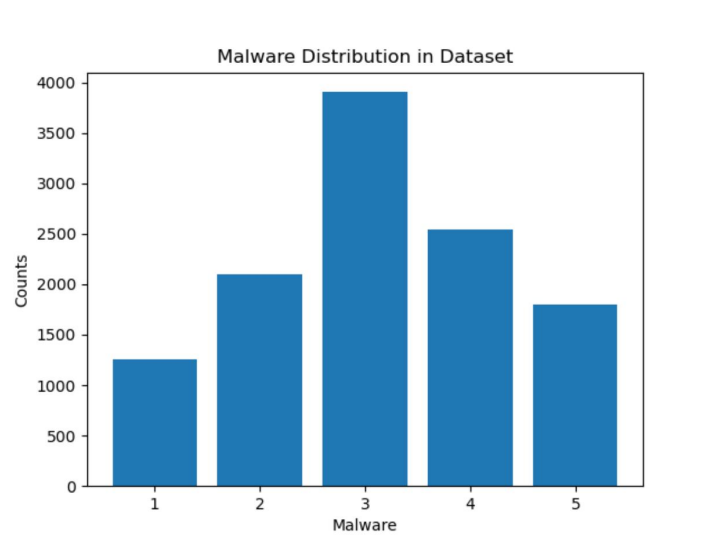
Vinod và cộng sự đã điều tra các lệnh gọi hệ thống với hai phương pháp lựa chọn tính năng để xử lý phần mềm độc hại trên Android. Sự tương tác của con người và đầu vào ngẫu nhiên là được sử dụng để tạo ra các cuộc gọi máy. Ngoài ra, độ bền của máy dò đối với các mẫu đối nghịch đã được nghiên cứu.

DL-Droid là khung phân tích động tự động sử dụng Deep Learning. Thử nghiệm hơn 30.000 ứng dụng trên thiết bị thực đã được tiến hành. Kết quả tiết lộ rằng DL-Droid có thể đạt được hiệu suất tốt hơn chỉ với các tính năng động và với tính năng lai.

# **III. PHÂN TÍCH TẬP DỮ LIỆU**

## **1. Mô tả tập dữ liệu**

Bộ dữ liệu được sử dụng trong nghiên cứu này là CICMalDroid 2020. Nghiên cứu này nhằm tạo ra một hệ thống phát hiện phần mềm độc hại cho ứng dụng Android có khả năng nhận biết phần mềm độc hại trong thời gian thực. Bộ dữ liệu này được thu thập từ nhiều nguồn khác nhau, bao gồm quản lý VirusTotal, blog bảo mật Contagio, AMD và các bộ dữ liệu khác từ các cuộc điều tra đang diễn ra. So với các bộ dữ liệu công khai khác, các mẫu trong bộ dữ liệu này là mới nhất, phức tạp và đa dạng. Chúng được thu thập từ năm 2017 đến 2018. Bộ dữ liệu này được chia thành 5 loại cụ thể, bao gồm Adware, Banking malware, SMS malware, Riskware và Harmless.



Hình 2: Phân phối phần mềm độc hại trong tập dữ liệu

Để phân tích dữ liệu, nghiên cứu sử dụng CopperDroid, một framework điều tra độc đáo dựa trên công nghệ VMI (Virtual Machine Introspection). CopperDroid tự động mô phỏng cách hoạt động của các ứng dụng Android ở cấp độ hệ điều hành và cấp độ cao hơn. Trong quá trình phân tích, 13.077 mẫu trong tổng số 17.341 mẫu đã thành công, trong khi số còn lại không thành công do vấn đề phân bổ bộ nhớ, tệp APK không hợp lệ và vấn đề thời gian chờ. Trong số các mẫu thu thập được, có 3.904 mẫu SMS malware, chiếm gần 34% tổng số mẫu. Ngoài ra, đã tìm thấy 2.546 mẫu Riskware, tương đương khoảng 22% tổng số mẫu. Adware chiếm 1.253 mẫu, tương đương khoảng 11% tổng số mẫu. Banking malware có 2.100 mẫu, chiếm khoảng 18%, và Harmless chiếm 1.795 mẫu, tương đương 15,5%.

Trong CopperDroid, các tệp APK được đánh giá và thông tin về thời gian chạy được lưu trong các tệp nhật ký. Kết quả phân tích của CopperDroid được xuất ra dưới dạng tệp JSON, dễ dàng phân tích và sử dụng kết hợp với dữ liệu bổ sung khác. Kết quả của phân tích bao gồm các thông tin tĩnh như dự đoán, quyền truy cập và dịch vụ, số lần xuất hiện của các loại tệp khác nhau, các sự kiện mờ nhạt và các lệnh API nhạy cảm. Ngoài ra, phân tích cũng bao gồm các cách hoạt động được quan sát một cách động, được chia thành ba nhóm:

Cuộc gọi hệ thống (Framework calls)

Cuộc gọi đến các đối tượng (Folio calls)

Các cách hoạt động tổ hợp (Composite ways of behaving).

Bộ dữ liệu này có 470 đặc trưng và được chia thành ba phần. Phần Composite Ways of behaving chiếm 7,4% dữ liệu, phần Programming interface và Framework calls chiếm 29,6%, và phần Folio Calls chiếm 63,0% tổng số dữ liệu.

## **2. Tiền xử lý tập dữ liệu**

Tiền xử lý dữ liệu (preprocessing) bao gồm chuẩn bị dữ liệu thô cho quá trình phân tích bằng cách chuyển đổi nó thành một dạng hữu ích và hiệu quả hơn. Bước này giúp đảm bảo rằng dữ liệu là chính xác, nhất quán và phù hợp để phân tích. [CHÚ Ý: Bộ dữ liệu không chứa giá trị NULL]. Các bước tiền xử lý đã thực hiện là như sau:

Lựa chọn đặc trưng (Feature Selection)

Một trong những bước đầu tiên trong quy trình tiền xử lý là lựa chọn đặc trưng. Bộ dữ liệu có một số lượng lớn các đặc trưng, có thể dẫn đến tình trạng overfitting và tăng độ phức tạp tính toán. Để giảm những vấn đề này, sử dụng phương pháp lựa chọn đặc trưng dựa trên ANOVA. Phương pháp này xếp hạng các đặc trưng dựa trên giá trị F của chúng liên quan đến biến mục tiêu và chọn ra các đặc trưng hàng đầu có mối quan hệ quan trọng nhất.

Sau khi áp dụng phương pháp lựa chọn đặc trưng dựa trên ANOVA, đã giảm không gian đặc trưng từ 470 xuống còn 120 đặc trưng, giữ lại những đặc trưng góp phần nhiều nhất vào việc giải thích phương sai trong biến mục tiêu. Quá trình lựa chọn này nhằm cải thiện hiệu suất của mô hình, giảm số chiều dữ liệu trong khi vẫn giữ lại thông tin quan trọng nhất.

Xử lý dữ liệu phân loại (Categorical Data)

Bộ dữ liệu chứa các biến phân loại, cần được chuyển đổi thành biểu diễn số để phù hợp với mô hình. Để giải quyết vấn đề này bằng cách sử dụng các kỹ thuật mã hóa phân loại phù hợp. Tuy nhiên, do việc chuyển đổi trước các giá trị phân loại thành giá trị số và không yêu cầu mã hóa hoặc xóa các hàng nữa, không cần thực hiện xử lý bổ sung trong vấn đề này.

Xử lý giá trị thiếu (Missing Values)

Bộ dữ liệu ban đầu đã được kiểm tra kỹ lưỡng để phát hiện giá trị thiếu. May mắn thay, không phát hiện giá trị thiếu nào, điều này giúp không phải đối mặt với các kỹ thuật khắc phục giá trị thiếu. Sự vắng mặt của giá trị thiếu đảm bảo rằng các mô hình của có thể được huấn luyện trên dữ liệu đầy đủ mà không gây ra thiên hướng hoặc lỗi.

## **3. Chia tỷ lệ tính năng**

Thực hiện chuẩn hóa đặc trưng là một bước quan trọng trong quy trình tiền xử lý, đặc biệt khi làm việc với các thuật toán học máy nhạy cảm đến tỷ lệ của các đặc trưng đầu vào. Trong báo cáo này, sử dụng StandardScaler để chuẩn hóa các đặc trưng và đảm bảo rằng chúng có cùng tỷ lệ so sánh được.

StandardScaler là một phương pháp phổ biến để chuẩn hóa đặc trưng, vì nó biến đổi dữ liệu sao cho có trung bình bằng 0 và độ lệch chuẩn bằng 1. Kỹ thuật chuẩn hóa này đặc biệt hữu ích khi các đặc trưng có phạm vi hoặc đơn vị khác nhau, điều này có thể dẫn đến việc huấn luyện mô hình bị thiên vị.

Áp dụng StandardScaler vào bộ dữ liệu sau bước lựa chọn đặc trưng. Điều này đảm bảo rằng tập hợp 120 đặc trưng đã được chuẩn hóa, cho phép các mô hình học máy hoạt động tối ưu mà không bị ảnh hưởng bởi tỷ lệ khác nhau của các đặc trưng.

Triển khai StandardScaler từ thư viện scikit-learn. Sau khi khớp scaler với dữ liệu huấn luyện, áp dụng các tham số chuẩn hóa đã học được vào cả tập dữ liệu huấn luyện và kiểm tra. Điều này đảm bảo rằng các mô hình được đánh giá trên dữ liệu đã được chuẩn hóa để cung cấp các chỉ số hiệu suất chính xác.

Bằng cách tuân thủ việc khớp scaler với dữ liệu huấn luyện và chỉ biến đổi dữ liệu kiểm tra, mô phỏng tình huống thực tế khi mô hình được tiếp xúc với dữ liệu mới, chưa được nhìn thấy trước. Mô hình nên được đánh giá dựa trên khả năng tổng quát hóa với dữ liệu mới này, và việc sử dụng các tham số chuẩn hóa đã học từ dữ liệu huấn luyện giúp duy trì sự phân tách giữa quá trình huấn luyện và kiểm tra, ngăn chặn rò rỉ dữ liệu và đảm bảo đánh giá hiệu suất chính xác.

## **4. Tách tập dữ liệu**

Để đánh giá hiệu suất của các mô hình học máy một cách chính xác, đã chia bộ dữ liệu thành hai phần: tập huấn luyện và tập kiểm tra. Sử dụng hàm train\_test\_split từ thư viện scikit-learn cho mục đích này. Bộ dữ liệu có tổng cộng 11.598 điểm dữ liệu.

Chia dữ liệu theo phương pháp Stratified Splitting:

Với sự quan trọng của việc duy trì tính toàn vẹn phân phối các lớp trong cả tập huấn luyện và tập kiểm tra, thực hiện phân chia theo phương pháp stratified. Phân chia theo phương pháp stratified đảm bảo tỷ lệ các lớp khác nhau trong bộ dữ liệu gốc được bảo toàn trong tập huấn luyện và tập kiểm tra. Điều này đặc biệt quan trọng khi làm việc với các bộ dữ liệu mất cân bằng để ngăn chặn một lớp bị thiếu kéo dài trong dữ liệu huấn luyện hoặc kiểm tra.

Tỷ lệ phân chia:

Chọn tỷ lệ phân chia 80-20, với 80% dữ liệu được phân vào tập huấn luyện và 20% vào tập kiểm tra. Tỷ lệ này đạt được sự cân bằng giữa việc có đủ dữ liệu để huấn luyện mô hình và một lượng dữ liệu hợp lý để đánh giá khả năng tổng quát hóa của mô hình.

Lợi ích của phân chia theo phương pháp stratified:

Phân chia theo phương pháp stratified mang lại một số lợi ích:

* Bảo toàn phân phối các lớp: Bằng cách đảm bảo phân phối các lớp được duy trì nhất quán trong cả tập huấn luyện và tập kiểm tra, ngăn chặn sự thiên vị và cải thiện việc đánh giá mô hình trên dữ liệu đa dạng.
* Giảm rủi ro quá khớp: Mô hình ít có khả năng quá khớp khi được đánh giá trên dữ liệu kiểm tra gần giống với phân phối thực tế trong thực tế.

Phân chia cuối cùng:

Sau khi áp dụng phân chia theo phương pháp stratified, thu được tập huấn luyện với 9278 điểm dữ liệu và tập kiểm tra với 2320 điểm dữ liệu.

# **IV. CÁC THUẬT TOÁN PHÂN LOẠI MÃ ĐỘC ANDROID**

## **A. Naive Bayes**

Thuật toán Naive Bayes là một thuật toán phân loại dựa trên nguyên tắc của lý thuyết xác suất và giả định "naive" về sự độc lập giữa các đặc trưng. Nó được sử dụng để phân loại các mẫu dữ liệu vào các lớp khác nhau dựa trên các đặc trưng của chúng.

Kết hợp machine learning và xử lý ngôn ngữ tự nhiên (NLP), thuật toán Naive Bayes sử dụng dữ liệu văn bản để phân loại. Quá trình này bao gồm:

* Tiền xử lý dữ liệu: Dữ liệu văn bản được chuyển đổi thành các biểu diễn số học, chẳng hạn như vectơ đặc trưng. Các bước tiền xử lý có thể bao gồm loại bỏ các ký tự đặc biệt, chuyển đổi văn bản thành vectơ từ, và chuẩn hóa dữ liệu.
* Xây dựng mô hình: Dữ liệu văn bản được sử dụng để xây dựng mô hình Naive Bayes. Mô hình này học từ dữ liệu huấn luyện để tính toán xác suất của từng lớp và xác suất điều kiện của từng đặc trưng trong lớp.
* Phân loại: Khi có một mẫu dữ liệu mới, mô hình tính toán xác suất dự đoán cho từng lớp dựa trên xác suất lớp và xác suất điều kiện. Lớp có xác suất cao nhất sẽ được chọn là kết quả phân loại của mẫu dữ liệu.

Trong thuật toán Naive Bayes, tính toán xác suất của một tệp tin thuộc về một lớp cụ thể dựa trên nhiều độ đo được thu thập từ tập dữ liệu huấn luyện. Một tệp tin có xác suất cao có thể là một phần mềm không độc hại, ngược lại một tệp tin có xác suất thấp là phần mềm độ hại.

Để thực hiện dự đoán, ta tính toán xác suất của từng lớp và xác suất điều kiện của từng mẫu dữ liệu. Xác suất tích lũy giúp dự đoán lớp của mẫu dữ liệu, trong đó xác suất lớp và xác suất mẫu dữ liệu được tính bằng phép nhân.

Cụ thể, quá trình tính toán xác suất trong Naive Bayes là như sau:

* Xác suất lớp (P(class)): Đầu tiên, tính toán xác suất của từng lớp dựa trên tần số xuất hiện của lớp đó trong tập dữ liệu huấn luyện.
* Xác suất điều kiện (P(data|class)): Tiếp theo, tính toán xác suất điều kiện của mẫu dữ liệu dựa trên các đặc trưng của nó và lớp tương ứng. Đối với mỗi đặc trưng, tính toán xác suất của giá trị đó xuất hiện trong lớp cụ thể, dựa trên tần số xuất hiện của giá trị đó trong tập dữ liệu huấn luyện.
* Xác suất tổng cộng (P(data)): Tính toán xác suất tổng cộng của mẫu dữ liệu bằng cách nhân các xác suất điều kiện với nhau và nhân với xác suất lớp tương ứng.
* Xác suất dự đoán (P(class|data)): Cuối cùng, tính toán xác suất dự đoán của mỗi lớp cho mẫu dữ liệu dựa trên xác suất tổng cộng và xác suất lớp. Xác suất dự đoán cao nhất cho lớp đó sẽ được chọn là kết quả dự đoán.

Naive Bayes thường được áp dụng trong các tác vụ phân loại văn bản, như phân loại tin tức, phân loại email, và phân loại ngôn ngữ tự nhiên khác.

## **B. Logistic Regression**

Thuật toán Logistic Regression là một thuật toán phân loại dựa trên phân tích hồi quy logistic. Nó được sử dụng để phân loại các mẫu dữ liệu vào các lớp khác nhau dựa trên các đặc trưng của chúng.

Kết hợp machine learning và xử lý ngôn ngữ tự nhiên (NLP), thuật toán Logistic Regression Classifiers sử dụng dữ liệu văn bản để phân loại. Quá trình này bao gồm:

* Tiền xử lý dữ liệu: Dữ liệu văn bản được chuyển đổi thành các biểu diễn số học, chẳng hạn như vectơ đặc trưng. Các bước tiền xử lý có thể bao gồm loại bỏ các ký tự đặc biệt, chuyển đổi văn bản thành vectơ từ, và chuẩn hóa dữ liệu.
* Xây dựng mô hình: Dữ liệu văn bản được sử dụng để xây dựng mô hình Logistic Regression. Mô hình này học từ dữ liệu huấn luyện để tìm ra các trọng số tối ưu cho các đặc trưng, sao cho mô hình có khả năng dự đoán chính xác lớp của các mẫu dữ liệu.
* Huấn luyện mô hình: Mô hình Logistic Regression được huấn luyện bằng cách tối ưu hàm mất mát (loss function) thông qua các phương pháp tối ưu hóa như gradient descent. Mục tiêu là tìm ra các trọng số mô hình sao cho mô hình có khả năng dự đoán tốt nhất.
* Phân loại: Khi có một mẫu dữ liệu mới, mô hình tính toán xác suất dự đoán rơi vào từng lớp dựa trên các trọng số đã được học. Mẫu dữ liệu sẽ được phân loại vào lớp có xác suất cao nhất.

Kết quả cuối cùng là một mô hình phân loại dựa trên các trọng số và thông tin văn bản của dữ liệu đầu vào. Logistic Regression Classifiers thường được sử dụng trong các tác vụ phân loại văn bản, như phân loại cảm xúc, phân loại chủ đề, và phân loại văn bản tự nhiên khác.

## **C. Random Forest**

Random Forest là một thuật toán học máy đám mây ngẫu nhiên, được sử dụng cho các tác vụ phân loại và hồi quy. Nó là một phương pháp tổ hợp của nhiều cây quyết định (decision trees) độc lập, gọi là cây con. Các cây con được xây dựng bằng cách lấy mẫu ngẫu nhiên từ tập dữ liệu huấn luyện và các đặc trưng của chúng.

Mỗi cây con trong Random Forest được huấn luyện trên một tập dữ liệu con duy nhất, được lấy mẫu từ tập dữ liệu huấn luyện gốc. Quá trình xây dựng cây con sử dụng các quy tắc phân chia để tách các mẫu dữ liệu vào các nhánh khác nhau dựa trên các đặc trưng. Khi có một mẫu dữ liệu mới, các cây con đưa ra dự đoán độc lập và kết quả được đưa ra bằng cách sử dụng phương pháp voting hoặc averaging trên các kết quả của các cây con.

Ứng dụng trong phân loại mã độc android:

Trong việc phân loại mã độc Android, Random Forest có thể được áp dụng để xây dựng một hệ thống phân loại đa lớp để phân biệt giữa các loại mã độc và ứng dụng không độc hại.

Quá trình ứng dụng Random Forest trong phân loại mã độc Android có thể được mô tả như sau:

* Tiền xử lý dữ liệu: Đầu tiên, các mẫu mã độc và không độc hại cần được thu thập và tiền xử lý. Các đặc trưng của từng mẫu được trích xuất, ví dụ: các quyền truy cập, hành vi gọi API, các tệp tin liên quan, các thông tin phân tích tĩnh, v.v.
* Xây dựng tập dữ liệu huấn luyện: Dữ liệu được chia thành hai phần, một phần dùng để huấn luyện và một phần dùng để đánh giá hiệu suất của mô hình. Các mẫu dữ liệu được gán nhãn là mã độc hoặc không độc hại.
* Xây dựng Random Forest: Một tập các cây quyết định con được xây dựng bằng cách lấy mẫu ngẫu nhiên từ tập dữ liệu huấn luyện. Mỗi cây con trong Random Forest được xây dựng bằng cách sử dụng các đặc trưng và nhãn tương ứng của các mẫu dữ liệu.
* Huấn luyện Random Forest: Các cây con trong Random Forest được huấn luyện thông qua quy trình xây dựng cây quyết định thông thường, sử dụng các phương pháp như ID3 (Chọn thuộc tính có độ thông tin cao nhất), C4.5 ( Chọn thuộc tính có độ thông tin cao nhất và có thể giảm thiểu entropy của dữ liệu) hoặc CART (Chọn thuộc tính có thể tối đa hóa độ chênh lệch giữa các lớp). Các cây con được điều chỉnh để tối ưu hóa độ chính xác và khả năng phân loại.
* Phân loại mã độc Android: Khi có một mẫu mã độc Android mới, các cây con trong Random Forest đưa ra dự đoán độc lập. Kết quả dự đoán được tính toán bằng cách sử dụng phương pháp voting hoặc averaging trên các dự đoán của các cây con. Kết quả cuối cùng cho biết liệu mẫu là mã độc hay không độc hại.

Kết hợp giữa NLP và machine learning trong việc phân loại mã độc Android có thể bao gồm việc sử dụng các đặc trưng liên quan đến văn bản như các chuỗi ký tự, tên gọi, lời nhắn, v.v. NLP có thể được sử dụng để xử lý và trích xuất thông tin từ các văn bản liên quan đến mã độc, như tài liệu hướng dẫn, bình luận từ người dùng, và các tài liệu liên quan khác. Các đặc trưng từ NLP có thể được kết hợp với các đặc trưng khác (như quyền truy cập, hành vi) và được sử dụng làm đầu vào cho Random Forest để phân loại mã độc Android.

## **D. Support Vector Machines**

Thuật toán Support Vector Machines (SVM) là một thuật toán học máy phân loại được sử dụng để phân tách dữ liệu thành hai hoặc nhiều lớp. SVM hoạt động bằng cách tìm kiếm một hyperplane tối ưu có thể phân tách hai lớp dữ liệu với khoảng cách lớn nhất giữa các điểm dữ liệu của hai lớp.

* Hyperplane trong SVM là một đường thẳng trong không gian đa chiều. Hyperplane này được sử dụng để phân tách hai lớp dữ liệu thành hai phần riêng biệt. Khoảng cách giữa hyperplane và các điểm dữ liệu của hai lớp càng lớn thì khả năng phân loại sai càng nhỏ.

Trong trường hợp phân loại mã độc Android, hyperplane sẽ phân tách mã độc và mã sạch thành hai phần riêng biệt. Khoảng cách lớn nhất giữa hyperplane và các điểm dữ liệu của hai lớp sẽ giúp giảm thiểu khả năng phân loại sai. SVM sẽ học các đặc trưng của mã độc dựa trên một tập dữ liệu gồm các ứng dụng Android đã được phân loại là mã độc hoặc mã sạch. Sau khi được đào tạo, SVM có thể sử dụng các đặc trưng này để phân loại các ứng dụng Android mới chưa được phân loại.

Hyperplane trong SVM được tính toán bằng cách sử dụng một thuật toán tối ưu hóa. Thuật toán này sẽ tìm kiếm hyperplane có thể đáp ứng cả hai điều kiện sau:

* Phân tách hai lớp dữ liệu: Hyperplane phải phân tách hai lớp dữ liệu thành hai phần riêng biệt.
* Khoảng cách lớn nhất: Khoảng cách giữa hyperplane và các điểm dữ liệu của hai lớp phải lớn nhất có thể.

Để tìm kiếm hyperplane tối ưu, SVM sử dụng một số thuật toán khác nhau, bao gồm:

* Thuật toán phân tích miền: Thuật toán này chia không gian dữ liệu thành các miền nhỏ hơn và sau đó tìm hyperplane trong mỗi miền.
* Thuật toán quy hoạch tuyến tính: Thuật toán này sử dụng các phương pháp tối ưu hóa tuyến tính để tìm hyperplane.
* Thuật toán kernel: Thuật toán này sử dụng các hàm kernel để chuyển đổi không gian dữ liệu thành một không gian mới, nơi hyperplane có thể được tìm thấy dễ dàng hơn.
* Thuật toán Support Vector Machines (SVM) là một thuật toán học máy phân loại được sử dụng để phân tách dữ liệu thành hai hoặc nhiều lớp. SVM hoạt động bằng cách tìm kiếm một hyperplane tối ưu có thể phân tách hai lớp dữ liệu với khoảng cách lớn nhất giữa các điểm dữ liệu của hai lớp.

SVM kết hợp machine learning và NLP như sau:

Machine learning được sử dụng để học các đặc trưng của mã độc từ một tập dữ liệu đã được phân loại. Các đặc trưng này có thể bao gồm các thông tin về mã nguồn của ứng dụng, các hoạt động của ứng dụng, hoặc các mối quan hệ giữa các ứng dụng.

NLP được sử dụng để phân tích mã nguồn của ứng dụng và xác định các đặc trưng liên quan đến mã độc. NLP có thể giúp phát hiện các mẫu ngôn ngữ độc hại, chẳng hạn như các hàm hoặc cấu trúc dữ liệu thường được sử dụng trong mã độc.

## **E. K-Nearest neighbor (K-NN)**

Thuật toán K-Nearest Neighbor (KNN) là một thuật toán học máy phân loại được sử dụng để phân loại một điểm dữ liệu mới dựa trên các điểm dữ liệu đã được phân loại. KNN hoạt động bằng cách tìm kiếm K điểm dữ liệu gần nhất với điểm dữ liệu mới và sau đó phân loại điểm dữ liệu mới dựa trên các điểm dữ liệu gần nhất đó.

Trong việc phân loại mã độc Android:

KNN hoạt động dựa trên giả định rằng các điểm dữ liệu gần nhau về mặt không gian thường có các đặc trưng tương tự nhau. Trong trường hợp phân loại mã độc Android, các điểm dữ liệu trong không gian là các ứng dụng Android.

Khi một ứng dụng Android mới được tải lên, KNN sẽ tính toán khoảng cách giữa ứng dụng mới và mỗi ứng dụng Android trong tập dữ liệu. Ứng dụng Android mới sẽ được phân loại là mã độc nếu nó nằm trong K ứng dụng Android gần nhất với các ứng dụng độc hại trong tập dữ liệu.

Có một số yếu tố cần xem xét khi sử dụng KNN để phân loại mã độc Android:

* Chọn giá trị K phù hợp. Giá trị K càng lớn thì KNN sẽ càng cẩn thận khi phân loại các ứng dụng mới. Tuy nhiên, giá trị K càng lớn thì KNN cũng sẽ càng chậm.
* Cách tính toán khoảng cách giữa các ứng dụng. KNN có thể sử dụng nhiều phương pháp khác nhau để tính toán khoảng cách, chẳng hạn như khoảng cách Euclid hoặc khoảng cách Manhattan.

KNN kết hợp machine learning và NLP như sau:

Machine learning được sử dụng để học các đặc trưng của mã độc từ một tập dữ liệu đã được phân loại. Các đặc trưng này có thể bao gồm các thông tin về mã nguồn của ứng dụng, các hoạt động của ứng dụng, hoặc các mối quan hệ giữa các ứng dụng.

NLP được sử dụng để phân tích mã nguồn của ứng dụng và xác định các đặc trưng liên quan đến mã độc. NLP có thể giúp phát hiện các mẫu ngôn ngữ độc hại, chẳng hạn như các hàm hoặc cấu trúc dữ liệu thường được sử dụng trong mã độc.

# **V. BẢNG SO SÁNH CÁC THUẬT TOÁN**

|  |  |  |
| --- | --- | --- |
| **Thuật toán** | **Ưu điểm** | **Hạn chế** |
| **Naive bayes** | Việc triển khai rất đơn giản và dễ hiểu. NB có thể được huấn luyện trên các tập dữ liệu nhỏ, không liên quan và vẫn đạt được kết quả tốt. Do đó, NB là một lựa chọn tốt cho các ứng dụng mà dữ liệu bị hạn chế. | Khi các đặc trưng của dữ liệu trong tập dữ liệu được đào tạo là tương quan với nhau, trình phân loại NB sẽ không thể học hỏi từ dữ liệu một cách hiệu quả và sẽ không hoạt động tốt. |
| **K-Nearest Neighbour (K-NN)** | Việc cập nhật có thể được thực hiện với chi phí rất thấp. Nó rất dễ để triển khai. Xử lý nhiều loại tập dữ liệu, bao gồm cả những tập dữ liệu có quy mô, định dạng hoặc ngoại lệ khác nhau. Nó không cần bất kỳ thời gian đào tạo nào. | Khi tập dữ liệu không đồng đều được phân phối và tập dữ liệu lớn, nó không hoạt động tốt (do đặc trưng của thuật toán là dựa vào điểm dữ liệu kề cận để dự đoán). Yêu cầu tính năng chia tỷ lệ gây tốn thời gian. |
| **Logistic Regression** | Nó không đòi hỏi cao sức mạnh tính toán. Ít phức tạp và rất đơn giản để triển khai. Có thể được đào tạo một cách hiệu quả để phân loại dữ liệu. Nó có thể dễ dàng phân loại dữ liệu chưa biết. Đạt được độ chính xác cao khi được áp dụng cho các tập dữ liệu đơn giản. | Có thể không hiệu quả khi được áp dụng cho các tập dữ liệu lớn hoặc phi tuyến. Có thể dẫn đến các dự đoán không chính xác khi được áp dụng cho các tập dữ liệu phức tạp. |
| **Support Vector Machine** | Có thể đạt được độ chính xác cao hơn so với các thuật toán phân loại khác. Hoạt động hiệu quả hơn với các tập dữ liệu có nhiều tính năng. Sử dụng bộ nhớ hiệu quả. Nó có thể xử lý hiệu quả. Có thể xử lý hiệu quả các tập dữ liệu phi tuyến tính và ít bị ảnh hưởng bởi nhiễu hơn so với các thuật toán khác. | Phải mất nhiều thời gian hơn để đào tạo mô hình. Con người rất khó giải thích và hiểu được. Nó không thể xử lý các tập dữ liệu lớn. Khi nhiễu nhiều hơn thì không thể xử lý tốt. |
| **Random Forest** | Giảm thiểu vấn đề quá khớp khi một mô hình trở nên quá khớp với dữ liệu huấn luyện và hoạt động kém trên dữ liệu chưa thấy. Đạt được độ chính xác cao. Xử lý hiệu quả các loại dữ liệu khác nhau. Các giá trị thiếu trong dữ liệu có thể tự động xử lý, không yêu cầu chia tỷ lệ tính năng giúp đơn giản hóa quá trình chuẩn bị dữ liệu và giảm thiểu nguy cơ lỗi. | Nó đòi hỏi nhiều thời gian hơn để huấn luyện mô hình. Cần bộ nhớ lớn hơn để xử lý thuật toán. Nếu có sự thay đổi nhỏ trong dữ liệu, nó có thể ảnh hưởng đến quá trình. Tính toán rất phức tạp khi so sánh với những thuật toán khác vì phải đánh giá nhiều cây quyết định và tổng hợp dự đoán. |

Bảng 1: Bảng so sánh các thuật toán

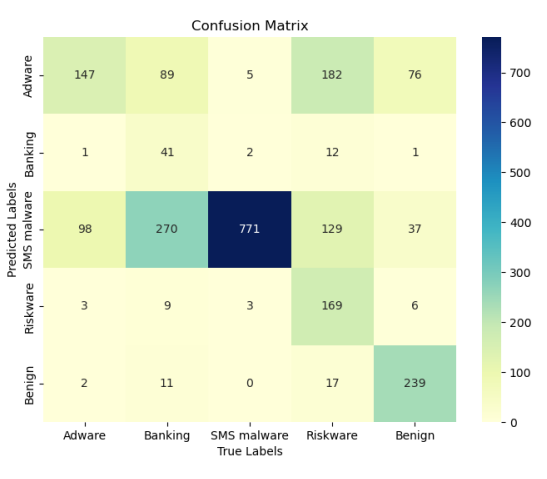
# **VI. ĐÀO TẠO VÀ THỬ NGHIỆM**

* Accuracy là tỷ lệ giữa số điểm dữ liệu được phân loại đúng với tổng số điểm dữ liệu. Accuracy là một chỉ số tổng quát, nhưng nó có thể không phản ánh chính xác hiệu suất của mô hình trong các trường hợp cụ thể.
* Precision là tỷ lệ giữa số điểm dữ liệu độc hại được phân loại đúng với tổng số điểm dữ liệu được phân loại là độc hại. Precision cao cho thấy mô hình ít phân loại nhầm mã sạch là mã độc.
* Recall là tỷ lệ giữa số điểm dữ liệu độc hại được phân loại đúng với tổng số điểm dữ liệu độc hại. Recall cao cho thấy mô hình ít bỏ sót mã độc.
* F1-Score là trung bình điều hòa của Precision và Recall. F1-Score cao cho thấy mô hình có cả Precision và Recall cao.

Đào tạo và thử nghiệm tập dữ liệu trên năm mô hình khác nhau. Chi tiết đào tạo và kiểm tra như sau:

## **1. Naive Bayes:**

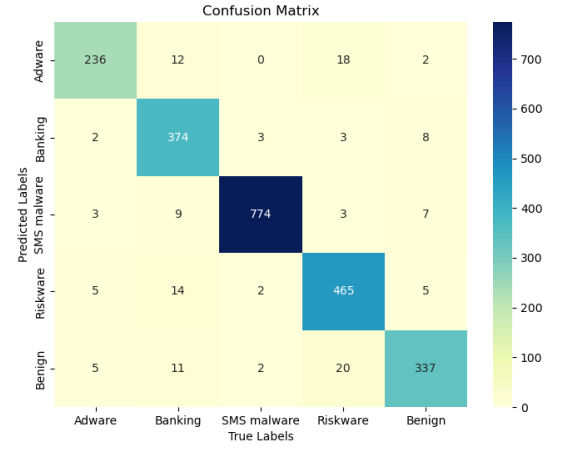
* Naive Bayes Classifier Accuracy: 0.5892
* Naive Bayes Classifier Precision: 0.6936
* Naive Bayes Classifier Recall: 0.5892
* Naive Bayes Classifier F1-Score: 0.5462



Hình 3: Ma trận Heatmap của Naive Bayes

## **2. Random Forest**

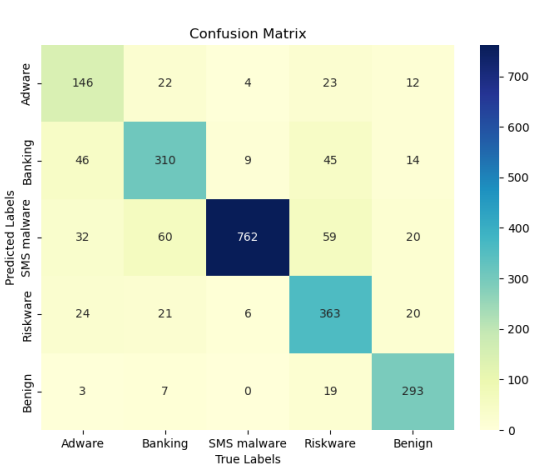
* Random Forest Classifier Accuracy: 0.9422
* Random Forest Classifier Precision: 0.9431
* Random Forest Classifier Recall: 0.9422
* Random Forest Classifier F1-Score: 0.9421



Hình 4: Ma trận Heatmap của Random Forest

## **3. Logistic Regression:**

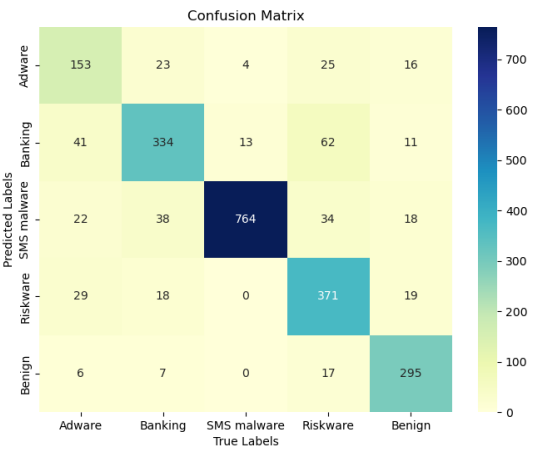
* Logistic Regression Accuracy: 0.8078
* Logistic Regression Precision: 0.8079
* Logistic Regression Recall: 0.8078
* Logistic Regression F1-Score: 0.8033



Hình 5: Ma trận Heatmap của Logistic Regression

## **4. Support Vector Machine**

* SVM Classifier Accuracy: 0.8263
* SVM Classifier Precision: 0.8264
* SVM Classifier Recall: 0.8263
* SVM Classifier F1-Score: 0.8234

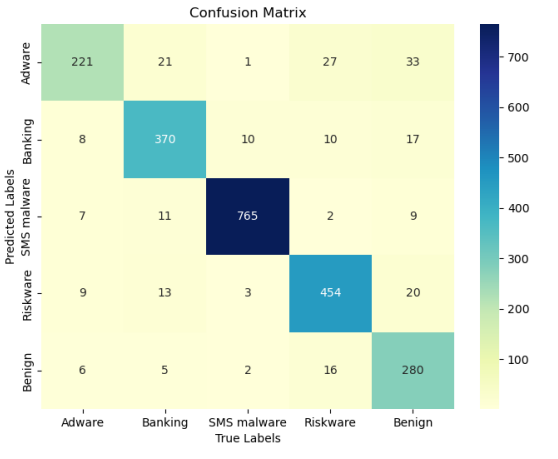


Hình 6: Ma trận Heatmap của Support Vector Machine

## **5. K-Nearest Neighbors**

Best k: 3

* K-Nearest Neighbors Classifier Accuracy: 0.9009
* K-Nearest Neighbors Classifier Precision: 0.9045
* K-Nearest Neighbors Classifier Recall: 0.9009
* K-Nearest Neighbors Classifier F1-Score: 0.9011



Hình 7: Ma trận Heatmap của KNN

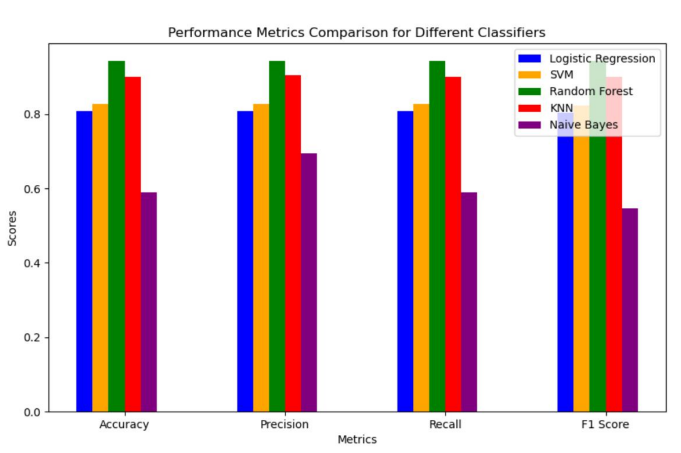
# **VII. PHÂN TÍCH SO SÁNH**

Sau khi đào tạo và thử nghiệm trên tất cả các mô hình, đạt kết quả như sau:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Models** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| **Logistic Regression** | 0.807759 | 0.807916 | 0.807759 | 0.803350 |
| **SVM** | 0.826293 | 0.826378 | 0.826293 | 0.823396 |
| **Random Forest** | **0.942241** | **0.943053** | **0.942241** | **0.942148** |
| **KNN** | 0.900862 | 0.904487 | 0.900862 | 0.901129 |
| **Naive Bayes** | 0.5892 | 0.6936 | 0.5892 | 0.5462 |

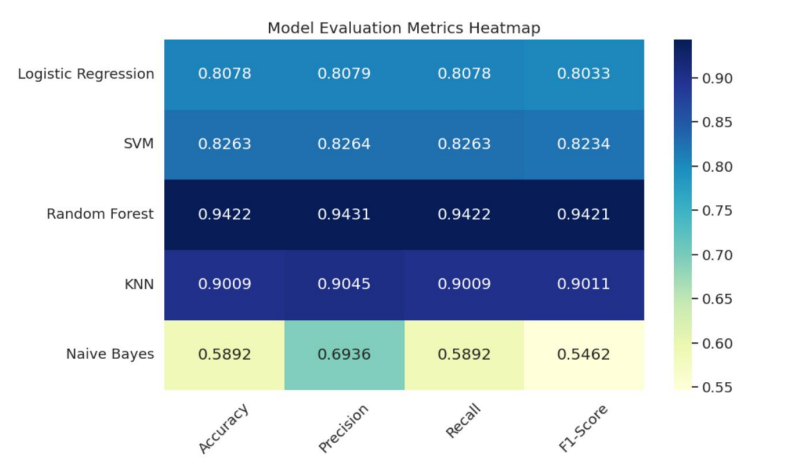
Bảng 2: So sánh điểm số liệu của các mô hình khác nhau

Từ bảng, thấy rằng Random Forest hoạt động tốt nhất trong thử nghiệm dữ liệu trong khi naive bayes có điểm ít nhất trong thử nghiệm. Hơn nữa, KNN cũng đã đạt được điểm số tốt. Điểm số được vẽ như sau cho so sánh trực quan:



Hình 8: So sánh ma trận hiệu suất cho các bộ phân loại khác nhau

Vì vậy, Random Forest có thể phát hiện phần mềm độc hại tiềm ẩn từ dữ liệu với độ chính xác cao hơn. Có thể hình dung rõ ràng sự khác biệt từ biểu đồ so sánh sau:



Hình 9: Sơ đồ Heatmap của số liệu đánh giá mô hình

# **VIII. KẾT LUẬN**

Tầm quan trọng của việc phát hiện phần mềm độc hại hiệu quả đã tăng lên đáng kể trong việc đảm bảo tính bảo mật và quyền riêng tư của các thiết bị di động được kết nối với mạng. Nhu cầu này nảy sinh do sự phát triển và tăng cường không ngừng của phần mềm độc hại, đặc biệt là trong lĩnh vực điện thoại di động Android, vốn đang có số lượng người dùng tăng lên đáng kể. Trong khi đã đạt được tiến bộ đáng kể trong việc phát hiện xâm nhập để bảo vệ quyền riêng tư và bảo vệ các thiết bị điện tử, các tác nhân độc hại cũng đang phát triển nhanh chóng, đặt ra những thách thức nghiêm trọng đối với an ninh mạng. Do đó, nhu cầu về các biện pháp an ninh mạnh mẽ hơn đã trở nên cấp thiết.

Ngoài ra, dòng phần mềm độc hại tinh vi xâm nhập vào các thiết bị cá nhân đang diễn ra với tốc độ nhanh chóng, vượt qua khả năng của các hệ thống phát hiện xâm nhập hiện có vốn nổi tiếng là chậm chạp và kém hiệu quả.

Các kỹ thuật NLP hiện đại đã được cải tiến rất nhiều và được sử dụng trong thực hành để hoàn thành các nhiệm vụ khác nhau, chẳng hạn như dịch máy, tóm tắt các văn bản lớn hơn và trả lời câu hỏi, v.v

Trong bài báo cáo này, đã khai thác thực tế này và áp dụng kỹ thuật NLP kết hợp với Machine learning thông qua các thuật toán RF (Random Forest) algorithm, LR (Logistic Regression), NB (Naive Bayes), SVM (Support Vector Machines), và K-NN (K\_nearest neighbors) để tìm các mẫu độc hại trong ứng dụng Android. Và cho thấy rằng thuật toán RF (Random Forest) cho kết quả phát hiện mã độc Android tốt nhất.

Việc giới thiệu tính năng phát hiện phần mềm độc hại theo thời gian thực hứa hẹn sẽ giải quyết được những vấn đề này do tính chất linh hoạt và nhanh chóng của nó, mặc dù lĩnh vực này vẫn còn tương đối chưa được khám phá.

Những cải tiến trong tương lai có thể liên quan đến việc đưa vào ứng dụng đã phát triển các thuộc tính động. Đáng chú ý, nghiên cứu hiện tại của chúng tôi tập trung vào việc xác định phần mềm độc hại trong các ứng dụng Android, nhưngcó khả năng mở rộng phạm vi để bao gồm các ứng dụng được nền tảng iOS hỗ trợ, sử dụng các bộ dữ liệu có liên quan.

# **TÀI LIỆU THAM KHẢO**

1. Hybrid sequence‐based Android malware detection using natural language processing

2. Hybrid sequence‐based Android malware detection using natural language processing. International Journal of Intelligent Systems

3. https://online-journals.org/index.php/i-jim/article/view/26433/10561

4. MalDozer: Automatic framework for android malware detection using deep learning

5. A Survey On Detecting Android Malware Using Machine Learning Technique

6. Detection of Potential Malware in Android Devices

7. https://gs.statcounter.com/os-market-share/mobile/worldwide

8. Android malware detection using improvised random forest algorithm

9. Improving nearest neighbor rule with a simple adaptive distance measure

10. An Android Malware Detection Model Based on DT-SVM

11. Android malware detection method based on naive Bayes and permission correlation algorithm

12. Dataset: <https://www.unb.ca/cic/datasets/maldroid-2020.html>.