Học máy khả diễn giải trong việc phát hiện Android malware

Alani, M. M., & Awad, A. I. (2022). Paired: An explainable lightweight android malware detection system. IEEE Access, 10, 73214-73228.

1 GIỚI THIỆU ĐỀ TÀI

Phát hiện và ngưan chặn phần mềm độc hại trên thiết bị di động đóng vai trò quan trọng trong lĩnh vực an ninh mạng. Trong những nưam gần đây, các phương pháp dựa trên học máy (Machine Learning - ML) đã được nghiên cứu rộng rãi và đạt được kết quả khả quan trong việc phát hiện phần mềm độc hại trên hệ điều hành Android. Nhiều nghiên cứu trước đây đã chỉ ra rằng các mô hình ML có thể đạt độ chính xác lên tới 99% trong việc phát hiện phần mềm độc hại.

Tuy nhiên, những nghiên cứu gần đây cũng chỉ ra rằng các thiết kế thực nghiệm phi thực tế mang lại những sai lệch đáng kể, dẫn đến hiệu suất quá lạc quan trong việc phát hiện phần mềm độc hại. Nghĩa là các mô hình ML có thể dựa vào các đặc trưng không liên quan đến hành vi độc hại/an toàn thực sự của ứng dụng để phân loại.

Vì vậy, việc hiểu được cách các mô hình ML hoạt động, lý do tại sao chúng có thể đạt được hiệu suất cao trở nên rất quan trọng. Trong bối cảnh này, các mô hình học máy khả diễn giải có thể cung cấp những phân tích sâu sắc về cách thức hoạt động của các mô hình ML.

Trong đề tài "Explainable ML for Cybersecurity (Android Malware Detection)", nhóm sẽ huấn luyện các bộ phân loại nhằm phát hiện mã độc trên Android và sử dụng SHAP để giải thích các tính nưang được chọn của mô hình nhằm đảm bảo rằng độ chính xác cao của bộ phân loại bắt nguồn từ những điều kiện có thể giải thích được.

2 CƠ SỞ LÝ THUYẾT

2.1 Máy học

Machine Learning là một lĩnh vực nhỏ của Khoa Học Máy Tính, nó có khả nưang tự học hỏi dựa trên dữ liệu đưa vào mà không cần phải được lập trình cụ thể.

Do cần có nguồn dữ liệu cực lớn để "học", máy học vẫn cần có sự tham gia của con người trong việc tìm hiểu dữ liệu cơ sở và lựa chọn các kĩ thuật phù hợp để phân tích thông tin, đánh giá mô hình. Đồng thời, trước khi sử dụng, dữ liệu phải được làm sạch, không có sai lệch và không có dữ liệu giả.

Tất cả các thuật toán máy học đều được xây dựng dựa trên hàm toán học có thể chỉnh sửa. Máy học tập trung vào nguyên tắc rằng tất cả các điểm dữ liệu phức tạp có thể được kết nối về mặt toán học bởi các hệ thống máy tính, miễn là những hệ thống này có đủ dữ liệu và công suất điện toán để xử lý dữ liệu đó.

Do vậy, độ chính xác của kết quả có mối tương quan trực tiếp và phụ thuộc khá lớn vào khối lượng dữ liệu đầu vào.

Các mô hình học máy được chia thành ba loại chính:

- Học có giám sát: được hiểu là cách sử dụng các tập dữ liệu được gắn nhãn để huấn luyện thuật toán phân loại hoặc dự đoán kết quả một cách chính xác.
- Học không giám sát: sử dụng các thuật toán học máy để phân tích và phân cụm các tập dữ liệu không được gắn nhãn. Các thuật toán này phát hiện ra các mẫu hoặc nhóm dữ liệu ẩn mà không cần sự can thiệp của con người.
- Học bán giám sát: là sự kết hợp giữa học tập có giám sát và không giám sát. Trong quá trình đào tạo, nó sử dụng một tập dữ liệu có nhãn nhỏ hơn học có giám sát để hướng dẫn phân loại, trích xuất tính nưang từ một tập dữ liệu lớn hơn, không được gắn nhãn.

2.2 Mô hình học máy khả diễn giải

Mô hình máy học khả diễn giải (Explainable Machine Learning) là một phương pháp tiếp cận trong học máy nhằm tạo ra các mô hình có thể giải thích và hiểu được cách thức hoạt động của chúng. Ngược lại với các mô hình "hộp đen" truyền thống, các mô hình này sử dụng các kỹ thuật để cung cấp thông tin về quá trình ra quyết định và các yếu tố ảnh hưởng đến kết quả dự đoán của mô hình. Điều này giúp con người hiểu rõ hơn về cách mô hình đưa ra dự đoán và tại sao nó chọn một quyết định cụ thể.

Việc sử dụng mô hình máy học khả diễn giải có thể giúp tưang tính minh bạch và đáng tin cậy của các ứng dụng học máy.

Khi một mô hình học máy có thể giải thích quyết định của nó, ta có thể kiểm tra những tiêu chí sau một cách dễ dàng:

- Tính công bằng (Fairness): Đảm bảo rằng dự đoán không bị ảnh hưởng bởi sai lệch (bias) và có tình trạng "phân biệt đối xử" với các nhóm dữ liệu đặc biệt.
- Tính riêng tư (Privacy): Đảm bảo rằng những thông tin nhạy cảm được bảo vệ.
- Tính đáp ứng nhanh (Robustness): Đảm bảo rằng những thay đổi nhỏ ở đầu vào không ảnh hưởng lớn tới kết quả đầu ra.
- Tính nhân quả (Causality): Đảm bảo rằng chỉ những mối quan hệ có tính nhân quả được sử dụng.
- Tính tin cậy (Trust): Đảm bảo mô hình đáng tin cậy hơn mô hình hộp đen.

2.3 SHAP - SHapley Additive exPlanationsSHAP - SHapley Additive exPlanations

SHAP được giới thiệu vào nưam 2017 như một phương pháp không phụ thuộc vào mô hình để giải thích các mô hình học máy. SHAP dựa trên các giá trị Shapley lấy từ lý thuyết trò chơi. Giá trị Shapley được tính bằng cách đo lường tác động của từng người chơi trong trò chơi đồng đội bằng cách tính

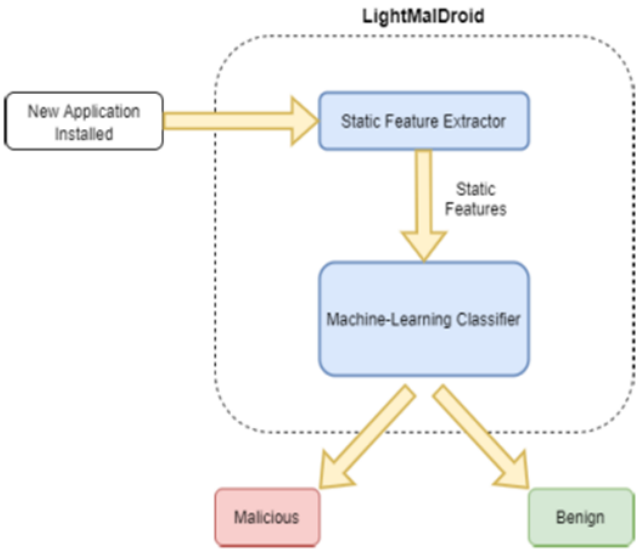
toán hiệu suất của đội khi có và không có người chơi đó. Trong học máy, phương pháp này tính toán tác động của từng đặc trưng bằng cách tính toán sự khác biệt giữa hiệu suất của mô hình khi có và không có đặc trưng đó. Điều này giúp chúng ta hiểu mức độ đóng góp của mỗi đặc trưng vào dự đoán theo hướng tích cực hoặc tiêu cực.

Tầm quan trọng của đặc trưng là kỹ thuật tính điểm cho tất cả các tính n*u*ang đầu vào cho một mô hình nhất định. Điểm số này đại diện cho tầm quan trọng của mỗi đặc trưng. Tầm quan trọng của đặc trưng được tính thông qua Gini importance sử dụng độ nhiễm bẩn của nút và chỉ có thể được tính trong các thuật toán học máy tuyến tính, chẳng hạn như hồi quy tuyến tính (LR), rừng ngẫu nhiên (RF), và cây quyết định (DT). Ngược lại, các giá trị SHAP không phụ thuộc vào mô hình và có thể được sử dụng để giải thích bất kỳ loại phân loại nào, bao gồm cả học sâu. Cách tính toán các giá trị SHAP có thể cung cấp những hiểu biết sâu sắc hơn về tác động của từng đặc trưng riêng lẻ đối với quyết định phân loại.

3 NỘI DUNG BÀI BÁO

3.1 Mô hình PAIRED

Trong bài báo tham khảo, họ đề xuất một mô hình phát hiện mã độc nhẹ dành cho các thiết bị Android. Hệ thống được đề xuất dựa trên việc trích xuất các đặc trưng tĩnh từ các ứng dụng đã cài đặt để đánh giá chúng. Quyết định lựa chọn các đặc trưng tĩnh nhằm mục đích đảm bảo hệ thống có thể đưa ra quyết định nhanh chóng mà không cần đợi ứng dụng có hành vi độc hại, bởi khi đó thiệt hại có thể đã xảy ra. Phân tích động có thể có rủi ro cao trong các ứng dụng độc hại có quyền root thiết bị Android vì thiết bị có thể đã bị root vào thời điểm ứng dụng được phát hiện. Cụ thể, PAIRED được thiết kế như Hình 1



Hình 1: Mô Hình PAIRED

PAIRED được thiết kế dựa trên các nguyên tắc:

1) Độ chính xác cao: Điều này đạt được bằng cách đào tạo đúng cách bộ phân loại học máy với số lượng tính n*u*ang nhỏ hơn nhưng mang lại hiệu quả phân loại cao nhất.

2) Nhẹ: Điều này đạt được bằng cách giảm số lượng tính n*u*ang xuống 84%. Sức mạnh xử lý và yêu cầu bộ nhớ để trích xuất các tính n*u*ang và chạy bộ phân loại sẽ giảm đáng kể trong khi vẫn duy trì độ chính xác. Một đóng góp khác nhằm làm cho hệ thống nhẹ hơn là sử dụng bộ phân loại không tốn nhiều tài nguyên.

3) Hiệu quả cao: Giảm số lượng tính n*u*ang đồng nghĩa với việc thời gian cần thiết để đưa ra quyết định sẽ giảm đáng kể. Quan trọng nhất là thời gian cần thiết để trích xuất đặc trưng cũng giảm đi đáng kể.

4) Tạo một bộ phân loại có thể khái quát hóa: Mô hình phân loại được đào tạo sẽ được kiểm tra bằng cách sử dụng tập hợp con tập dữ liệu chưa được sử dụng trong quá trình đào tạo. Mô hình cũng sẽ được kiểm tra bằng cách sử dụng một tập dữ liệu khác chưa từng được sử dụng trong đào tạo để đảm bảo tính khái quát.

Hệ thống này sẽ được kích hoạt bất cứ khi nào một ứng dụng mới được cài đặt trên thiết bị Android. Mỗi ứng dụng bao gồm một tệp APK, là một tệp nén bao gồm source code, resources, nội dung và manifest file. Mã nguồn được mã hóa dưới dạng tệp Dalvik Executable (DEX) có thể được diễn giải bằng Dalvik Virtual Machine. Tệp manifest bao gồm một số khai báo và thông số kỹ thuật. Các tài nguyên khác có thể chứa hình ảnh và tệp HTML.

Các tệp DEX là một mã thực thi nhị phân được biên dịch và các tính n*u*ang không thể được trích xuất trực tiếp từ chúng. Do đó, chúng phải được giải mã thành các định dạng khác có thể đọc và giải thích được (ví dụ: mã Smali hoặc mã Java). Mã Smali là một dạng trung gian được dịch ngược từ các tệp DEX và về cơ bản là định dạng mã assembly của một ứng dụng. Chỉ sau khi các tệp APK đã được dịch ngược thì các tính n*u*ang mới có thể được trích xuất từ chúng. Trình trích xuất tính n*u*ang tĩnh bắt đầu trích xuất các tính n*u*ang cần thiết và chuyển chúng đến bộ phân loại. Sau đó, trình phân loại sẽ lấy dữ liệu đầu vào và đưa ra dự đoán xem ứng dụng đã cài đặt là độc hại hay vô hại. Một ứng dụng độc hại có thể bị sandboxed hoặc bị xóa ngay lập tức. Trong các phiên bản Android mới hơn, Dalvik đã được thay thế bằng Android Runtime (ART). ART là thời gian chạy được quản lý được các ứng dụng và dịch vụ hệ thống sử dụng để thực thi các thông số kỹ thuật mã byte DEX được đề cập trước đó. Điều này có nghĩa là ART và Dalvik là các compatible runtimes chạy mã byte DEX và các ứng dụng được phát triển cho Dalvik có thể dễ dàng hoạt động trên ART nếu đáp ứng một số điều kiện cụ thể. Trong cả hai hệ thống, các tính n*u*ang được sử dụng trong mô hình đề xuất có thể được trích xuất mà không cần chạy ứng dụng trên thiết bị Android.

3.2 Dataset

Trong bài báo, nhóm tác giả đã sử dụng dataset Drebin-215 để training mô hình. Tập dữ liệu này được tạo bằng cách trích xuất 215 đặc trưng theo phương pháp phân tích tĩnh của các ứng dụng độc hại. Bộ dữ liệu bao gồm 15.036 instances,

mỗi instance đại diện cho một ứng dụng. Tổng cộng có 9476 instances trong bộ dữ liệu này là lành tính. 5560 còn lại là độc hại. 215 đặc điểm đến từ các nguồn trích xuất khác nhau. Các loại đặc trưng trong Drebin-215 được thể hiện trong Hình 2

Feature type	Number of features
API call signature	73
Manifest permission	113
Command signature	06
Intent	23
Total	215

Hình 2: Feature types trong Drebin-215 dataset

Và khi test mô hình, họ đã sử dụng hai tập dữ liệu khác để đảm bảo rằng mô hình được đào tạo có khả n*u*ang khái quát hóa tốt ngoài tập dữ liệu huấn luyện của nó. Tập dữ liệu thử nghiệm đầu tiên là Malgenome-215. Các tính n*u*ang có trong Malgenome-215 tương tự như trong Drebin-215. Tuy nhiên, nó bao gồm 3799 instances, trong đó có 1260 instances độc hại và 2593 instances lành tính.

Tập dữ liệu thử nghiệm thứ hai là CICmalDroid2020. Nó bao gồm nhiều đặc trưng được trích xuất bằng các phương pháp khác nhau, bao gồm cả các đặc trưng đã được chọn trong nghiên cứu này. Bộ dữ liệu bao gồm các tính n*u*ang được trích xuất từ 11.598 instances (1795 instances lành tính và 9803 instances độc hại). Tập dữ liệu bao gồm các loại phần mềm độc hại khác nhau như: adware, banking malware, SMS malware, và riskware. Các đặc trưng được trích xuất đã được xử lý trước để chuẩn bị thử nghiệm bằng cách sử dụng bộ phân loại đã được đào tạo.

3.3 Tính khả diễn giải của mô hình

Một trong những mục tiêu của nghiên cứu này là tạo ra một mô hình có thể giải thích được, điều này sẽ làm t*u*ang sự tin cậy vào giải pháp được đề xuất. Khả n*u*ang giải thích không chỉ tạo dựng niềm tin vào mô hình mà còn đảm bảo rằng độ chính xác đạt được bắt nguồn từ các điều kiện có thể giải thích được chứ không phải từ hoạt động hộp đen. Để phục vụ cho tính khả diễn giải của mô hình, nhóm tác giả sẽ dựa vào SHapley Additive exPlanations (SHAP).

4 THỰC NGHIỆM VÀ ĐÁNH GIÁ

4.1 Công cụ thực hiện

Để hiện thực hóa được mô hình đã đề cập ở trên, chúng em đã sử dụng những công cụ sau:

- Google Colab: Môi trường để xây dựng mô hình.
- Python: Ngôn ngữ lập trình chính được sử dụng để triển khai hệ thống.
- Flask: Flask là một micro web framework cho Python, được sử dụng để xây dựng ứng dụng web cho hệ thống.
- SHAP: được sử dụng để tạo ra các giá trị giải thích, giúp hiểu rõ hơn về các đặc trưng quan trọng ảnh hưởng đến quyết định của mô hình.
- APKTool: được sử dụng để giải mã tệp APK để phân tích các thành phần bên trong của ứng dụng.
- Androguard: Androguard là một bộ công cụ phân tích phần mềm Android, được sử dụng để phân tích và trích xuất các đặc trưng như các quyền, thành phần, và các cuộc gọi API để tạo ra vector đặc trưng cho mô hình máy học từ tệp APK.
- Scikit-learn: được sử dụng để xây dựng và huấn luyện các mô hình máy học (Random Forest, Logistic, Decision, Gaussian Naïve, Support Vector Machine) cũng như để thực hiện quá trình giảm số lượng đặc trưng từ 215 xuống 35.

4.2 Demo

Trước tiên, về phần huấn luận mô hình, bài báo đề xuất chia làm bốn giai đoạn sau:

1) Huấn luyện và thử nghiệm ban đầu: Tập dữ liệu được chia ngẫu nhiên thành làm hai tập dữ liệu con gồm: tập huấn luyện 75% và tập thử nghiệm 25% để thu được kết quả ban đầu bằng cách sử dụng cả n*u*am bộ phân loại.

2) Lựa chọn tính n*u*ang: Bước tiếp theo là chọn số lượng tính n*u*ang thấp hơn so với 215 tính n*u*ang ban đầu. Phương pháp được sử dụng trong việc lựa chọn tính n*u*ang là loại bỏ tính n*u*ang đệ quy (recursive feature elimination - RFE) dựa trên tầm quan trọng của tính n*u*ang. Sau quá trình lựa chọn tính n*u*ang, tập dữ liệu rút gọn thu được sẽ được sử dụng để huấn luyện lại các bộ phân loại và tạo ra một tập kết quả mới.

3) Xác thực chéo 10 lần: Sử dụng xác thực chéo 10 lần trên tập dữ liệu vừa thu được ở giai đoạn 2 như một bước xác thực bổ sung. Trong quy trình xác thực chéo 10 lần, dữ liệu được chia ngẫu nhiên thành 10 tập hợp con. Dữ liệu sau đó đã trải qua 10 chu kỳ huấn luyện và kiểm tra. Trong mỗi chu kỳ, một tập hợp con trong số 10 tập hợp đó được loại khỏi quá trình huấn luyện và được sử dụng cho quá trình thử nghiệm. Điều này được lặp lại 10 lần cho đến khi tất cả 10 tập hợp con được sử dụng để thử nghiệm một lần. Mỗi chu kỳ tạo ra một bộ phân loại với các thông số hiệu suất cụ thể. Nếu các tham số này có phương sai cao thì bộ phân loại sẽ bị quá khớp và không khái quát hóa chính xác trong tập dữ liệu. Nếu phương sai thấp thì giá trị trung bình của các tham số hiệu suất là đáng tin cậy.

4) Kiểm tra tập dữ liệu bổ sung: Để xác thực thêm, các bộ phân loại đã đào tạo được kiểm tra bằng cách sử dụng thêm hai bộ dữ liệu (Malgenome-215 và CICmalDroid2020) để đảm bảo rằng các bộ phân loại có khả n*u*ang khái quát hóa tốt ngoài tập dữ liệu huấn luyện của chúng.

Giai đoạn 1, trước tiên ta chia dữ liệu thành hai phần (code trong Hình 3), sau đó khởi tạo các bộ phân loại cần huấn luyện (code trong Hình 4) và thực hiện huấn luyện các bộ phân loại vừa khởi tạo (code trong Hình 5).

```
# Chia tập dữ liệu thành tập huấn luyện và tập kiểm tra với tỉ lệ 75% và 25%
X_train, X_test, y_train, y_test = train_test_split(df1, y, test_size=0.25, random_state=42)
```

Hình 3: Chia dữ liệu thành hai phần

Kết quả thu được được biển hiện trong hình Hình 6

```python
# Khởi tạo các bộ phân loại
classifiers = {
    'Random Forest': RandomForestClassifier(),
    'Logistic Regression': LogisticRegression(),
    'Decision Tree': DecisionTreeClassifier(),
    'Gaussian Naive Bayes': GaussianNB(),
    'Support Vector Machine': SVC()
}

# Tạo DataFrame để lưu trữ kết quả
results = pd.DataFrame(columns=['Classifier', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
```

Hình 4: Khởi tạo các bộ phân loại

```python
# Huấn luyện và kiểm tra từng bộ phân loại
for name, clf in classifiers.items():
    start_train_time = time.time()
    # Huấn luyện bộ phân loại
    clf.fit(X_train, y_train)
    end_train_time = time.time()
    training_time = end_train_time - start_train_time

    start_test_time = time.time()
    # Dự đoán trên tập kiểm tra
    y_pred = clf.predict(X_test)
    end_test_time = time.time()
    testing_time = end_test_time - start_test_time

    # Tính các số liệu đánh giá
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='binary')  # Sử dụng 'binary' nếu nhãn là nhị phân
    recall = recall_score(y_test, y_pred, average='binary')
    f1 = f1_score(y_test, y_pred, average='binary')

    # In kết quả của bộ phân loại hiện tại
    print(f"Classifier: {name}")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
    print(f"Training Time: {training_time:.4f} seconds")
    print(f"Testing Time: {testing_time:.4f} seconds")
    print("\n" + "-"*50 + "\n")
```

Hình 5: Huấn luyện và in ra kết quả của từng bộ phân loại

Classifier	Accuracy	Precision	Recall	F1 Score	Training Time	Testing Time
Random Forest	0.9867	0.9935	0.9709	0.982	1.1555 s	0.0781 s
Logistic Regression	0.9761	0.9707	0.9652	0.9679	0.3432 s	0.0154 s
Decision Tree	0.9761	0.9674	0.9688	0.9681	0.2286 s	0.0094 s
Gaussian Naive Bayes	0.7166	0.5713	0.9759	0.7207	0.0545 s	0.0199 s
Support Vector Machine	0.9838	0.9849	0.9716	0.9782	2.2990 s	1.3115 s

Hình 6: Kết quả sau khi huấn luyện

Giai đoạn 2, để lựa chọn ra đặc trưng quan trong trong 215 đặc trưng, ta sử dụng thuật toán RFE. Cụ thể, quá trình được thực hiện như sau:

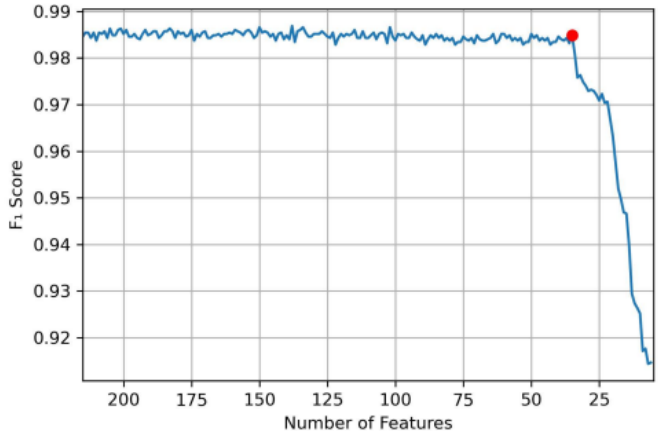
1) Trong mỗi lần lặp, tập dữ liệu sẽ được chia ngẫu nhiên thành 75% tập huấn luyện và 25% tập thử nghiệm. Sau đó, sử dụng bộ phân loại Random Forest để huấn luyện mô hình. Sau quá trình huấn luyện, bộ phân loại được kiểm tra bằng tập kiểm tra.

2) Tầm quan trọng của đặc trưng được tính toán cho tất cả các đặc trưng được sử dụng trong quá trình huấn luyện bộ phân loại. Tầm quan trọng của đặc trưng có thể được đo lường bằng mức giảm tạp chất trung bình được tính toán từ tất cả các decision tree trong forest mà không cần giả định rằng dữ liệu có thể phân tách tuyến tính hay không.

3) Tính nuang có giá trị quan trọng thấp nhất sẽ bị loại bỏ.

4) Sau đó, quá trình này sẽ được lặp lại trong khi giám sát chặt chẽ hiệu suất của hệ thống bằng cách ghi lại F1-score của bộ phân loại. Quá trình được lặp lại cho đến khi đạt đến một ngưỡng nhất định. Ngưỡng này được chọn khi hệ thống chứng kiến hiệu suất giảm mạnh. Vì nhóm tác giả nhận thấy điểm F1 của bộ phân loại giảm nhanh chóng vượt quá ngưỡng 35 đặc trưng, nên ngưỡng này được chọn là 35.

Biểu đồ về tác động của việc giảm các đặc trưng lên F1-Score được biểu hiện trong hình Hình 7.



Hình 7: Tác động của việc giảm tính nuang đến F1-score

Từ thuật toán trên, nhóm đã code phần lựa chọn đặc trưng. Đoạn code này được trình bày trong Hình 8.

```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Tên của các đặc trưng ban đầu
initial_feature_names = list(dataset.columns)

# Khởi tạo mô hình Random Forest
model = RandomForestClassifier()

# Số lượng đặc trưng mục tiêu
target_features = 35

# Danh sách các đặc trưng đã bị xóa
deleted_features = []

# Lặp cho đến khi số lượng đặc trưng của Dataset giảm xuống dưới target_features
while dataset.shape[1] > target_features:

    # Huấn luyện mô hình với tập huấn luyện
    model.fit(X_train, y_train)

    # Tính độ quan trọng của từng đặc trưng
    importance = model.feature_importances_

    # Tìm chỉ số của đặc trưng có độ quan trọng thấp nhất
    i = np.argmin(importance)

    # Xóa đặc trưng đó khỏi Dataset
    dataset = np.delete(dataset, i, axis=1)

    # Xóa tên đặc trưng khỏi danh sách tên
    deleted_features.append(initial_feature_names.pop(i))

    # Cập nhật lại X_train và X_test
    X_train, X_test, y_train, y_test = train_test_split(dataset, y, test_size=0.2, random_state=42)

# In ra số lượng đặc trưng cuối cùng
print("Final number of features:", dataset.shape[1])

# In ra danh sách các đặc trưng đã bị xóa
print("Deleted features:", deleted_features)

# In ra danh sách các đặc trưng còn lại
print("Remaining features:", initial_feature_names)
```

Hình 8: Code chọn 35 đặc trưng từ 215 đặc trưng

Sau đó ta huấn luyện lại mô hình với 35 đặc trưng vừa chọn được phía trên(Hình 9).

Ta thu được kết quả trong Hình 10.

Giai đoạn 3, xác thực chéo 10 lần (Hình 11).

Giai đoạn 4, ta sử dụng bộ phân loại có chỉ số tốt nhất trong các bộ phân loại là Random Forest để test lại với các bộ dữ liệu bổ sung. Với dataset Malgenome-215, ta có kết

```python
# Huấn luyện và kiểm tra từng bộ phân loại
for name, clf in classifiers.items():
    start_train_time = time.time()
    # Huấn luyện bộ phân loại
    clf.fit(X_train_35, y_train_35)
    end_train_time = time.time()
    training_time = end_train_time - start_train_time

    start_test_time = time.time()
    # Dự đoán trên tập kiểm tra
    y_pred_35 = clf.predict(X_test_35)
    end_test_time = time.time()
    testing_time = end_test_time - start_test_time

    # Tính các số liệu đánh giá
    accuracy = accuracy_score(y_test_35, y_pred_35)
    precision = precision_score(y_test_35, y_pred_35, average='binary')
    recall = recall_score(y_test_35, y_pred_35, average='binary')
    f1 = f1_score(y_test_35, y_pred_35, average='binary')

    # In kết quả của bộ phân loại hiện tại
    print(f"Classifier: {name}")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print("Confusion Matrix:")
    print(confusion_matrix(y_test_35, y_pred_35))
    print(f"Training Time: {training_time:.4f} seconds")
    print(f"Testing Time: {testing_time:.4f} seconds")
    print("\n" + "-"*50 + "\n")
```

Hình 9: Huấn luyện lại mô hình với 35 đặc trưng vừa chọn được

Classifier	Accuracy	Precision	Recall	F1 Score	Training Time	Testing Time
Random Forest	0.9806	0.9862	0.9616	0.9738	0.7091 s	0.0532 s
Logistic Regression	0.9548	0.9447	0.9339	0.9393	0.0976 s	0.0007 s
Decision Tree	0.9729	0.9691	0.9581	0.9636	0.0509 s	0.0015 s
Gaussian Naive Bayes	0.884	0.8123	0.8977	0.8529	0.0130 s	0.0022 s
Support Vector Machine	0.9742	0.9802	0.9503	0.965	0.8864 s	0.3555 s

Hình 10: Kết quả huấn luyện lại các bộ phân loại với 35 đặc trưng

quả sau: Accuracy: 0.9884, Precision: 0.9885, Recall: 0.9884, F1-Score: 0.9884, Test Time: 0.0149 seconds (code được thể hiện trong Hình 12). Và với dataset CICmalDroid2020, ta có kết quả sau: Accuracy: 0.9407, Precision: 0.9416, Recall: 0.9407, F1-Score: 0.9407, Test Time: 0.0736 seconds (code được thể hiện trong Hình 13).

Dựa vào mô hình đã được huấn luyện ở trên, nhóm đã xây dựng 1 trang web nơi mà người dùng có thể đưa file apk lên , sau khi phân tích và dự đoán trang web sẽ trả về kết quả file apk đó là độc hại hay bình thường cũng như cung cấp hình ảnh biểu đồ diễn giải mức độ ảnh hưởng của các đặc trưng đối với quyết định của mô hình.

Đoạn code xử lí tải tải tệp apk lên được thể hiện trong Hình 14.

Sau đó, hệ thống sẽ sử dụng APKTool để giải mã filke apk (code được thể hiện trong Hình 15)

Tiếp theo, hệ thống sử dụng Androguard để phân tích file apk và trích xuất các quyền, thành phần, cuộc gọi API, và hành động intent. Các thông tin này sẽ được sử dụng để tạo ra vector đặc trưng cho file apk (code được thể hiện trong Hình 16).

Sau đó, hệ thống sẽ kiểm tra các đặc trưng vừa được trích xuất so với các 35 đặc trưng mà ta đã chọn ra trong giai

```python
from sklearn.model_selection import train_test_split, KFold
# 10-fold cross-validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)
y_np = y.values
results = []

for train_index, test_index in kf.split(dataset):
    X_train_35, X_test_35 = dataset[train_index], dataset[test_index]
    y_train_35, y_test_35 = y_np[train_index], y_np[test_index]

    model.fit(X_train_35, y_train_35)
    y_pred_35_10 = model.predict(X_test_35)

    accuracy = accuracy_score(y_test_35, y_pred_35_10)
    precision = precision_score(y_test_35, y_pred_35_10, average='binary')
    recall = recall_score(y_test_35, y_pred_35_10, average='binary')
    f1 = f1_score(y_test_35, y_pred_35_10, average='binary')

    # Lưu kết quả của fold hiện tại
    results.append({
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1
    })

    # In kết quả của fold hiện tại
    print(f"Fold {len(results)}:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print("\n" + "-"*50 + "\n")
```

Hình 11: Code xác thực chéo 10 lần

```python
model_mal = RandomForestClassifier()
# Huấn luyện mô hình với tập huấn luyện
model_mal.fit(X_train_mal, y_train_mal)

# Bắt đầu đo thời gian dự đoán
start_time = time.time()

# Dự đoán nhãn cho tập kiểm tra
y_pred_mal = model_mal.predict(X_test_mal)

# Kết thúc đo thời gian dự đoán
end_time = time.time()
test_time = end_time - start_time

# Tính toán các chỉ số hiệu suất
accuracy = accuracy_score(y_test_mal, y_pred_mal)
precision = precision_score(y_test_mal, y_pred_mal, average='weighted')
recall = recall_score(y_test_mal, y_pred_mal, average='weighted')
f1 = f1_score(y_test_mal, y_pred_mal, average='weighted')

# In ra các chỉ số hiệu suất và thời gian kiểm tra
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"Test Time: {test_time:.4f} seconds")
```

Hình 12: Test lại bộ phân loại Random Forest với Malgenome-215 dataset

đoạn 2 huấn luyện mô hình (code được thể hiện trong Hình 17).

Tiếp theo, các đặc trưng được đưa vào mô hình máy học đã được huấn luyện để dự đoán xem có phải là phần mềm độc hại hay không. Kết quả dự đoán sẽ là 'Malicious' (phần

```
model = RandomForestClassifier()
# Huấn luyện mô hình với tập huấn luyện
model.fit(X_train_cic, y_train_cic)

# Bắt đầu đo thời gian dự đoán
start_time = time.time()

# Dự đoán nhãn cho tập kiểm tra
y_pred_cic = model.predict(X_test_cic)

# Kết thúc đo thời gian dự đoán
end_time = time.time()
test_time = end_time - start_time

# Tính toán các chỉ số hiệu suất
accuracy = accuracy_score(y_test_cic, y_pred_cic)
precision = precision_score(y_test_cic, y_pred_cic, average='weighted')
recall = recall_score(y_test_cic, y_pred_cic, average='weighted')
f1 = f1_score(y_test_cic, y_pred_cic, average='weighted')

# In ra các chỉ số hiệu suất và thời gian kiểm tra
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"Test Time: {test_time:.4f} seconds")
```

Hình 13: Test lại bộ phân loại Random Forest với CICmal-Droid2020 dataset

```
@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return jsonify({'error': 'Không có tệp'})
    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': 'Không có tệp được chọn'})
    if file:
        file_path = os.path.join('uploads', file.filename)
        file.save(file_path)
```

Hình 14: Code xử lý tải tệp apk lên

```
decompiled_dir = 'decompiled_apk'
if not os.path.exists(decompiled_dir):
    subprocess.run(['apktool', 'd', apk_path, '-o', decompiled_dir])
```

Hình 15: Code giải mã file apk

mềm độc hại) hoặc 'Benign' (không độc hại) (code được thể hiện trong Hình 18).

Tạo biểu đồ SHAP để sử dụng SHAP tạo ra giá trị giải thích cho dự đoán. Biểu đồ sẽ giải thích mức độ ảnh hưởng của các đặc trưng để đưa ra kết qua dự đoán (code được thể hiện trong Hình 19).

Hiển thị kết quả dự đoán và biểu đồ SHAP trên trang web (code được thể hiện trong Hình 20).

Kết quả sau khi up file lên trang web vừa tạo được thể hiện trong Hình 21.

4.3 Phân tích kết quả SHAP

Kết quả SHAP (SHapley Additive exPlanations) trong Hình 21 cho thấy mức độ ảnh hưởng của các đặc trưng khác nhau đến dự đoán của mô hình về việc phần mềm độc hại. Dưới đây là phân tích chi tiết của các kết quả:

```
a, d, dx = AnalyzeAPK(apk_path)

permissions = a.get_permissions()
components = {
    'activities': a.get_activities(),
    'services': a.get_services(),
    'receivers': a.get_receivers(),
    'providers': a.get_providers()
}

api_calls = set()
intent_actions = set()
for method in dx.get_methods():
    m = method.get_method()
    if not hasattr(m, 'get_instructions'):
        continue
    for instruction in m.get_instructions():
        op_value = instruction.get_op_value()
        if 0x6e <= op_value <= 0x72:
            try:
                class_name, method_name, descriptor = instruction.get_operands()[0][2].split('->')
                api_calls.add(f"{class_name}->{method_name}")
            except Exception as e:
                pass
        elif op_value == 0x1a:
            try:
                string_value = instruction.get_operands()[1][2]
                if string_value.startswith('android.intent.action.'):
                    intent_actions.add(string_value)
            except Exception as e:
                pass

feature_vector = [0] * len(drebin_features)
```

Hình 16: Trích xuất các đặc trưng

```
# Kiểm tra quyền
for permission in permissions:
    feature_name = permission.split('.')[-1]
    if feature_name in drebin_features:
        index = drebin_features.index(feature_name)
        feature_vector[index] = 1

# Kiểm tra thành phần
for component_type, component_list in components.items():
    for component in component_list:
        if component in drebin_features:
            index = drebin_features.index(component)
            feature_vector[index] = 1

# Kiểm tra gọi API
for feature in api_calls:
    if feature in drebin_features:
        index = drebin_features.index(feature)
        feature_vector[index] = 1

# Kiểm tra hành động intent
for feature in intent_actions:
    if feature in drebin_features:
        index = drebin_features.index(feature)
        feature_vector[index] = 1
```

Hình 17: Kiểm tra các đặc trưng đã trích xuất so với 35 đặc trưng của model

- Trục hoành (X-axis): Biểu thị giá trị SHAP, đại diện cho mức độ ảnh hưởng của từng đặc trưng đến đầu ra của mô hình. Giá trị SHAP càng lớn, ảnh hưởng của đặc trưng đó đến dự đoán càng lớn. Giá trị SHAP bên phải trục tung (giá trị dương) sẽ biểu thị rằng đặc trưng này góp phần làm tuang dự đoán của mô hình về việc phần mềm là độc hại. Ngược lại, giá trị SHAP bên trái trục tung (giá trị âm) biểu thị rằng đặc trưng

```python
features = extract_features(file_path)

prediction = model.predict([features])
result = 'Malicious' if prediction[0] == 1 else 'Benign'
print(f"Dự đoán: {result}")
```

Hình 18: Dự đoán nhãn

```python
features_np = np.array(features).reshape(1, -1)

explainer = shap.Explainer(model)
shap_values = explainer(features_np)

shap_values_der = shap_values.values
shap_values_for_class = shap_values_der[:, :, 1]
```

Hình 19: Tạo biểu đồ SHAP

```python
shap.summary_plot(shap_values_for_class, features=features_np, feature_names=drebin_features, show=False)

plot_path = os.path.join('static', 'shap_summary.png')
plt.savefig(plot_path)
plt.close()

return render_template('result.html', result=result)
```

Hình 20: Code hiển thị kết quả lên trang web

này góp phần làm giảm dự đoán của mô hình về việc phần mềm là độc hại.

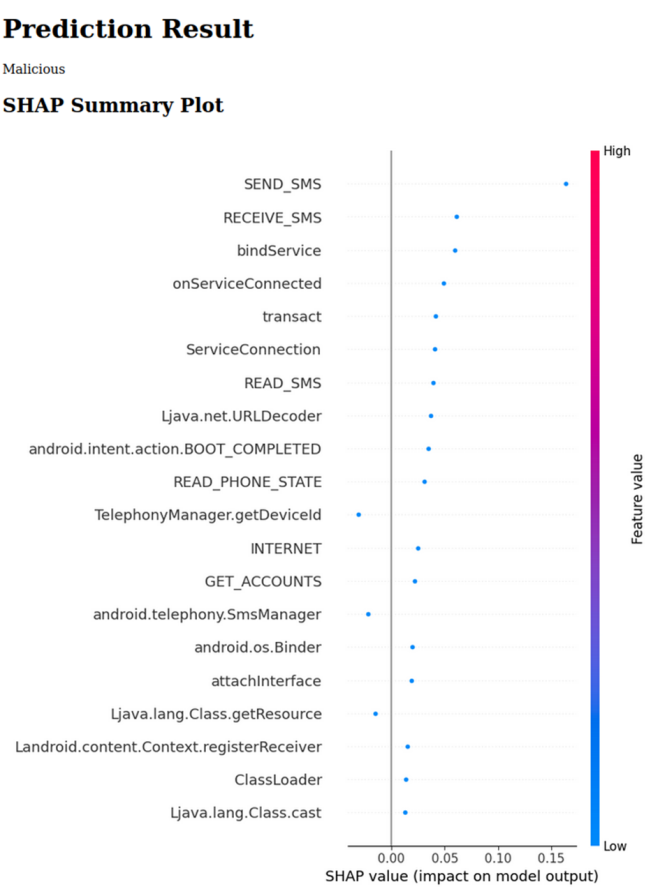
- Python: Trục tung (Y-axis): Liệt kê các tính nuang đã được mô hình sử dụng trong dự đoán.
- Biểu thị giá trị của các đặc trưng. Màu xanh biểu thị giá trị thấp và màu đỏ biểu thị giá trị cao của các đặc trưng. Điều này cho phép bạn thấy không chỉ mức độ ảnh hưởng mà còn cả hướng ảnh hưởng (tích cực hay tiêu cực).

Phân tích cụ thể các đặc trưng ta có thể thấy được:

- SEND_SMS và RECEIVE_SMS: Có ảnh hưởng rất lớn đến dự đoán với giá trị SHAP cao nhất. Hai đặc trưng này này sẽ góp phần thúc đẩy mô hình dự đoán rằng phần mềm là độc hại.
- bindService, onServiceConnected, transact, ServiceConnection, READ_SMS, Ljava.net.URLDecoder, android.intent.action.BOOT_COMPLETED, ClassLoader, READ_PHONE_STATE, INTERNET, attachInterface, GET_ACCOUNTS, android.os.Binder, Landroid.Context.Context.registerReceiver, Ljava.lang.Class.cast: Có ảnh hưởng ít hơn vì giá trị SHAP ít hơn. Các đặc trưng này cũng góp phần thúc đẩy mô hình dự đoán rằng phần mêm độc hại.
- TelephonyManager.getDeviceId, Ljava.lang.Class.getResource, android.telephony.SmsManager: Đây là các đặc trưng có ảnh hưởng khá ít vì giá trị SHAP khá thấp và các tính nuang này góp phần thúc đẩy mô hình dự đoán rằng phần mềm là bình thường.

5 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

PAIRED đã trích xuất 35 tính nuang tĩnh từ các ứng dụng và chuyển chúng đến bộ phân loại học máy đã được huấn luyện



Hình 21: Code hiển thị kết quả lên trang web

để đưa ra dự đoán xem ứng dụng đó là độc hại hay lành tính. Kết quả thử nghiệm cho thấy hệ thống PAIRED được đề xuất hoạt động rất tốt so với các phương pháp hiện đại. Điểm mới trong đóng góp này được tóm tắt bằng việc tạo ra một hệ thống nhẹ với độ chính xác cao, sản xuất phiên bản rút gọn của bộ dữ liệu Drebin-215 cũng như cải thiện hiệu quả và tính tổng quát của hệ thống được đề xuất. Mô hình cũng được giải thích bằng cách sử dụng các giá trị SHAP để tuang độ tin cậy và hiểu rõ hoạt động nội bộ của mô hình phân loại được đề xuất.

Công việc tiếp theo nhóm em có thể sẽ thực hiện là cải thiện độ chính xác bằng cách sử dụng các bộ dữ liệu và bộ phân loại bổ sung.

Hướng nghiên cứu trong tương lai của nhóm tác giả là hướng tới việc tạo ra chức nuang cập nhật mô hình học máy dựa trên đám mây mạnh mẽ. Một hướng khác sẽ là nghiên cứu sâu hơn về việc giảm bộ nhớ và yêu cầu xử lý của bộ phân loại.