

BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



BÁO CÁO DỰ ÁN
NHẬP MÔN KHOA HỌC DỮ LIỆU

*Đề tài: “ NHẬN DIỆN MALWARE TRONG FILE APK
SỬ DỤNG DOC2VEC VÀ GCN”*

NHẬP MÔN KHOA HỌC DỮ LIỆU

NHÓM 2 - LỚP N05

Nhóm 02:	NO5
Đào Duy Thông	: B21DCCN696
Vũ Thành Tuyên	: B21DCCN780
Nguyễn Việt Khiêm	: B21DCCN458
Đỗ Đăng Khoa	: B21DCCN068
Hoàng Hữu Đức	: B21DCCN240
Trần Sỹ Tiến	: B21DCCN709

HÀ NỘI - 2024

BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



BÁO CÁO DỰ ÁN
NHẬP MÔN KHOA HỌC DỮ LIỆU

*Đề tài: “ NHẬN DIỆN MALWARE TRONG FILE APK
SỬ DỤNG DOC2VEC VÀ GCN”*

Nhóm 02:	NO5
Đào Duy Thông	: B21DCCN696
Vũ Thành Tuyên	: B21DCCN780
Nguyễn Việt Khiêm	: B21DCCN458
Đỗ Đăng Khoa	: B21DCCN068
Hoàng Hữu Đức	: B21DCCN240
Trần Sỹ Tiến	: B21DCCN709

LỜI CẢM ƠN

Đầu tiên, nhóm chúng em xin gửi lời cảm ơn chân thành đến Học viện Công nghệ Bưu chính Viễn thông đã đưa môn học Nhập môn Khoa học dữ liệu vào chương trình giảng dạy. Đặc biệt, chúng em xin gửi lời cảm ơn sâu sắc đến giảng viên bộ môn - Thầy Ths. Đinh Xuân Trường đã dạy dỗ, truyền đạt những kiến thức quý báu cho chúng em trong suốt thời gian học tập vừa qua. Trong thời gian tham gia lớp học của thầy, chúng em đã có thêm nhiều kiến thức bổ ích, tinh thần học tập hiệu quả, nghiêm túc. Đây chắc chắn sẽ là hành trang quý giá giúp chúng em vững bước sau này.

Môn học Nhập môn Khoa học dữ liệu là môn học thú vị, vô cùng bổ ích và có tính thực tế cao. Môn học cung cấp kiến thức thiết thực, phù hợp với nhu cầu của sinh viên. Tuy nhiên, do vốn kiến thức còn hạn chế và khả năng tiếp thu thực tế còn chưa hoàn thiện, dù chúng em đã cố gắng hết sức, nhưng bài tập lớn khó tránh khỏi những thiếu sót. Kính mong thầy xem xét và góp ý để bài tập lớn của nhóm chúng em được hoàn thiện hơn.

Nhóm chúng em xin chân thành cảm ơn!

Hà Nội, ngày 26 tháng 9 năm 2024

Đại diện nhóm

Đào Duy Thông

MỤC LỤC

LỜI CẢM ƠN.....	i
MỤC LỤC.....	ii
DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT	iii
DANH MỤC CÁC HÌNH VẼ.....	v
DANH MỤC CÁC BẢNG	viii
PHẦN MỞ ĐẦU.....	ix
CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI.....	1
1.1 Tổng quan đề tài	1
1.2 Mục tiêu và định hướng giải pháp	1
1.3 Đóng góp của đề tài	2
1.4 Giới thiệu về bài toán nhận diện malware	2
1.5 Tổng quan giải pháp	2
CHƯƠNG 2. BÁO CÁO TIẾN ĐỘ TÙNG TUẦN Topic 1-12	4
2.1 Tổng quan quá trình làm dự án trong 12 tuần	4
CHƯƠNG 3. BÁO CÁO DỰ ÁN	61
3.1 Thu thập dữ liệu	61
3.1.1 Nguồn dữ liệu.....	61
3.1.2 Tiền xử lý dữ liệu	61
3.1.3 Tải dữ liệu từ Androzoo	63
3.2 Thống kê và Trực quan hóa Dữ liệu	64
3.2.1 Biểu đồ	64
3.2.2 Biểu Đồ Histogram.....	65
3.2.3 Biểu Đồ Boxplot.....	66
3.2.4 Biểu Đồ KDE	67
3.2.5 Biểu Đồ Tần Suất.....	68

DANH MỤC THUẬT NGỮ VÀ TỪ VIẾT TẮT

Thuật ngữ	Ý nghĩa
Androguard	Công cụ dịch ngược và phân tích ứng dụng Android
API: Application Programming Interface	Giao diện lập trình ứng dụng
APK: Android Package	Gói cài đặt ứng dụng Android
Benign	Phần mềm hợp pháp, không gây hại
Doc2Vec: Document to Vector	Phương pháp chuyển đổi văn bản thành vector
FCG: Function Call Graph	Đồ thị gọi hàm
GCN: Graph Convolutional Network	Mạng nơ-ron tích chập trên đồ thị
Malware: Malicious Software	Phần mềm độc hại

DANH MỤC HÌNH VẼ

1.1	Tổng quan kiến trúc dự án	3
2.1	Email xin API Key	5
2.2	Email phản hồi của Androzoo	5
2.3	Log của quá trình tải file APK xuống	8
2.4	Dataset gồm 1000 file APK benign	8
2.5	File APK khi được mở bằng notepad (chứa các kí tự unicode sau khi đóng gói, đặt tên file bằng sha256)	10
2.6	File APK khi được đổi phương thức đọc thành .zip và giải nén	10
2.7	Dataframe sau khi xử lí	12
2.8	Dataframe	13
2.9	Benign histogram	14
2.10	Malware histogram	14
2.11	Benign boxplot	15
2.12	Malware boxplot	15
2.13	Benign kde	16
2.14	Malware kde	16
2.15	Benign frequency	17
2.16	Malware frequency	17
2.17	Architecture Doc2vec	19
2.18	DM and DBOW	19
2.19	Embedding java with doc2vec	20
2.20	CNN	22
2.21	Minh họa về GCN	23
2.22	So sánh CNN với GCN	23
2.23	Input và output của GCN	24
2.24	GCN cho phân loại đồ thị	24
2.25	Widget CSV File Import	26
2.26	Widget Select Columns	26
2.27	Widget Select Columns	27
2.28	Widget Data Sampler	27
2.29	Widget Test and Score	28
2.30	Widget Predictions	28
2.31	Kết nối các widget trong Orange Data Mining	29
2.32	Lựa chọn các thuộc tính để huấn luyện mô hình	30
2.33	Caption	30
2.34	Kết nối Data Sampler với Test and Score	31
2.35	Kết nối các mô hình với Test and Score	31

2.36	Lựa chọn Test on test data	31
2.37	Lựa chọn Test on test data	31
2.38	Kết nối dữ liệu vào Predictions	32
2.39	Kết nối mô hình với Predictions	32
2.40	Kết nối output của Predictions với Data Table để xem dữ liệu dự đoán	32
2.41	Kết quả đánh giá mô hình	33
2.42	Kết quả dự đoán	33
2.43	Mô hình tổng quan	34
2.44	Kết quả của PCA	35
2.45	Ví dụ về Hierarchical clustering	36
2.46	Cấp bậc trong dữ liệu	37
2.47	Hierarchical clustering trong Orange data mining	37
2.48	Kết quả của Hierarchical clustering	37
2.49	Hierarchical clustering trên scatterplot	38
2.50	K-means	38
2.51	Kết quả của K-mean trên dữ liệu apk	39
2.52	NN - 10	40
2.53	NN - 10, 10	40
2.54	NN - 10, 20, 10	41
2.55	NN - 10, 20, 20, 20, 20, 20, 10	41
2.56	Sơ đồ minh họa cho thí nghiệm số lớp khác nhau	42
2.57	Kết quả thí nghiệm số lớp khác nhau	42
2.58	NN với số unit 1	43
2.59	NN với số unit 10	43
2.60	NN với số unit 100	44
2.61	NN với số unit 1000	44
2.62	Sơ đồ minh họa cho thí nghiệm số lớp khác nhau	45
2.63	Kết quả thí nghiệm số unit khác nhau	45
2.64	Kết quả thí nghiệm	46
2.65	Architecture Doc2vec	47
2.66	DM and DBOW	48
2.67	Embedding java with doc2vec	49
2.68	Log Embedding java qua doc2vec	51
2.69	Kết quả Embedding java qua doc2vec	52
2.70	noSQL database	53
2.71	Bên trong một Collection	54
2.72	Collections	55
3.1	Email phản hồi của Androzoo	61
3.2	Benign histogram	65
3.3	Malware histogram	65

3.4 Benign boxplot	66
3.5 Malware boxplot	66
3.6 Benign kde	67
3.7 Malware kde	67
3.8 Benign frequency	68
3.9 Malware frequency	68
3.10 Architecture Doc2vec	73
3.11 DM and DBOW	73
3.12 Embedding java with doc2vec	75
3.13 Log Embedding java qua doc2vec	78
3.14 Kết quả Embedding java qua doc2vec	78
3.15 CNN vs GCN	79
3.16 Function call graph	79
3.17 Doc2Vec	80
3.18 Tổng quan mô hình	80
3.19 Áp dụng cho 1771 file	84

DANH MỤC BẢNG BIỂU

PHẦN MỞ ĐẦU

Hiện nay, cuộc cách mạng công nghiệp 4.0 đang mang lại những bước tiến vượt bậc trong nhiều lĩnh vực, đặc biệt là trong ngành công nghệ thông tin. Công nghệ trí tuệ nhân tạo và học máy ngày càng đóng vai trò quan trọng trong việc xử lý và phân tích dữ liệu lớn. Một trong những thách thức lớn của thế giới số là vấn đề an ninh mạng, với số lượng lớn ứng dụng phần mềm độc hại (malware) ngày càng gia tăng. Phân tích và nhận diện malware, đặc biệt là trong các file APK của hệ điều hành Android, trở thành một nhu cầu cấp thiết để bảo vệ hệ thống và người dùng.

Hiện tại, có nhiều phương pháp truyền thống để phát hiện malware, bao gồm phân tích tĩnh và động. Tuy nhiên, các phương pháp này gặp nhiều hạn chế khi phải xử lý khối lượng lớn dữ liệu, hoặc đối phó với các loại malware tinh vi hơn như polymorphic malware. Hơn nữa, việc chỉ dựa vào chữ ký hoặc các mẫu đã biết có thể bỏ sót những mối đe dọa mới chưa được phát hiện. Điều này đặt ra nhu cầu về các phương pháp phân tích mới hiệu quả hơn trong việc nhận diện malware tiềm ẩn.

Nhằm giải quyết những hạn chế đó, đề tài này nghiên cứu và phát triển một hệ thống nhận diện malware trong file APK dựa trên việc kết hợp hai phương pháp: Doc2Vec để chuyển đổi các đặc trưng của mã nguồn thành biểu diễn số, và Graph Convolutional Networks (GCN) để phân tích mối quan hệ giữa các thành phần trong ứng dụng. Mục tiêu của đồ án là xây dựng một mô hình có khả năng nhận diện chính xác các ứng dụng độc hại, cải thiện hiệu quả phát hiện so với các phương pháp truyền thống.

Với mục tiêu trên, đề tài có tên “**Nhận diện malware trong file APK sử dụng Doc2Vec và GCN**” đóng góp vào việc cải thiện hệ thống phát hiện phần mềm độc hại bằng cách khai thác các kỹ thuật học sâu tiên tiến. Đồ án sẽ bao gồm các phần chính: giới thiệu vấn đề, khảo sát các phương pháp hiện tại, đề xuất mô hình kết hợp Doc2Vec và GCN, thực nghiệm và đánh giá kết quả, kết luận và hướng phát triển tiếp theo. Nội dung trình bày trong báo cáo gồm 3 chương chính:

- **Chương 1: Giới thiệu chung** Trình bày tổng quan về đề tài nhận diện malware trong file APK, nêu rõ mục tiêu nghiên cứu, đối tượng phân tích và phương hướng giải quyết. Ngoài ra, phần này cũng sẽ giới thiệu các công nghệ liên quan, bao gồm kỹ thuật học sâu (Doc2Vec và GCN), đồng thời xác định các bước cần thiết để đạt được mục tiêu.
- **Chương 2: Báo cáo tiến độ từng tuần trong quá trình thực hiện dự án** Chương này sẽ trình bày chi tiết tiến độ làm việc hàng tuần, bao gồm các hoạt động nghiên cứu, thu thập dữ liệu, cài đặt, thử nghiệm và đánh giá các phương pháp nhận diện malware trong file APK bằng cách sử dụng Doc2Vec và GCN.
- **Chương 3: Kết luận quá trình thực hiện dự án** Bao gồm bài học kinh nghiệm và những kết quả đạt được trong suốt quá trình thực tập. Từ đó, rút ra các điểm mạnh cũng như những điều cần cải thiện và hướng phát triển tiếp theo trong tương lai, đặc biệt trong việc nâng cao độ chính xác của mô hình nhận diện malware.

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

Chương 1 giới thiệu bài toán nhận diện phần mềm độc hại (malware) và khảo sát các phương pháp liên quan. Đặc biệt, chương này tập trung vào việc ứng dụng các công cụ và kỹ thuật như Androguard, Doc2Vec, và Graph Convolutional Networks (GCN) để phân tích và nhận diện mã độc. Androguard được sử dụng để dịch ngược file APK và tạo đồ thị Function Call Graph (FCG), cung cấp thông tin về cấu trúc và hành vi của mã độc. Bằng cách kết hợp việc trích xuất đặc trưng từ Doc2Vec và phân tích quan hệ qua GCN, nghiên cứu đề xuất một giải pháp nhận diện malware hiệu quả hơn so với các phương pháp truyền thống.

1.1 Tổng quan đề tài

Khảo sát hiện trạng

Nhận diện phần mềm độc hại là một lĩnh vực quan trọng trong bảo mật thông tin. Với sự gia tăng không ngừng của các mã độc, các phương pháp truyền thống sử dụng đặc trưng tĩnh (static features) và động (dynamic features) trở nên kém hiệu quả khi đối mặt với các biến thể mã độc mới. Trong đề tài này, chúng tôi đề xuất sử dụng Androguard để dịch ngược các file APK và tạo đồ thị Function Call Graph (FCG) nhằm phân tích sâu hơn về hành vi của mã độc.

Androguard là một công cụ mạnh mẽ trong phân tích mã Android, cho phép trích xuất các thông tin quan trọng từ file APK và dịch ngược mã bytecode về dạng dễ phân tích hơn. Doc2Vec được sử dụng để chuyển các đoạn mã thành các vector biểu diễn, trong khi GCN được áp dụng để phân tích cấu trúc đồ thị FCG, qua đó nhận diện các mẫu mã độc thông qua các mối quan hệ giữa các hàm gọi trong mã.

Các giải pháp hiện tại và hạn chế

Hiện nay, các phương pháp nhận diện malware truyền thống dựa vào việc phân tích tĩnh và động đã bộc lộ nhiều hạn chế trong việc xử lý các mẫu malware phức tạp và thay đổi liên tục. Các phương pháp hiện đại hơn, như việc dịch ngược mã và phân tích đồ thị chức năng (FCG), kết hợp với học sâu, đặc biệt là Doc2Vec và GCN, có khả năng phân tích sâu hơn về mối quan hệ giữa các thành phần mã độc.

Tuy nhiên, việc sử dụng Androguard để phân tích mã ngược đòi hỏi thời gian và tài nguyên tính toán lớn. Đồng thời, việc triển khai GCN cũng đòi hỏi xử lý đồ thị phức tạp, có thể dẫn đến thời gian huấn luyện lâu và độ khó trong việc giải thích kết quả mô hình.

1.2 Mục tiêu và định hướng giải pháp

Mục tiêu của đề tài là xây dựng một hệ thống nhận diện phần mềm độc hại sử dụng Androguard để dịch ngược file APK và tạo đồ thị FCG. Sau đó, sử dụng Doc2Vec để chuyển mã thành các vector biểu diễn và GCN để phân tích các mối quan hệ giữa các hàm gọi trong mã, từ đó nhận diện chính xác các mẫu mã độc mới. Cụ thể, các bước thực hiện bao gồm:

Dịch ngược và phân tích mã bằng Androguard: Trích xuất các hàm và tạo đồ thị FCG từ các file APK để phân tích cấu trúc của mã độc. Trích xuất đặc trưng từ mã bằng Doc2Vec: Chuyển các đoạn mã thành vector để biểu diễn đặc trưng của mã độc. Xây dựng và phân tích đồ thị FCG bằng GCN: Sử dụng GCN để phát hiện các mối quan hệ giữa các hàm và nâng cao hiệu quả nhận diện malware.

1.3 Đóng góp của đề tài

Đề tài này có ba đóng góp chính như sau:

Đưa ra một khảo sát toàn diện về các công cụ và phương pháp hiện đại như Androguard, Doc2Vec, và GCN trong việc nhận diện phần mềm độc hại. Đề xuất một giải pháp kết hợp giữa dịch ngược mã và phân tích đồ thị FCG để nâng cao hiệu quả phát hiện malware. Thực hiện các thử nghiệm đánh giá mô hình trên các tập dữ liệu mã độc phổ biến, cung cấp phân tích chi tiết về hiệu suất của hệ thống nhận diện đề xuất.

1.4 Giới thiệu về bài toán nhận diện malware

Phần mềm độc hại (malware) đang trở thành mối đe dọa lớn đối với hệ thống an ninh mạng. Các công cụ như Androguard cho phép dịch ngược mã và phân tích các mối quan hệ trong mã thông qua đồ thị chức năng (FCG). Bài toán đặt ra là làm thế nào để kết hợp phân tích mã và học sâu nhằm phát hiện chính xác các mẫu malware mới. Đề tài này cung cấp một phương pháp tiếp cận mới dựa trên việc phân tích đồ thị hàm và sử dụng Doc2Vec và GCN để trích xuất và học các đặc trưng từ mã độc.

1.5 Tổng quan giải pháp

Đề tài tập trung vào việc xây dựng một giải pháp nhận diện malware dựa trên luồng xử lý như sau:

1. Dịch ngược file APK và tạo đồ thị FCG (Function Call Graph):

- Sử dụng **Androguard** để dịch ngược file APK thành mã nguồn Java.
- Tạo đồ thị FCG thể hiện mối quan hệ gọi hàm (function call) giữa các thành phần trong mã nguồn. Các nút của đồ thị đại diện cho các hàm, và các cạnh biểu thị mối quan hệ gọi hàm giữa chúng.

2. Học đặc trưng từ mã nguồn bằng Doc2Vec:

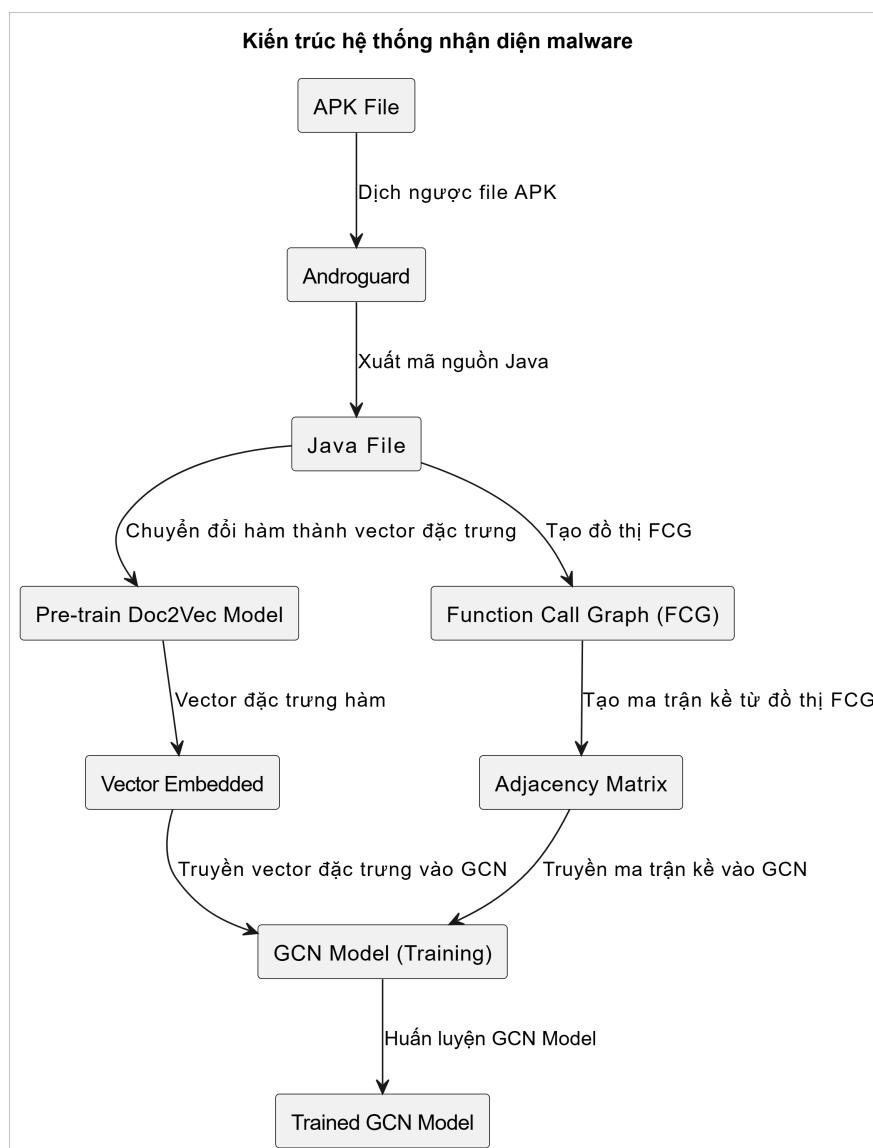
- Sử dụng mã Java đã được trích xuất từ file APK làm đầu vào cho **Doc2Vec** để chuyển đổi các đoạn mã hoặc hàm thành các vector biểu diễn (embeddings).
- Các vector này phản ánh ngữ nghĩa và cấu trúc của từng hàm, cho phép mô hình học được các đặc điểm nổi bật của các đoạn mã độc.

3. Tạo ma trận kề từ đồ thị FCG:

- Từ đồ thị FCG, tạo ma trận kề (adjacency matrix) để biểu diễn các mối quan hệ giữa các hàm dưới dạng dữ liệu đầu vào cho mô hình **Graph Convolutional Network (GCN)**.
- Ma trận kề kết hợp với các vector đặc trưng từ Doc2Vec để cung cấp thông tin đầy đủ về cấu trúc đồ thị và nội dung mã.

4. Phân tích đặc trưng đồ thị và huấn luyện mô hình dự đoán bằng GCN:

- Đưa các vector biểu diễn (node features) và ma trận kề của đồ thị FCG vào **GCN** để học các đặc trưng toàn cục của đồ thị.
- GCN sẽ tận dụng các mối quan hệ trong đồ thị để xác định các mẫu mã độc dựa trên cấu trúc và hành vi của các hàm.
- Huấn luyện mô hình dự đoán với tập dữ liệu mã độc và mã lành tính, từ đó xây dựng hệ thống nhận diện chính xác các mẫu mã độc mới.



Hình 1.1: Tổng quan kiến trúc dự án

Kết quả mong đợi

- Hệ thống có khả năng nhận diện mã độc chính xác hơn so với các phương pháp truyền thống nhờ tận dụng đồng thời thông tin về ngữ nghĩa mã (**Doc2Vec**) và mối quan hệ cấu trúc trong đồ thị (**GCN**).
- Cung cấp một phương pháp hiệu quả trong việc phát hiện các mẫu malware mới, đáp ứng yêu cầu trong bối cảnh mã độc ngày càng phức tạp.

CHƯƠNG 2. BÁO CÁO TIẾN ĐỘ TÙNG TUẦN Topic 1-12

Chương này tổng hợp và trình bày chi tiết tiến độ thực hiện của đề tài qua các tuần từ 1 (20/09/2024). Nội dung bao gồm các công việc đã thực hiện, kết quả đạt được, khó khăn gặp phải và cách giải quyết cho từng giai đoạn. Mỗi tuần sẽ được mô tả cụ thể với những hoạt động nghiên cứu, triển khai và phân tích liên quan đến đề tài. Bên cạnh đó, chương còn đánh giá sự tiến triển của dự án theo từng thời điểm và đưa ra kế hoạch điều chỉnh nếu cần thiết để đảm bảo tiến độ và chất lượng của đề tài nghiên cứu..

2.1 Tổng quan quá trình làm dự án trong 12 tuần

Tuần Topic 1 (20/9 - 27/9)

Chủ đề tìm hiểu tuần 1: Tìm hiểu nguồn dữ liệu và cách tổ chức dữ liệu của AndroZoo. Tìm hiểu API, cách truy cập và tải dữ liệu. Code Python để tải được dữ liệu về và tạo dataset trên Kaggle để sử dụng cho các bước tiếp theo.

Thành viên tham gia

Đào Duy Thông - 50%

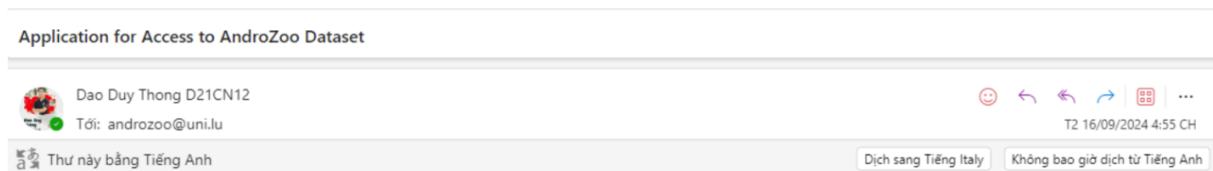
Vũ Thành Tuyên - 50%

Mục tiêu tuần Topic 1:

1. Tìm hiểu về nguồn dữ liệu và cách tổ chức dữ liệu của Androzoo
2. Tìm hiểu cách truy cập dữ liệu của Androzoo
3. Tiền xử lý file CSV
4. Thực hiện cào dữ liệu từ Androzoo.
5. Tạo dataset bao gồm danh sách benign và malware từ dữ liệu đã cào.

Kết quả của tuần Topic:

1. Thực hiện nội dung 1: Tìm hiểu về nguồn dữ liệu và cách tổ chức dữ liệu của Androzoo
 - **Tìm hiểu về AndroZoo :** AndroZoo là một bộ sưu tập ứng dụng Android đang phát triển, được thu thập từ nhiều nguồn khác nhau, bao gồm cả cửa hàng ứng dụng chính thức Google Play. Hiện tại, nó chứa 24,718,475 tệp APK khác nhau, mỗi tệp đã được (hoặc sẽ sớm được) phân tích bởi nhiều sản phẩm phần mềm diệt virus để xác định ứng dụng nào bị phát hiện là phần mềm độc hại.
 - **Cách AndroZoo tổ chức dữ liệu :** AndroZoo cung cấp API để người dùng có thể truy cập : [https://androzoo.uni.lu/api/download?apikey=\\\${APIKEY}\&sha256=\\\${SHA256}](https://androzoo.uni.lu/api/download?apikey=\${APIKEY}\&sha256=\${SHA256}). Trong đó APIKEY là API Key của người dùng được cấp thông qua email, SHA256 là mã hash, mã này được lưu trữ trong 1 file CSV nặng 2,7GB, mỗi hàng chứa thông tin của 1 file APK như : sha256 : mã hash để tải file APK, apksize : kích thước file APK, vt-tool-detection : số phần mềm virus phát hiện file APK là độc hại,...
2. Thực hiện nội dung 2: Tìm hiểu cách truy cập dữ liệu của Androzoo
 - **Các điều kiện sử dụng dữ liệu từ AndroZoo.** Androzoo yêu cầu người dùng gửi email từ tài khoản edu để lấy được API key. Nhóm đã gửi email và được phản hồi như dưới đây :



I hope this email finds you well. My name is Dao Duy Thong, and I am a student at Posts and Telecommunications Institute of Technology (PTIT). I am writing to formally request access to the AndroZoo dataset for research purposes.

I have thoroughly reviewed the terms and conditions outlined by AndroZoo and hereby confirm the following:

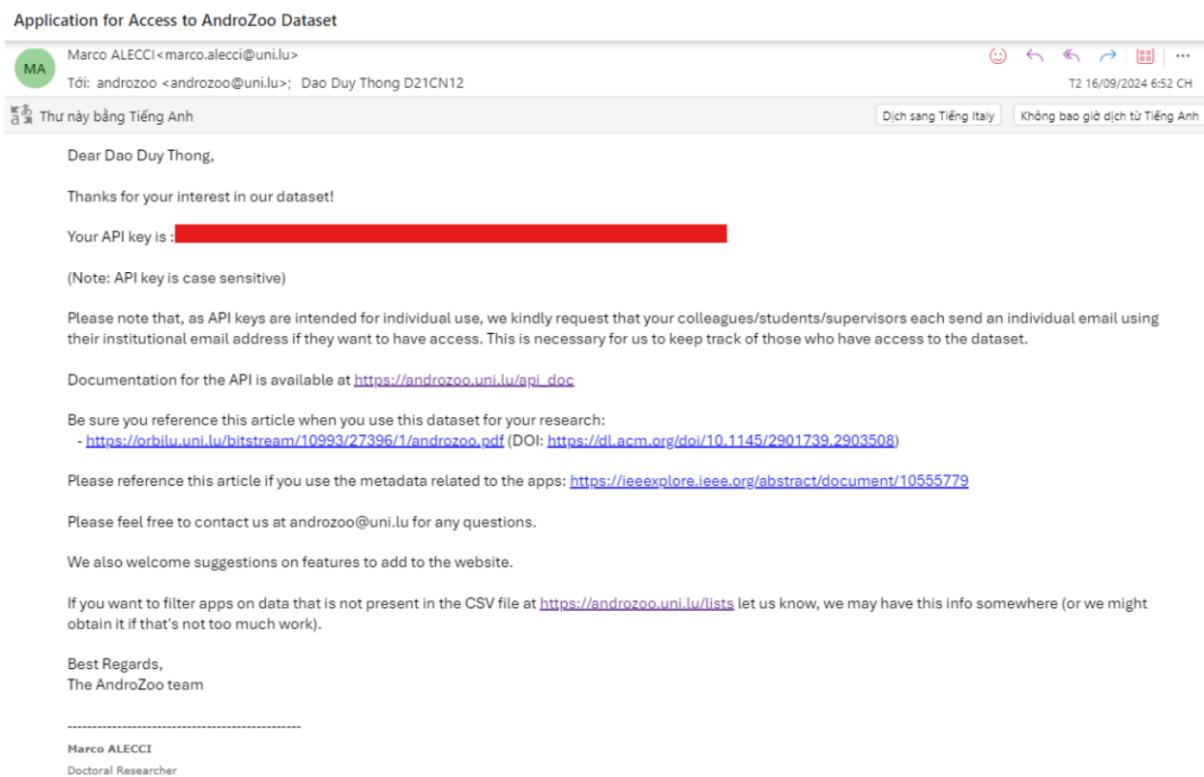
- I have evaluated the legal situation of downloading and working on copyrighted applications with regard to the laws of Vietnam and the policies of Posts and Telecommunications Institute of Technology.
- I will not redistribute the data without your explicit consent.
- I will not use the data to create an application marketplace.
- I will not use the data for any commercial purposes.
- The API key provided to me will be strictly personal, and I will not distribute it or make it publicly available.

I would greatly appreciate your consideration of this application. If further information or documentation is required, please do not hesitate to contact me.

Thank you for your time and support.

Best regards,
Dao Duy Thong
Student
Posts and Telecommunications Institute of Technology

Hình 2.1: Email xin API Key



Hình 2.2: Email phản hồi của Androzoo

Sau khi email xin cấp API KEY, nhóm đã được phản hồi và cấp 1 API KEY để truy cập và tải xuống các file APK của AndroZoo.

3. Thực hiện nội dung 3: Tiền xử lí file CSV

Vì file csv có kích thước rất lớn, chứa rất nhiều mã hash để tải các file APK, cần lọc hoặc cắt chỉ lấy đủ số lượng file APK cần thiết cho quá trình làm dự án (khoảng 2000 file). Cụ thể :

- Bước 1 : Đọc file CSV đã được giải nén trước đó
- Bước 2: Lọc bỏ các dòng có chứa 'snaggamea' trong cột 'pkg-name'
- Bước 3: Chuyển đổi cột 'dex-date' sang định dạng datetime
- Bước 4: Lọc các dòng có 'dex-date' trong khoảng từ 01/06/2023 đến 01/06/2024
- Bước 5: Lọc các dòng có apk-size nhỏ hơn 30 MB ($30 * 1024 * 1024 = 31457280$ bytes)
- Bước 6: Tách danh sách benign và malware dựa vào cột 'vt-detection' và lấy 1000 file đầu tiên.
- Bước 7: Lưu danh sách benign và malware vào các file CSV riêng

Code xử lý file CSV

```

1 import pandas as pd
2 df = pd.read_csv('/kaggle/input/androzoolist/latest.csv')
3 df_filtered = df[~df['pkg_name'].str.contains('snaggamea', na=False)]
4 df_filtered['dex_date'] = pd.to_datetime(df_filtered['dex_date'], errors='coerce')
5 df_filtered_dex_date_range = df_filtered[
6     (df_filtered['dex_date'] >= '2023-06-01') &
7     (df_filtered['dex_date'] <= '2024-06-01')
8 ]
9 df_filtered_dex_date_range_under30 = df_filtered_dex_date_range[
10    df_filtered_dex_date_range['apk_size'] < 31457280]
11 df_filtered_dex_date_range_under30.to_csv(filtered_file_path, index=False)
12 benign_list = df_filtered_dex_date_range_under30[
13     df_filtered_dex_date_range_under30['vt_detection'] == 0].head(1000)
14 malware_list = df_filtered_dex_date_range_under30[
15     df_filtered_dex_date_range_under30['vt_detection'] > 0].head(1000)
16 benign_file_path = '/kaggle/working/benign_list_1500.csv'
17 malware_file_path = '/kaggle/working/malware_list_1500.csv'
18
19 benign_list.to_csv(benign_file_path, index=False)
20 malware_list.to_csv(malware_file_path, index=False)
21
22 # Print the paths to confirm successful save
23 print("Benign list saved at:", benign_file_path)
24 print("Malware list saved at:", malware_file_path)

```

Sau bước này sẽ có 1 dataset gồm 2 file csv : benignlist.csv và malwarelist.csv, mỗi file có khoảng 1000 dòng. Tổng dung lượng apk của benign là 16,7 GB, của malware là 17,34 GB

4. Thực hiện bước 4: Cào dữ liệu từ AndroZoo.

Sử dụng API key và mã hash với cú pháp:

[https://androzoo.uni.lu/api/download?apikey=\\$APIKEY&sha256=\\$SHA256](https://androzoo.uni.lu/api/download?apikey=$APIKEY&sha256=$SHA256)

Duyệt qua các hàng trong file CSV, lấy mã sha256 và dùng request trong Python để tải file APK về.

Do tổng dung lượng APK của benign và malware lớn hơn 19,5GB (dung lượng tối đa cho output), cần thực hiện tải xuống hai lần.

```
1   from concurrent.futures import ThreadPoolExecutor
2
3   import pandas as pd
4
5   import os
6
7   import requests
8
9
10  def multithread(func, ls):
11
12      with ThreadPoolExecutor(10) as p:
13          ans = p.map(func, ls)
14
15      return ans
16
17
18  def download(sha256_hash):
19
20      sha256_hash, cnt = sha256_hash.split("#")
21
22
23      pathsave = "/kaggle/working/"
24
25
26      base_url = 'https://androzoo.uni.lu/api/download?apikey=6205
27
28      e414a89f470504a13bc12dfeecb11db866d927248f748213410cb168b6d2&sha256
29
30      ='
31
32
33      download_url = base_url + sha256_hash
34
35      response = requests.get(download_url, stream=True)
36
37
38      if response.status_code == 200:
39
40          file_name = pathsave + f"{sha256_hash}.apk"
41
42          with open(file_name, 'wb') as file:
43
44              for chunk in response.iter_content(chunk_size=8192):
45
46                  file.write(chunk)
47
48          print(f"File {cnt} - {file_name} ")
49
50      else:
51
52          print(f"Error {sha256_hash}: {response.status_code}")
53
54  data = pd.read_csv("/kaggle/input/d/suzuai/dataset-benign-malware/benign.
55
56      csv")
57
58  hashList = data['sha256'].to_list()
59
60  hashList = [f'{a}#{i}' for i, a in enumerate(hashList)]
61
62  multithread(download,hashList)
```

Hình 2.3: Log của quá trình tải file APK xuống

5. Thực hiện bước 5: Tạo dataset.

Từ output của code tải file apk, tạo 2 dataset là Benign và Malware, với mỗi dataset có 1000 file APK tương ứng benign/malware.

Hình 2.4: Dataset gồm 1000 file APK benign

Khó khăn : Do chưa thống nhất cách lọc file CSV nên nhóm đã lọc bằng thời gian và chọn ra các file dưới 30mb và cắt lấy mỗi loại benign/malware 1000 file, cách này mặc dù không sai nhưng còn rườm rà. Thay vào đó có thể lọc ra các file dưới 30mb và có vt-detection = 0 và lấy head(1000) để lấy 1000 file benign, tương tự với malware.
 → **Kết luận:** Dự án đã hoàn thành các bước chuẩn bị dữ liệu quan trọng, tạo nền tảng vững chắc cho các bước phân tích và nhận diện malware trong file APK trong các tuần tiếp theo.

Tuần Topic 2 (27/09 - 04/10)

Chủ đề tìm hiểu tuần 2: Làm sạch dữ liệu từ dữ liệu thô sau khi đã lấy từ AndroZoo.

Mục tiêu tuần Topic 2:

1. Tìm hiểu về cách thức tổ chức dữ liệu của dataset
2. Xử lí file CSV bằng Python
3. Tạo dataset mới sau khi làm sạch

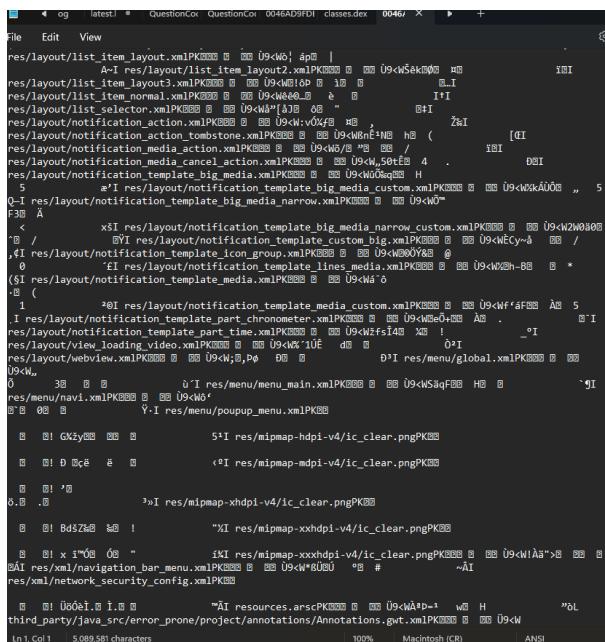
Thành viên tham gia

Đỗ Đăng Khoa, Nguyễn Việt Khiêm: Làm sạch dữ liệu

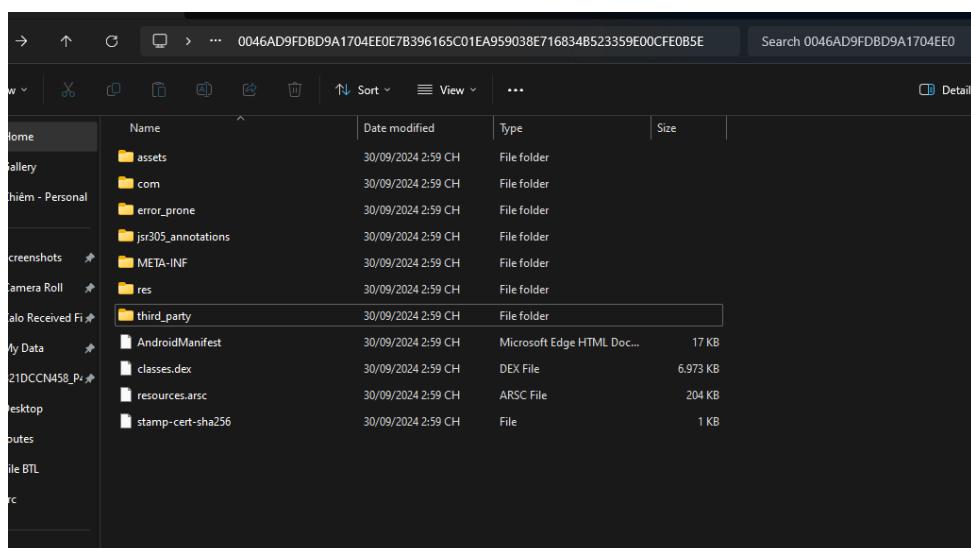
Hoàng Hữu Đức: Thống kê, hình dung dữ liệu

Kết quả của tuần Topic 2:

1. Thực hiện nội dung 1: Tìm hiểu về cách thức tổ chức dữ liệu của dataset
 - **File CSV:** File CSV của dataset chứa thông tin về file APK. Sau khi tiền xử lí ở Tuần 1, nhóm đã có 2 file CSV bao gồm: benign.csv chứa thông tin về các file APK là benign và malware.csv chứa thông tin về các file APK là malware. Các trường trong file CSV:
 - **sha256, sha1, md5:** Đây là các mã băm của file APK.
 - **apk size:** Kích thước của file APK.
 - **dex size:** Kích thước của tệp classes.dex (không tính các tệp dex khác).
 - **dex date:** Ngày ghi trên tệp dex bên trong tệp zip (đôi khi không chính xác hoặc bị thay đổi). Ngày dex hiện nay thường không đáng tin cậy: Hầu hết các ứng dụng từ Google Play đều có ngày dex là năm 1980.
 - **pkg name, vercode:** Tên gói Android và mã phiên bản (như được ghi trong tệp manifest). Lưu ý: pkg name có thể duy nhất trong một thị trường (ví dụ, hai tệp APK với cùng pkg name trên Google Play có thể thuộc về cùng một nhà phát triển).
 - **vt detection, vt scan date:** Số lượng phần mềm diệt virus từ VirusTotal (VT) phát hiện tệp APK này là phần mềm độc hại vào ngày vt scan date (nếu có).
 - **markets:** Danh sách các thị trường nơi chúng tôi đã thấy tệp APK này, phân tách bằng ký tự '|'. Lưu ý: Nếu không thấy một thị trường, không có nghĩa là tệp APK không xuất hiện trên thị trường đó, chỉ là chúng tôi chưa thấy nó ở đó.
 - **Tệp APK:** File .apk của dataset chứa mã nguồn phần mềm sau khi đã tải về.



Hình 2.5: File APK khi được mở bằng notepad (chứa các ký tự unicode sau khi đóng gói, đặt tên file bằng sha256)



Hình 2.6: File APK khi được đổi phương thức đọc thành .zip và giải nén

Sau khi giải nén file .apk, ta có thể thấy cấu trúc chung của một phần mềm Android:

- : /assets: Thư mục chứa các tài nguyên thô (raw assets) không được biên dịch. Ví dụ: file ảnh, file gif, âm thanh, hoặc dữ liệu khác mà ứng dụng sử dụng trực tiếp.
 - /res (resource): Chứa các tài nguyên như layout, chuỗi văn bản (strings), hình ảnh, màu sắc, style của ứng dụng. Đây là nơi chứa tất cả các tài nguyên mà Android sử dụng và sẽ được biên dịch khi xây dựng APK.
 - /META-INF: Thư mục chứa thông tin về chứng chỉ chữ ký số (digital signature) của ứng dụng, đảm bảo APK không bị sửa đổi khi cài đặt.
 - /AndroidManifest.xml: Đây là tệp kê khai quan trọng của ứng dụng. Nó chứa thông tin về package,

activity, service, quyền truy cập (permissions), phiên bản ứng dụng, và các thành phần khác mà hệ điều hành Android cần để chạy ứng dụng. Tóm lại đây chính là cấu hình phần mềm.

- /classes.dex: Đây là tệp chứa mã nguồn của ứng dụng sau khi đã được biên dịch thành Dalvik bytecode. Đây là định dạng mà máy ảo Dalvik và Android Runtime (ART) có thể thực thi được trên thiết bị Android.
- /resources.arsc: Tệp chứa tài nguyên biên dịch (compiled resources) của ứng dụng, bao gồm các chuỗi văn bản, định nghĩa style, và các tài nguyên khác mà ứng dụng sử dụng.
- /stamp-cert-sha256: Tệp này liên quan đến chứng chỉ SHA-256 dùng để kiểm tra tính toàn vẹn và bảo mật của tệp APK.
- và các file khác

2. Thực hiện nội dung 2: Xử lí file CSV bằng Python. Vì dataset được lưu trên Kaggle nên nhóm đã viết code Python trên Kaggle và trực tiếp xử lí dataset trên nền tảng này. Code xử lí:

```

1      import pandas as pd
2
3
4      def getPath(folder):
5          return [f"{folder}/{item}" for item in os.listdir(folder)]
6
7      def writeTxt(data, path):
8          with open(path, "w") as file:
9              file.write("\n".join(data))

benign = "/kaggle/input/benign"
malware_1 = "/kaggle/input/dataset-malware/malware1/content/drive/
MyDrive/New_Dataset/ApkDataset/batch_malware"

paths_benign = getPath(benign)
paths_malware_1 = getPath(malware_1)

benign_csv = pd.read_csv("/kaggle/input/dataset-benign-malware/benign.
csv")
malware_csv = pd.read_csv("/kaggle/input/dataset-benign-malware/malware
.csv")

benign_csv['label'] = 'benign'
malware_csv['label'] = 'malware'

merged_df = pd.concat([benign_csv, malware_csv], ignore_index=True)

merged_df

```

Thêm trường label = "benign" cho dataframe benign và trường label = "malware" cho dataframe malware. Sau đó hợp 2 dataframe thành 1.

	dex_date	apk_size	pkg_name	vercode	vt_detection	vt_scan_date	dex_size	markets	label
06D8D0224F5EC2953F82344F	2023-09-28 07:14:48	5089721	com.nfl.nba.ncaaf.nhl.mlbb.all.live.sportlive.news	1.900000e+01	0.0	2023-11-11 11:03:51	7140156	play.google.com	benign
7C202FCD66A81C9F3DAF7570	2023-09-08 13:31:46	31077305	com.funonlife.mobileC	1.370000e+02	0.0	2023-12-08 12:43:28	2272672	play.google.com	benign
F4F711B05B2BFEDCE1E92FD	2023-10-03 13:04:32	25732099	com.sycous.mysycous2	2.200000e+01	0.0	2023-12-07 08:41:13	8120984	play.google.com	benign
645B4B5ABF8128B040F85B6	2023-06-12 10:02:38	24228104	com.StoreForce.ESSMobile	4.201413e+06	0.0	2023-07-13 10:02:24	8391960	play.google.com	benign
3943833468FA597A7403CB04	2023-07-31 07:56:14	22624335	com.neroxxi.cronaldo	5.000000e+00	0.0	2024-01-13 07:51:06	6706192	play.google.com	benign
...
0A65807B7AE277EF4D3E7C62	2023-10-11 19:26:16	21159710	com.wurzelkraut.Stash2Go	4.640000e+02	2.0	2023-11-11 11:16:40	8532760	play.google.com	malware
F72169CC07F50756D219B0A2	2024-01-15 13:14:56	19063806	com.huahitong.ssydt	5.860000e+02	3.0	2024-03-19 06:57:01	11995672	appchina	malware
F8529AACDB6B81624006303	2023-08-29 00:44:28	17327672	com.global.fortress.classic	1.300000e+01	1.0	2023-11-16 06:51:07	8302668	play.google.com	malware
EA3DD8990C24FBE1B45D8EF9	2023-10-26 15:45:32	21778220	com.ebe.mobilebanking	2.023103e+09	3.0	2023-12-05 08:05:02	2367032	play.google.com	malware
E3CE16997090460DEA80D709	2023-06-29 02:01:16	5463027	cn.trinea.android.developertools	6.970000e+02	1.0	2023-07-28 16:33:48	7508488	play.google.com	malware

Hình 2.7: Dataframe sau khi xử lí

```
1 merged_df = merged_df.drop(columns=['sha1', 'md5']) #
2 merged_df.info()
```

Loại bỏ 2 trường không cần thiết (sha256, sha1, md5 là các mã hoá khác nhau của file APK nên chỉ cần giữ lại sha256). Dataframe không có null nên không cần xử lí phần bị thiếu.

```
1 sum(merged_df.duplicated())
1 merged_df["dex_date"] = pd.to_datetime(merged_df['dex_date'])
2 merged_df["vt_scan_date"] = pd.to_datetime(merged_df['vt_scan_date'])
```

Loại bỏ các file có kích thước nhỏ hơn 1mb

```
1 merged_df = merged_df.loc[merged_df['apk_size'] > 1000000]
2 merged_df.info()
```

Normalize các trường dữ liệu kiểu int64, float64 về khoảng [0, 1]

```
1 def normalize(column):
2     return (column - column.min()) / (column.max() - column.min())
3     merged_df[['apk_size', 'vercode', 'vt_detection', 'dex_size']] =
4         merged_df[['apk_size', 'vercode', 'vt_detection', 'dex_size']].apply(normalize)
```

Format lại kiểu dữ liệu: dex date và vt scan date đang có kiểu object format thành datetime, market và label là kiểu categorical format thành one hot encoding

```
1 one_hot_market = pd.get_dummies(merged_df['markets'])
2 one_hot_label = pd.get_dummies(merged_df['label'])
3 # Drop column B as it is now encoded
4 merged_df = merged_df.drop(['markets', 'label'], axis = 1)
5 # Join the encoded df
6 merged_df = merged_df.join(one_hot_market)
```

7

```
merged_df = merged_df.join(one_hot_label)
```

```
merged_df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 2019 entries, 0 to 2030
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   sha256          2019 non-null    object  
 1   dex_date         2019 non-null    datetime64[ns]
 2   apk_size         2019 non-null    float64 
 3   pkg_name         2019 non-null    object  
 4   vercode          2019 non-null    float64 
 5   vt_detection     2019 non-null    float64 
 6   vt_scan_date    2019 non-null    datetime64[ns]
 7   dex_size         2019 non-null    float64 
 8   appchina         2019 non-null    bool    
 9   fdroid            2019 non-null    bool    
 10  play.google.com  2019 non-null    bool    
 11  benign            2019 non-null    bool    
 12  malware           2019 non-null    bool    
dtypes: bool(5), datetime64[ns](2), float64(4), object(2)
memory usage: 151.8+ KB
```

Hình 2.8: Dataframe

3. Thực hiện nội dung 3: Tạo dataset mới sau khi làm sạch

```
df.to_csv('/kaggle/input/data.csv', index=False)
```

Khó khăn

Trong tuần này, nhóm gặp một số khó khăn trong việc thông nhất tham số đầu vào cho mô hình. Đặc biệt, chúng tôi đang tìm kiếm cách rời rạc hóa cho SHA256 để cải thiện khả năng phân tích. Tuy nhiên, do những thách thức kỹ thuật, nhóm đã quyết định tạm thời lưu trường SHA256 ở dạng nguyên bản. Điều này giúp chúng tôi có thể tiếp tục phân tích mà không bị gián đoạn, mặc dù có thể không tận dụng hết được lợi ích của việc rời rạc hóa.

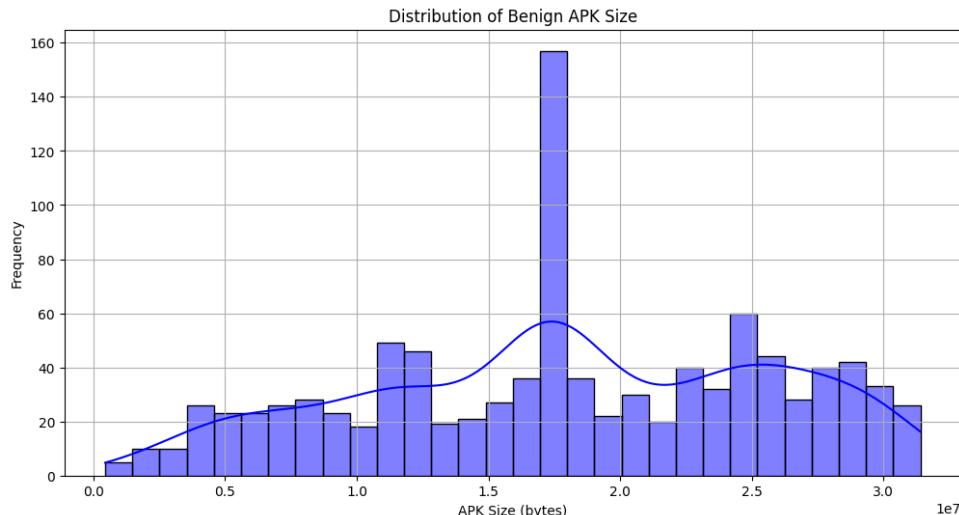
Thống kê và Trực quan hóa Dữ liệu

Biểu đồ

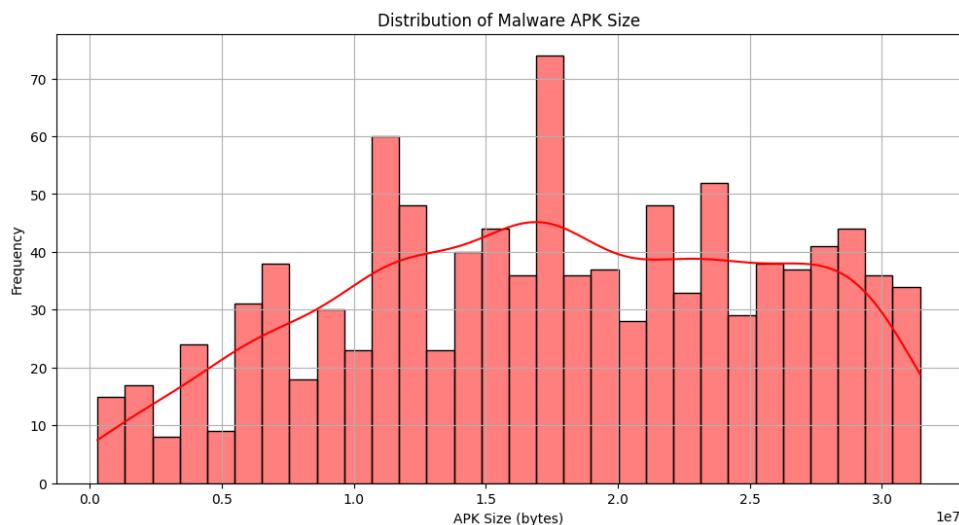
Trong phần này, chúng tôi trình bày các biểu đồ để trực quan hóa kích thước của các file APK thuộc hai loại: benign và malware. Các biểu đồ này bao gồm:

- **Biểu đồ Histogram:** Hiển thị phân bố kích thước APK, giúp nhận diện sự phân bố tổng quát của cả hai loại file.
- **Biểu đồ Boxplot:** Giúp xác định các giá trị ngoại lệ và khoảng phân bố của kích thước APK cho từng loại.
- **Biểu đồ KDE (Kernel Density Estimate):** Cung cấp một ước lượng mật độ cho phân phối kích thước APK, cho phép nhìn nhận rõ hơn về cấu trúc phân bố.
- **Biểu đồ Tân suất:** Hiển thị số lần xuất hiện của từng khoảng kích thước APK cho cả hai loại, từ đó giúp xác định các xu hướng kích thước.

Biểu Đồ Histogram



Hình 2.9: Benign histogram

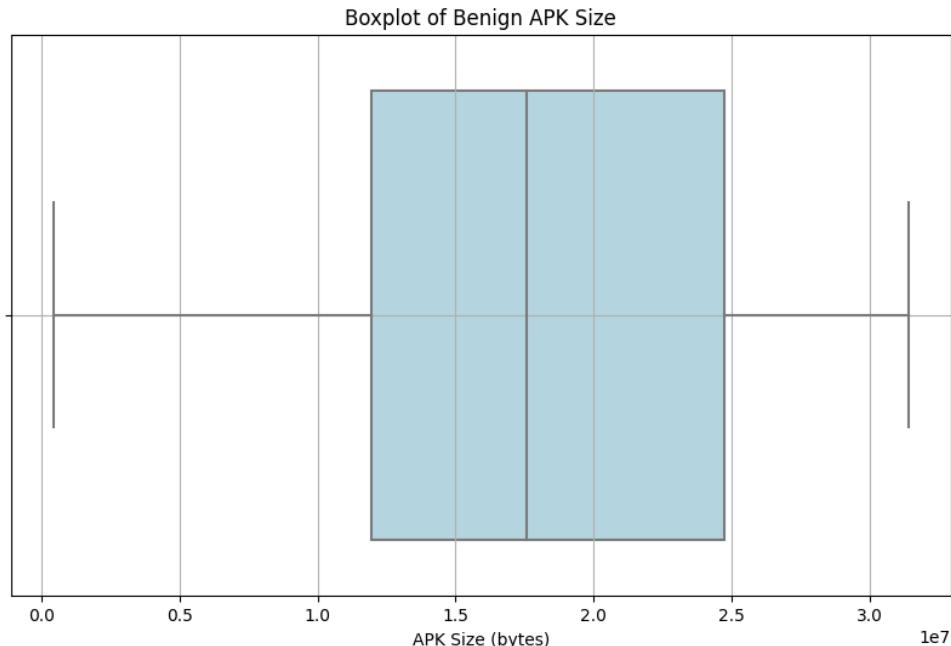


Hình 2.10: Malware histogram

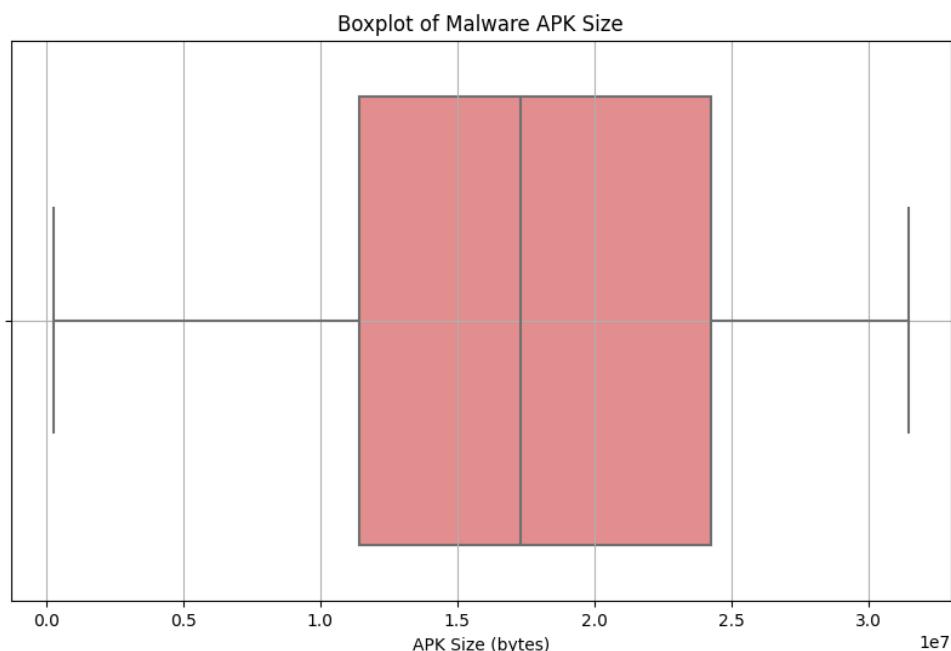
Biểu đồ histogram cho thấy sự phân bố kích thước của các file APK malware và benign. Qua phân tích, chúng tôi nhận thấy rằng:

- Đối với các file benign, kích thước thường tập trung trong khoảng từ 10 đến 20 MB, cho thấy đây là kích thước phổ biến của các ứng dụng an toàn.
- Ngược lại, kích thước của các file malware có sự phân bố rộng hơn, với nhiều file có kích thước nhỏ hơn 30 MB. Điều này cho thấy rằng nhiều malware được thiết kế để nhẹ hơn nhằm dễ dàng lén lút vào hệ thống.

Biểu Đồ Boxplot



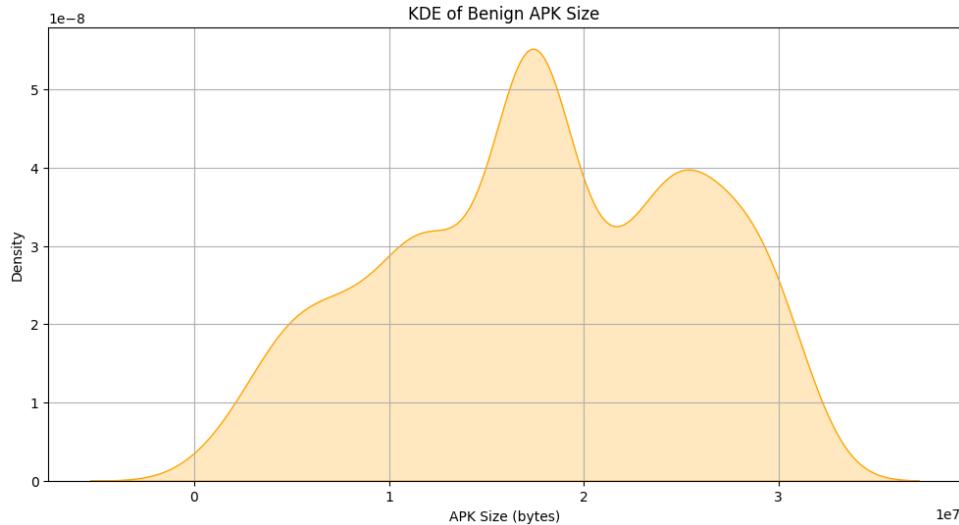
Hình 2.11: Benign boxplot



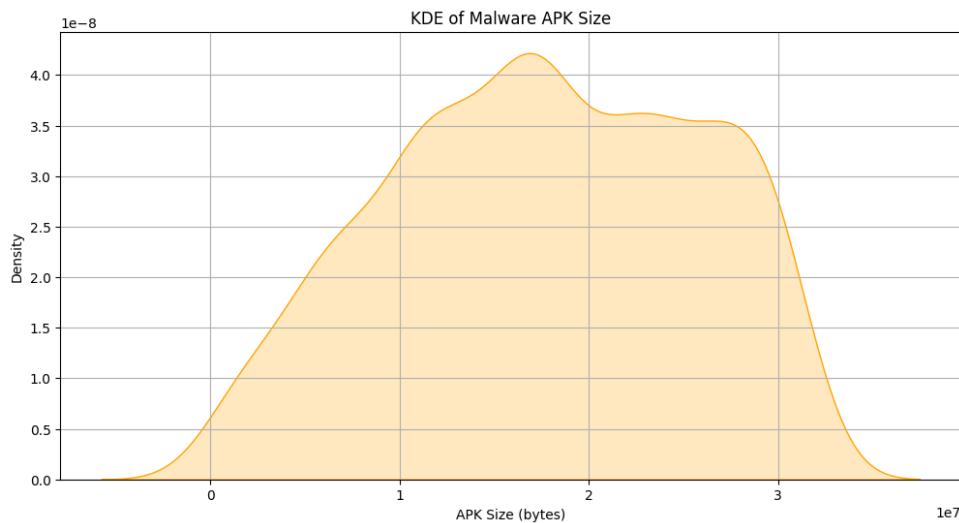
Hình 2.12: Malware boxplot

Biểu đồ boxplot cho thấy kích thước APK trung vị của benign khoảng 15 MB, trong khi malware có trung vị khoảng 10 MB. Phân tích sâu hơn cho thấy:

- Các giá trị ngoại lệ đáng chú ý ở cả hai loại cho thấy có nhiều file có kích thước lớn hơn, cụ thể là có một số file đạt đến kích thước lớn hơn 100 MB. Điều này có thể liên quan đến các ứng dụng đa chức năng hoặc các ứng dụng chứa nhiều tài nguyên.

Biểu Đồ KDE

Hình 2.13: Benign kde

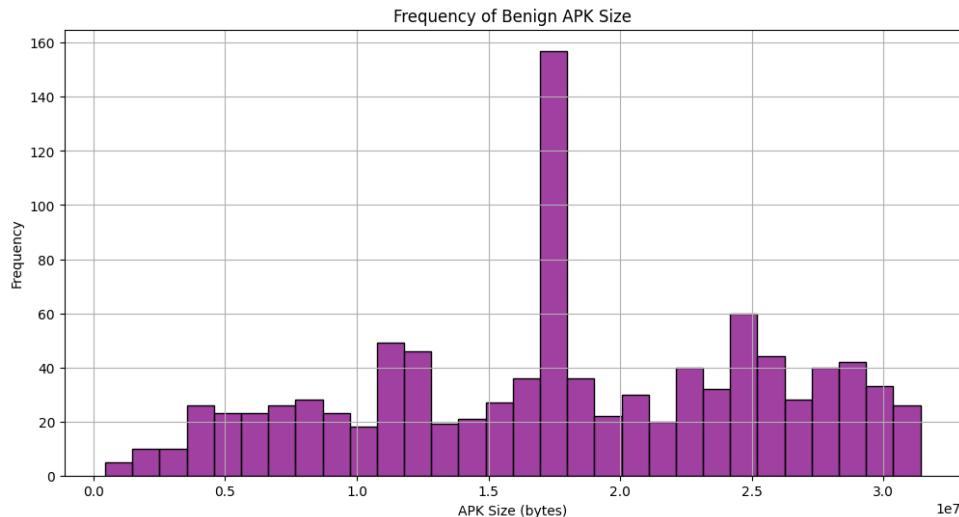


Hình 2.14: Malware kde

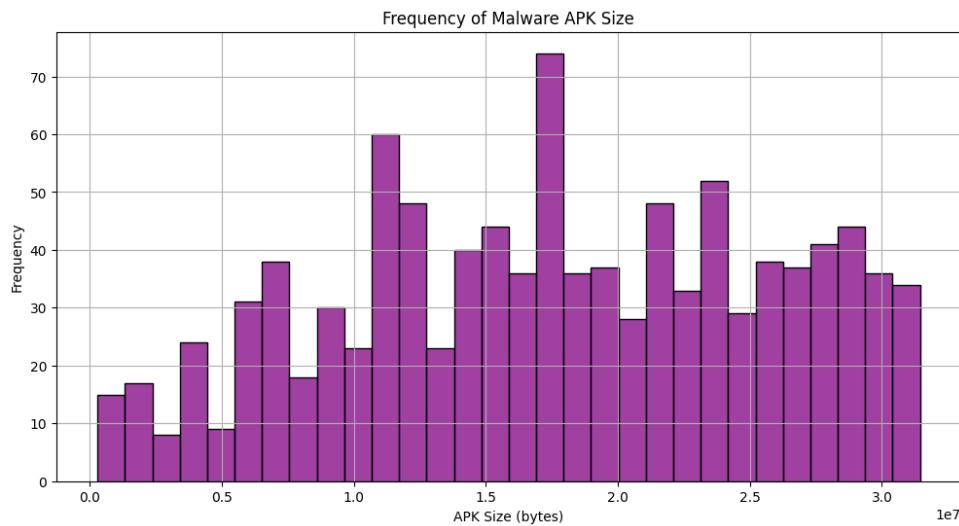
Biểu đồ KDE cung cấp cái nhìn mượt mà hơn về phân bố kích thước APK của cả hai loại. Phân tích cho thấy:

- Đường cong mật độ cho thấy rằng benign thường có kích thước lớn hơn malware. Điều này có thể phản ánh rằng các ứng dụng an toàn thường chứa nhiều chức năng và dữ liệu hơn.
- Hơn nữa, sự chồng chéo giữa hai phân bố cũng cho thấy một số file benign có thể bị nhầm lẫn với malware nếu chỉ dựa vào kích thước.

Biểu Đồ Tần Suất



Hình 2.15: Benign frequency



Hình 2.16: Malware frequency

Biểu đồ tần suất cho thấy số lượng file APK ở từng khoảng kích thước cho cả hai loại. Phân tích cho thấy:

- Các khoảng kích thước từ 10 đến 20 MB có tần suất cao nhất cho benign, cho thấy rằng đây là kích thước điển hình cho các ứng dụng an toàn.
- Đối với malware, tần suất cao hơn được ghi nhận ở khoảng kích thước nhỏ hơn 30 MB, nhấn mạnh việc thiết kế nhẹ hơn của nhiều malware.

Phân Tích

Qua việc trực quan hóa dữ liệu, rút ra một số nhận định quan trọng:

- Phân bố kích thước:** Phần lớn các file benign có kích thước lớn hơn so với malware. Điều này có thể gợi ý rằng nhiều malware được thiết kế nhẹ nhàng nhằm dễ dàng lén lút vào hệ thống.
- Giá trị ngoại lệ:** Cả hai loại file đều có giá trị ngoại lệ với một số file có kích thước lớn hơn 100 MB. Điều

này cần được xem xét kỹ lưỡng hơn để xác định chức năng của chúng, và có thể yêu cầu các bước kiểm tra bổ sung để xác định xem chúng có phải là nguy hiểm hay không.

- **Tính khả thi trong phân tích:** Dữ liệu về kích thước APK có thể là một chỉ số hữu ích trong việc phân loại file. Tuy nhiên, cần kết hợp với các thông số khác như nội dung, độ phức tạp và hành vi để có được cái nhìn chính xác hơn về tính an toàn.

Kết Luận

Phân tích kích thước của các file APK malware và benign thông qua các biểu đồ trực quan hóa đã cung cấp cái nhìn sâu sắc về đặc điểm của cả hai loại. Việc sử dụng các phương pháp thống kê và trực quan hóa giúp chúng tôi hiểu rõ hơn về cấu trúc của dữ liệu, từ đó hỗ trợ cho quá trình phát triển mô hình phân loại. Trong tương lai, nhóm sẽ tiếp tục tìm cách cải thiện các tham số đầu vào và nghiên cứu sâu hơn về các giá trị ngoại lệ cũng như các yếu tố khác ảnh hưởng đến tính an toàn của các file APK.

→ Kết luận: Dự án đã hoàn thành các bước chuẩn bị dữ liệu quan trọng, tạo nền tảng vững chắc cho các bước phân tích, thống kê và xây dựng mô hình nhận diện malware trong file APK trong các tuần tiếp theo.

Tuần Topic 3 (05/10 - 12/10)

Chủ đề tìm hiểu tuần 3: Thực hiện doc2vec cho dữ liệu java được tạo từ apk và công cụ dịch ngược apk thành mã nguồn(java)

Thành viên tham gia

Đào Duy Thông - 33%

Vũ Thành Tuyên - 33%

Hoàng Hữu Đức - 33%

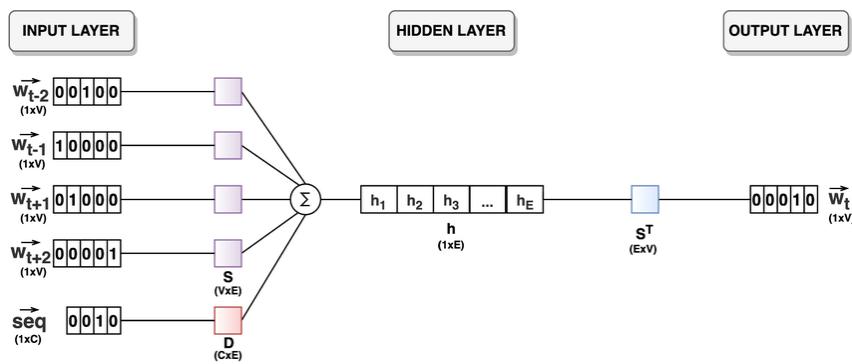
Mục tiêu tuần Topic 3: Trình bày khái quát về lý thuyết, cách áp dụng doc2vec vào dữ liệu ở dạng java (do được tạo ra từ các file apk)

1. Tìm hiểu về lý thuyết doc2vec
2. Lý do áp dụng kĩ thuật doc2vec vào bài toán phân loại apk
3. Cách sử dụng kĩ thuật doc2vec vào bài toán phân loại apk

Kết quả của tuần Topic:

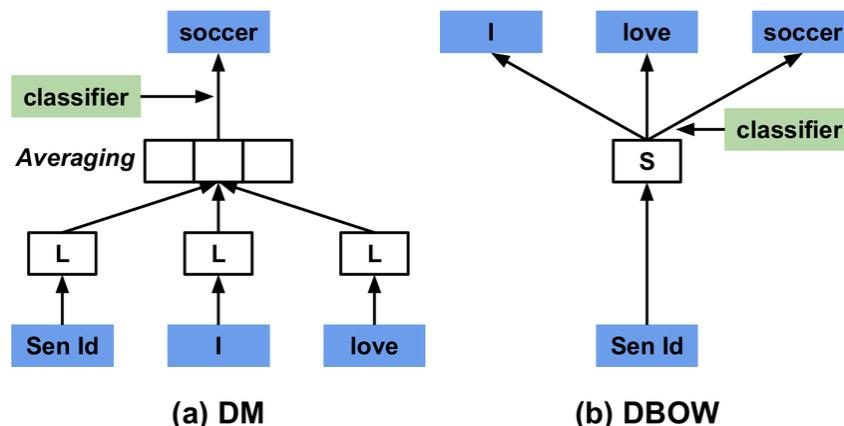
1. Tìm hiểu về lý thuyết doc2vec

- (a) Mục tiêu của Doc2Vec là học các biểu diễn vector cho các đoạn văn, câu, hoặc tài liệu, thay vì chỉ học vector cho các từ đơn lẻ như trong Word2Vec. Bằng cách này, Doc2Vec không chỉ nắm bắt ý nghĩa của từng từ mà còn hiểu được ngữ cảnh tổng thể của một đoạn văn hoặc tài liệu, từ đó giúp mô hình có khả năng biểu diễn ngữ nghĩa của các văn bản dài.



Hình 2.17: Architecture Doc2vec

- (b) Có hai kiến trúc chính trong Doc2Vec là Distributed Memory (DM) và Distributed Bag of Words (DBOW). Với DM, mô hình học cách dự đoán từ tiếp theo trong một câu dựa trên các từ trước đó và vector của tài liệu, kết hợp cả thông tin về ngữ cảnh và cấu trúc của văn bản. Điều này giúp vector biểu diễn của tài liệu giữ được ngữ cảnh mà các từ xuất hiện. Ngược lại, DBOW hoạt động tương tự như mô hình Skip-gram của Word2Vec, khi nó học cách dự đoán các từ ngẫu nhiên trong tài liệu mà không sử dụng thông tin ngữ cảnh. Dù cách tiếp cận khác nhau, cả hai đều nhằm mục đích tạo ra một biểu diễn vector cho toàn bộ tài liệu.
- (c) Doc2Vec đã được sử dụng rộng rãi trong nhiều ứng dụng, từ phân loại văn bản, tìm kiếm thông tin, đến phân tích cảm xúc và khuyến nghị. Nó giúp tạo ra các biểu diễn vector có khả năng nắm bắt ngữ nghĩa và mối quan hệ giữa các tài liệu, từ đó giúp các mô hình học máy có thể dễ dàng xử lý văn bản hơn. Một trong những lợi ích chính của Doc2Vec là khả năng làm việc với các văn bản có độ dài và ngữ cảnh khác nhau, điều mà Word2Vec không thể làm được. Điều này làm cho Doc2Vec trở thành một công cụ mạnh mẽ trong xử lý ngôn ngữ tự nhiên.



Hình 2.18: DM and DBOW

2. Lý do áp dụng kỹ thuật doc2vec vào bài toán phân loại apk: khi mà phân tích file apk ta nhận thấy ta có thể dịch ngược file apk thành file java, và từ đó ta có thể làm giàu đồ thị hơn với các đặc trưng trích xuất từ file ngôn ngữ java

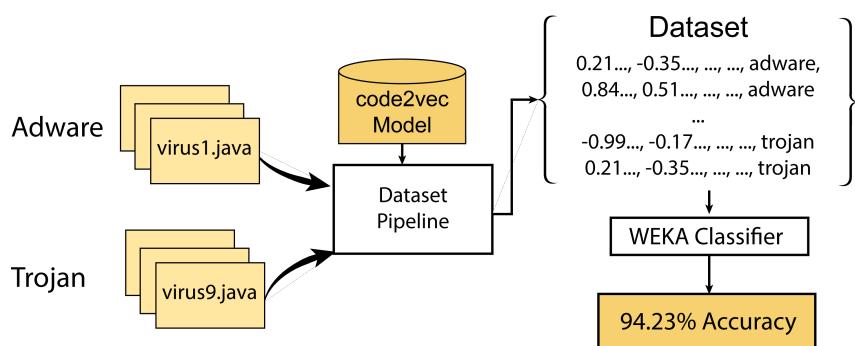
- (a) Doc2Vec vào ngôn ngữ Java, chúng ta có thể xây dựng các ứng dụng phân tích văn bản và xử lý ngôn ngữ tự nhiên (NLP) mạnh mẽ. Doc2Vec giúp chuyển các đoạn văn bản, tài liệu, hoặc đoạn mã nguồn

thành các vector có kích thước cố định, biểu diễn ngữ nghĩa của nội dung. Điều này có thể áp dụng cho nhiều bài toán liên quan đến mã nguồn trong Java, chẳng hạn như phân loại, tìm kiếm và gợi ý mã, phát hiện lối, và phân tích mã nguồn.

- (b) Doc2Vec mang lại nhiều lợi ích khi áp dụng vào việc phân tích và xử lý mã nguồn Java. Nó giúp biểu diễn mã nguồn dưới dạng các vector số học, cho phép các mô hình học máy hiểu được ngữ nghĩa của các đoạn mã, từ đó hỗ trợ phân loại các file apk

3. Cách sử dụng kỹ thuật doc2vec vào bài toán phân loại apk: Kỹ thuật Doc2Vec có thể được sử dụng để phân loại các tệp APK (Android Package) bằng cách chuyển đổi mã nguồn java của APK thành các biểu diễn vector, sau đó sử dụng các vector này để huấn luyện mô hình phân loại.

- (a) APK chứa mã nguồn (thường là mã Java hoặc mã Smali)
- (b) Mã nguồn: Mã nguồn Java hoặc Smali cần được chuyển thành dạng văn bản, tương tự như một tài liệu để có thể áp dụng Doc2Vec. Có thể coi mỗi đoạn mã (class, method) là một "tài liệu".
- (c) Sau khi có thể chuyển đổi thành các file java thì ta có thể nhúng các đoạn văn bản đó cho một mô hình đã được đào tạo từ trước trên huggingface để thu được các vector đặc trưng cho file apk của ta



Hình 2.19: Embedding java with doc2vec

4. Công cụ dịch ngược: JADX và Androguard Trong quá trình phân tích các tệp APK, việc sử dụng các công cụ dịch ngược là rất quan trọng để trích xuất mã nguồn từ các tệp nhị phân. Hai công cụ phổ biến trong lĩnh vực này là JADX và Androguard, mỗi công cụ đều có những ưu và nhược điểm riêng.

- (a) **Java decompiler** JADX là một công cụ dịch ngược mã nguồn từ tệp APK thành mã Java có thể đọc được. Nó sử dụng các thuật toán phân tích tĩnh để tái tạo mã nguồn, giúp người dùng dễ dàng hiểu cấu trúc và logic của ứng dụng.

Ưu điểm:

- **Giao diện người dùng thân thiện:** JADX cung cấp giao diện đồ họa (GUI) giúp người dùng dễ dàng duyệt qua mã nguồn và tìm kiếm thông tin.
- **Chất lượng mã nguồn tái tạo:** JADX có khả năng tái tạo mã nguồn với cấu trúc rõ ràng, giúp dễ dàng nhận diện các lớp và phương thức.
- **Hỗ trợ đa nền tảng:** JADX có thể chạy trên nhiều hệ điều hành khác nhau, bao gồm Windows, macOS và Linux.

- **Tính năng tìm kiếm mạnh mẽ:** Jadx cho phép tìm kiếm theo từ khóa, giúp người dùng nhanh chóng xác định các phần cụ thể trong mã nguồn.

Nhược điểm:

- **Không hoàn hảo:** Mặc dù Jadx tái tạo mã nguồn tốt, nhưng đôi khi nó có thể gặp khó khăn với các đoạn mã phức tạp hoặc mã được mã hóa.
- **Hạn chế trong phân tích động:** Jadx chủ yếu tập trung vào phân tích tĩnh, do đó không cung cấp khả năng phân tích hành vi động của ứng dụng.

(b) **Androguard** Androguard là một thư viện Python chuyên dụng cho việc phân tích và xử lý các tệp APK. Nó không chỉ hỗ trợ dịch ngược mà còn cung cấp các công cụ để phân tích sâu hơn về cấu trúc và hành vi của ứng dụng.

Ưu điểm:

- **Tính linh hoạt cao:** Androguard có thể được tích hợp vào các ứng dụng Python khác, cho phép người dùng tùy chỉnh và mở rộng tính năng theo nhu cầu.
- **Phân tích tĩnh và động:** Androguard cung cấp khả năng phân tích tĩnh và hỗ trợ các công cụ phân tích động, giúp người dùng hiểu rõ hơn về hành vi của ứng dụng.
- **Hỗ trợ tìm kiếm:** Androguard cho phép tìm kiếm và trích xuất thông tin từ các tệp APK một cách dễ dàng.
- **Cộng đồng và tài liệu phong phú:** Androguard có một cộng đồng phát triển mạnh mẽ và tài liệu phong phú, giúp người dùng dễ dàng tìm kiếm hỗ trợ và hướng dẫn.

Nhược điểm:

- **Đường cong học tập:** Androguard có thể yêu cầu kiến thức lập trình Python để sử dụng hiệu quả, điều này có thể là một rào cản cho những người không quen với lập trình.
- **Thiếu giao diện đồ họa:** Androguard chủ yếu hoạt động trên dòng lệnh, không có giao diện đồ họa trực quan như Jadx, điều này có thể gây khó khăn cho người dùng mới.

(c) **So sánh Jadx và Androguard**

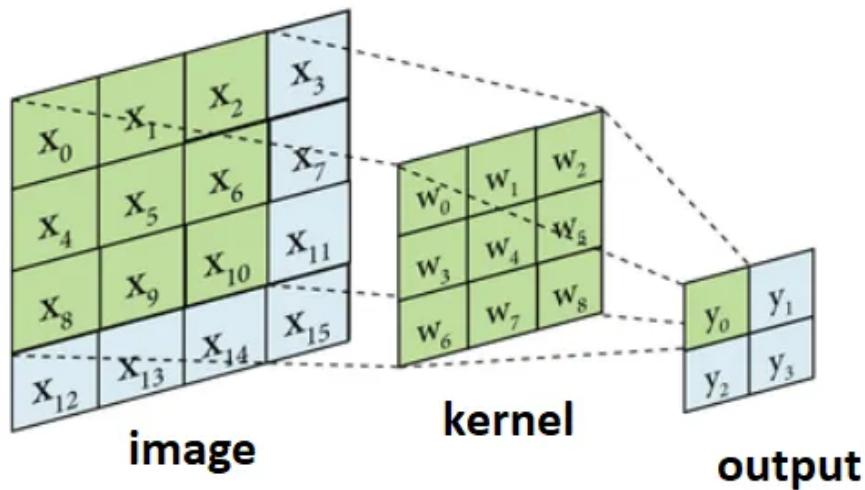
- **Giao diện người dùng:** Jadx có giao diện đồ họa thân thiện, trong khi Androguard chỉ hỗ trợ dòng lệnh, điều này khiến Jadx dễ sử dụng hơn cho người mới.
- **Chất lượng mã nguồn:** Jadx thường cung cấp mã nguồn có cấu trúc rõ ràng hơn, trong khi Androguard có thể cho phép người dùng khai thác mã nguồn một cách linh hoạt hơn thông qua lập trình.
- **Phân tích:** Androguard cung cấp khả năng phân tích động tốt hơn, trong khi Jadx chủ yếu tập trung vào phân tích tĩnh.
- **Linh hoạt:** Androguard có tính linh hoạt cao hơn nhờ khả năng tích hợp vào các ứng dụng Python khác, trong khi Jadx chủ yếu là một công cụ độc lập.

→ Kết luận: Doc2Vec giúp biến các đoạn mã và thông tin của APK thành các vector có ý nghĩa, từ đó hỗ trợ phân loại các tệp APK. Kỹ thuật này đặc biệt hữu ích trong các bài toán phát hiện phần mềm độc hại, khi mà nội dung mã nguồn của các ứng dụng có thể tiết lộ hành vi nguy hiểm. Cả JADX và Androguard đều là những công cụ mạnh mẽ cho việc phân tích các tệp APK. Lựa chọn giữa chúng phụ thuộc vào nhu cầu cụ thể của người dùng và mức độ quen thuộc với lập trình và phân tích mã nguồn.

Mô hình Graph Convolutional Network (GCN)

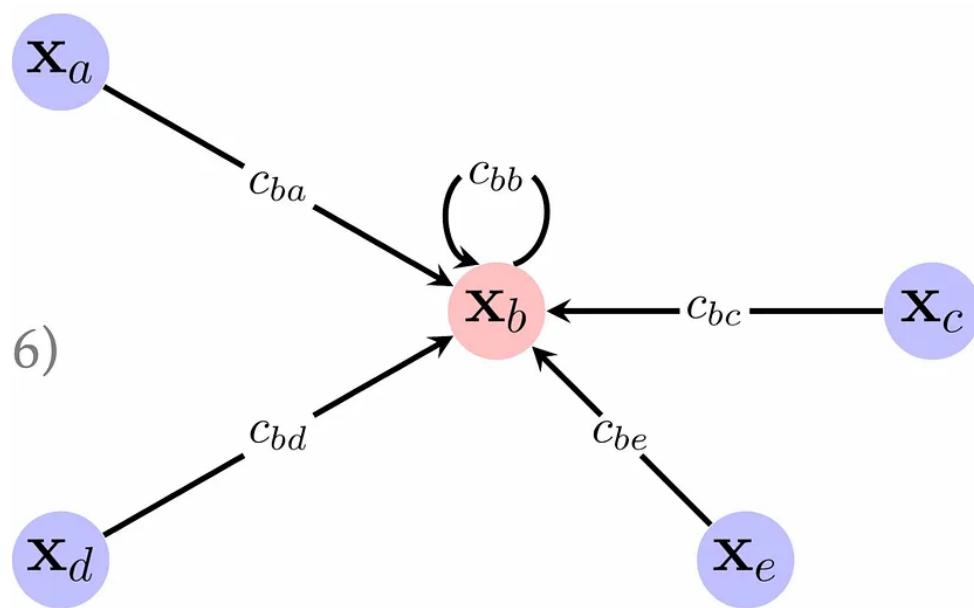
1. Lý thuyết về GCN

- (a) Trong học sâu, một trong những lớp cơ bản nhất là Convolutional Neural Network (CNN). Lớp này thu thập và xử lý thông tin của các pixel xung quanh nhằm tạo ra lớp thông tin cô đọng với số chiều ít hơn.

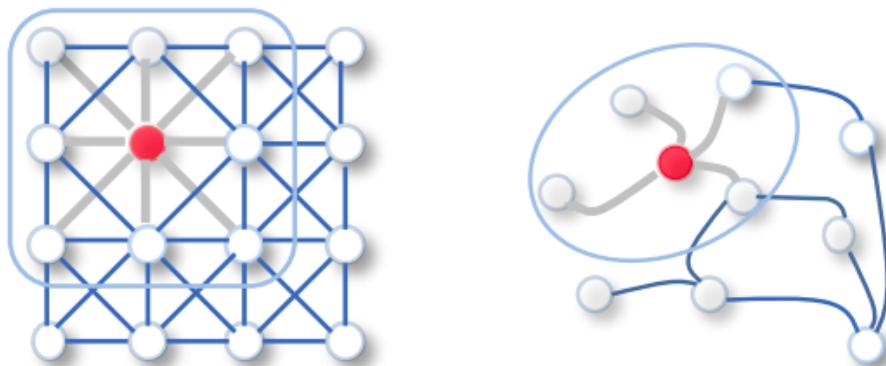


Hình 2.20: CNN

- (b) Một lớp GCN hoạt động tương tự so với CNN, tuy nhiên thay cho các pixel lân cận, thông tin các node lân cận được sử dụng bao gồm cả node trung tâm



Hình 2.21: Minh họa về GCN



(a) 2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes the weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size.

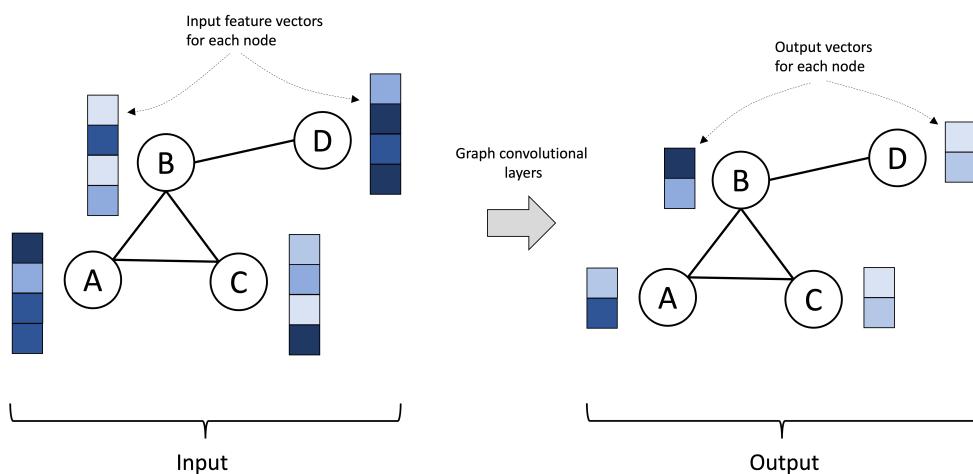
(b) Graph Convolution. To get a hidden representation of the red node, one simple solution of the graph convolutional operation is to take the average value of the node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size.

Hình 2.22: So sánh CNN với GCN

2. Input và output của mô hình GCN

(a) Input: GCN lấy input là một đồ thị với một tập các feature vector. Mỗi feature tương ứng với thông tin của một node

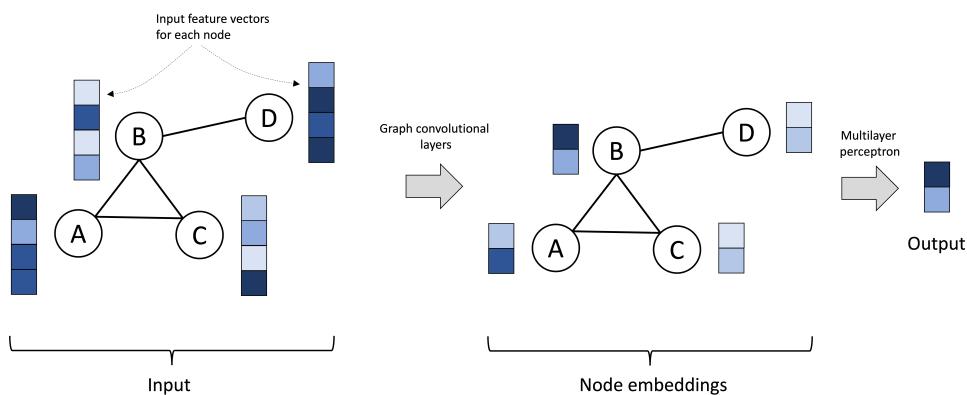
Output: Là một đồ thị tương tự với các output vector cho mỗi node của đồ thị. Các vector thường có số chiều ít hơn với các vector của input.



Hình 2.23: Input và output của GCN

(b) **Ứng dụng:** Với những tác vụ trên node như bài toán phân loại, các vector output của GCN có thể coi như là kết quả cuối cùng của mô hình. Trong bài toán phân loại node, các vector tương đương với xác suất node thuộc class nào.

Ngoài ra, có những bài toán ở mức đồ thị quan tâm đến kết quả của toàn bộ đồ thị. Ví dụ trong bài toán phân loại, có thể ta quan tâm đến việc phân loại cả đồ thị hơn là từng node. Đối với bài toán này, thay vì sử dụng trực tiếp kết quả của GCN, các vector được đưa vào một mạng khác như multilayer perceptron để đưa ra một kết quả duy nhất là class của đồ thị.



Hình 2.24: GCN cho phân loại đồ thị

3. Ứng dụng của mô hình GCN cho bài toán phát hiện mã độc.

Trong bài toán phát hiện mã độc, mỗi file apk được coi như một đồ thị. Các file này sau khi được giải mã sẽ có dạng một đồ thị với các node tương đương với các hàm trong file, các cạnh của đồ thị tương ứng với các lời gọi hàm. GCN có vai trò nhận input là các đồ thị này và đưa ra kết quả là class tương ứng cho file đó (malware hoặc là benign).

Tuần Topic 4 (13/10 - 19/10)

Chủ đề tìm hiểu tuần 4:

- Sử dụng Orange Data Mining gắn nhãn

2. Tìm hiểu lại về Doc2vec
3. Tìm hiểu lại về GCN

Thành viên tham gia

- Đào Duy Thông - Sử dụng Orange - 33
- Vũ Thành Tuyên
-

Mục tiêu tuần Topic 4:

1. Sử dụng Orange Data Mining dự đoán nhãn

Từ file CSV đã gắn nhãn benign/malware, sử dụng các mô hình như Logistic Regression, Random Forest,... để gắn nhãn cho dữ liệu chưa có nhãn.

2. Tìm hiểu lại về Doc2vec
3. Tìm hiểu lại về GCN

Kết quả của tuần Topic 4:

1. Thực hiện nội dung 1

Bài toán : sử dụng Orange data mining để dự đoán benign/malware dựa trên các đặc trưng như apk_size, vt_detection, dexsize và vercode. Trong đó :

- apk_size: Tổng kích thước của tệp APK tính theo byte, bao gồm mã nguồn của ứng dụng, tài nguyên, tệp tài sản, và bất kỳ tệp bổ sung nào được đóng gói vào APK.
- dex_size: Kích thước của tệp classes.dex, đây là tệp Dalvik Executable (DEX) chính chứa bytecode Java đã được biên dịch, chạy trên hệ điều hành Android. Trường này chỉ tính kích thước của tệp DEX đầu tiên (classes.dex), không bao gồm các tệp DEX bổ sung.
- vercode: Mã phiên bản được xác định trong tệp manifest của ứng dụng Android. Đây là một giá trị số nguyên đại diện cho phiên bản của APK, được tăng dần mỗi khi ứng dụng được cập nhật. Nó có thể không khớp với chuỗi phiên bản hiển thị (ví dụ: “1.2.3”).
- vt_detection: Số lượng công cụ diệt virus (AV) trên VirusTotal đã gắn cờ APK này là phần mềm độc hại. VirusTotal tổng hợp kết quả từ nhiều công cụ diệt virus để đánh giá xem tệp có độc hại hay an toàn. Trong lần thử nghiệm này, sẽ thử tập dữ liệu không có đặc trưng vt_detection vì để có vt_detection thì phải dùng ứng dụng khác quét

Dữ liệu huấn luyện : 2031 bản ghi, đã được gắn nhãn benign/ malware

Dữ liệu dự đoán : 1056 bản ghi chưa được gắn nhãn

Các widget sử dụng :

- CSV File import



CSV File Import !

Hình 2.25: Widget CSV File Import

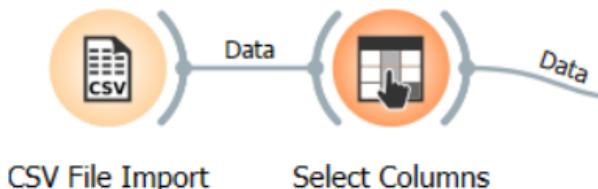
CSV File Import dùng để nhập bảng dữ liệu từ tệp định dạng CSV.

Đầu ra :

- Dữ liệu: tập dữ liệu từ tệp .csv
- Khung dữ liệu: đối tượng DataFrame của pandas

Tiện ích CSV File Import đọc các tệp được phân cách bằng dấu phẩy và gửi tập dữ liệu đến kênh đầu ra của nó. Các dấu phân cách tệp có thể là dấu phẩy, dấu chấm phẩy, dấu cách, tab hoặc dấu phân cách được xác định thủ công. Lịch sử của các tệp được mở gần đây nhất được duy trì trong tiện ích.

- **Select Columns**



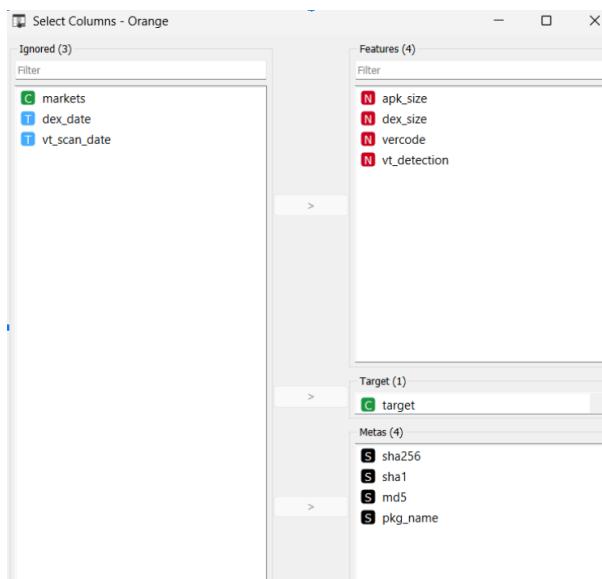
Hình 2.26: Widget Select Columns

Lựa chọn thủ công các thuộc tính dữ liệu và thành phần của miền dữ liệu.

Đầu vào : Dữ liệu: tập dữ liệu đầu vào

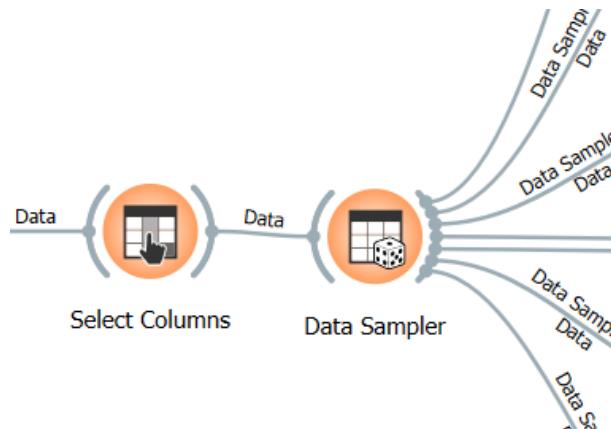
Đầu ra : Dữ liệu: tập dữ liệu với các cột được thiết lập trong tiện ích

Tiện ích Select Columns được sử dụng để biên soạn thủ công miền dữ liệu. Người dùng có thể quyết định thuộc tính nào sẽ được sử dụng và cách sử dụng. Orange phân biệt giữa các thuộc tính thông thường, thuộc tính lớp (tùy chọn) và thuộc tính meta. Ví dụ, để xây dựng mô hình phân loại, miền sẽ bao gồm một tập hợp các thuộc tính và một thuộc tính lớp rời rạc.



Hình 2.27: Widget Select Columns

- Data Sampler



Hình 2.28: Widget Data Sampler

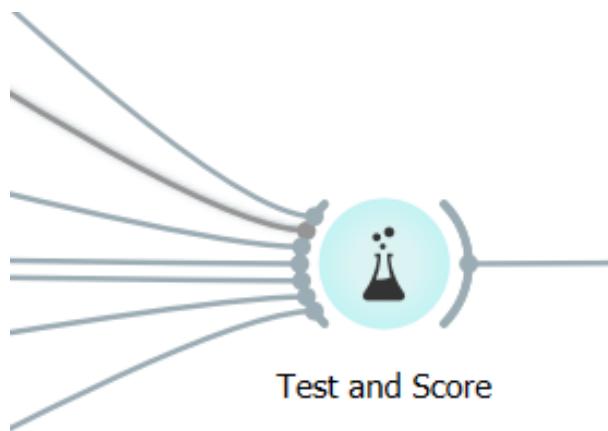
Data Sampler trong Orange là một widget dùng để chọn ngẫu nhiên một phần dữ liệu từ tập dữ liệu ban đầu. Nó giúp trích xuất các mẫu nhỏ từ dữ liệu để thử nghiệm mô hình hoặc để đánh giá chất lượng của dữ liệu trước khi sử dụng toàn bộ.

Đầu vào Dữ liệu: tập dữ liệu đầu vào

Đầu ra

- Mẫu dữ liệu: các trường hợp dữ liệu được lấy mẫu
- Dữ liệu còn lại: dữ liệu ngoài mẫu

- Test and Score



Hình 2.29: Widget Test and Score

Test and Score là một widget trong Orange dùng để đánh giá hiệu suất của các mô hình học máy. Nó kiểm tra mô hình bằng cách sử dụng các phương pháp kiểm tra chéo và đo lường các chỉ số như độ chính xác, AUC, F1, độ nhạy và độ đặc hiệu.

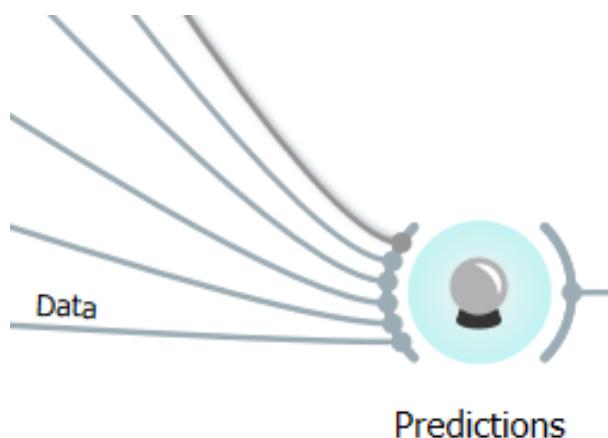
Đầu vào

- Dữ liệu: tập dữ liệu đầu vào
- Dữ liệu thử nghiệm: dữ liệu riêng biệt để thử nghiệm
- Learner : thuật toán

- **Các mô hình**

Sử dụng các mô hình như SVM, Random Forest, Neural Network, kNN, Logistic Regression

- **Predictions**



Hình 2.30: Widget Predictions

Hiển thị dự đoán của mô hình về dữ liệu.

Đầu vào

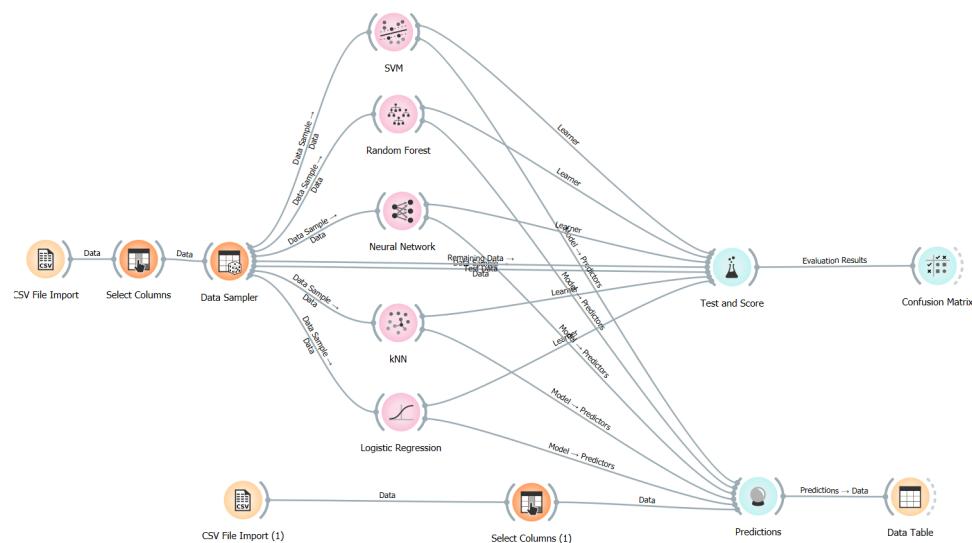
- Dữ liệu: tập dữ liệu đầu vào
- Các thuật toán : các thuật toán được sử dụng trên dữ liệu

Đầu ra

- Dự đoán: dữ liệu có thêm dự đoán
- Kết quả đánh giá: kết quả thử nghiệm thuật toán phân loại

Tiện ích nhận một tập dữ liệu và một hoặc nhiều bộ dự đoán (mô hình dự đoán, không phải thuật toán học). Nó xuất ra dữ liệu và các dự đoán.

Mô hình Orange



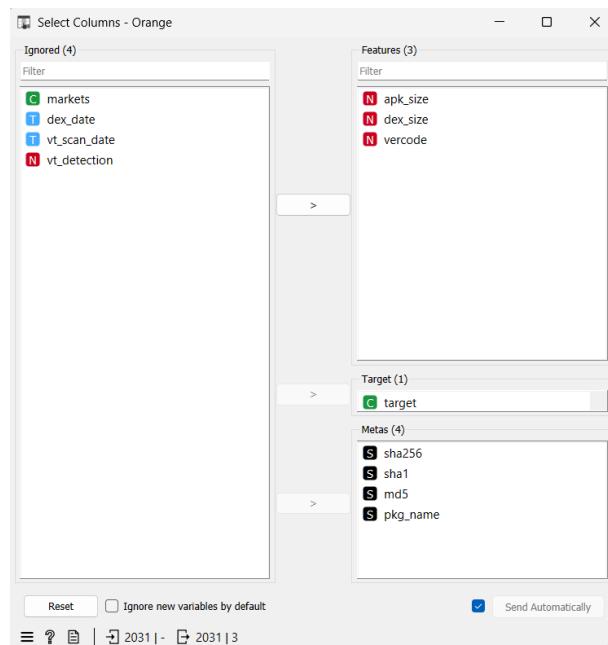
Hình 2.31: Kết nối các widget trong Orange Data Mining

Luồng hoạt động

(a) Nhập dữ liệu từ file CSV bằng CSV File Import

(b) Phân loại các cột bằng Select Columns

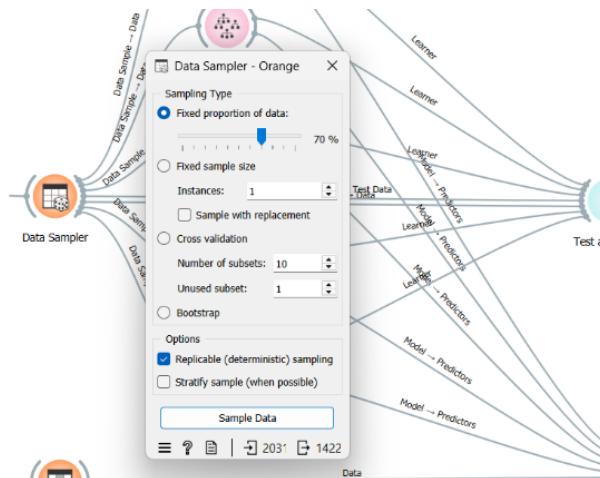
Nhập dữ liệu từ CSV File Import Dùng widget Select Columns để chọn ra các cột thuộc tính, cột target , loại bỏ các cột meta.



Hình 2.32: Lựa chọn các thuộc tính để huấn luyện mô hình

(c) Sử dụng widget Data Sampler để chia tập dữ liệu

Chia dữ liệu thành 2 phần : Training data và Test Data với tỉ lệ 70/30.

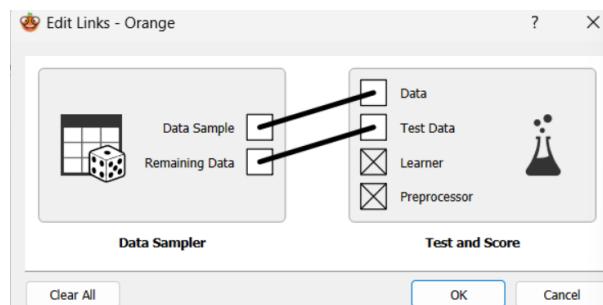


Hình 2.33: Caption

(d) Training và test mô hình Sử dụng widget Test and score để đánh giá mô hình

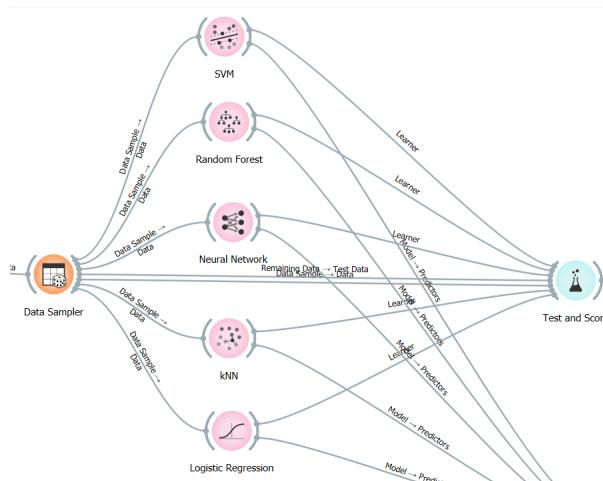
Kết nối output Data Sample của Data Sampler với Data của Test and Score

Kết nối Remaining Data của Data Sampler với Test Data của Test and Score



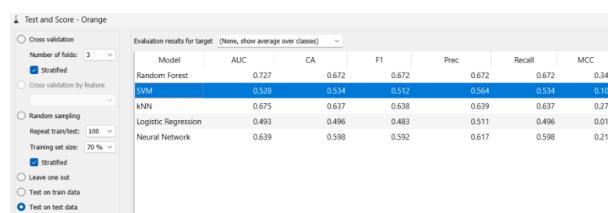
Hình 2.34: Kết nối Data Sampler với Test and Score

Kết nối các mô hình với Test and Score để đánh giá



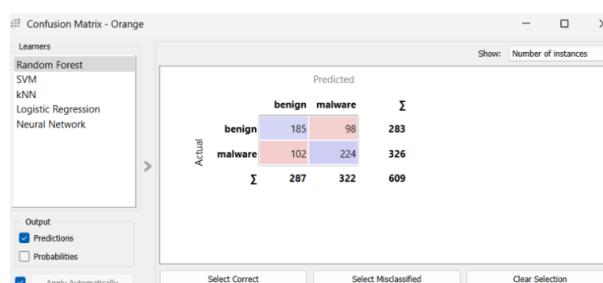
Hình 2.35: Kết nối các mô hình với Test and Score

Lựa chọn Test on test data



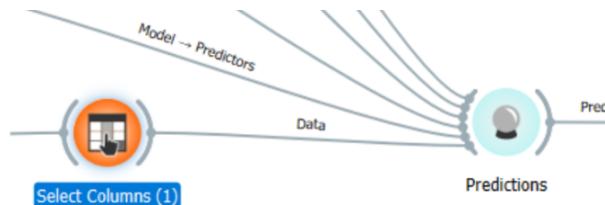
Hình 2.36: Lựa chọn Test on test data

(e) Kết nối Test and Score với Confusion Matrix để xem chi tiết kết quả đánh giá



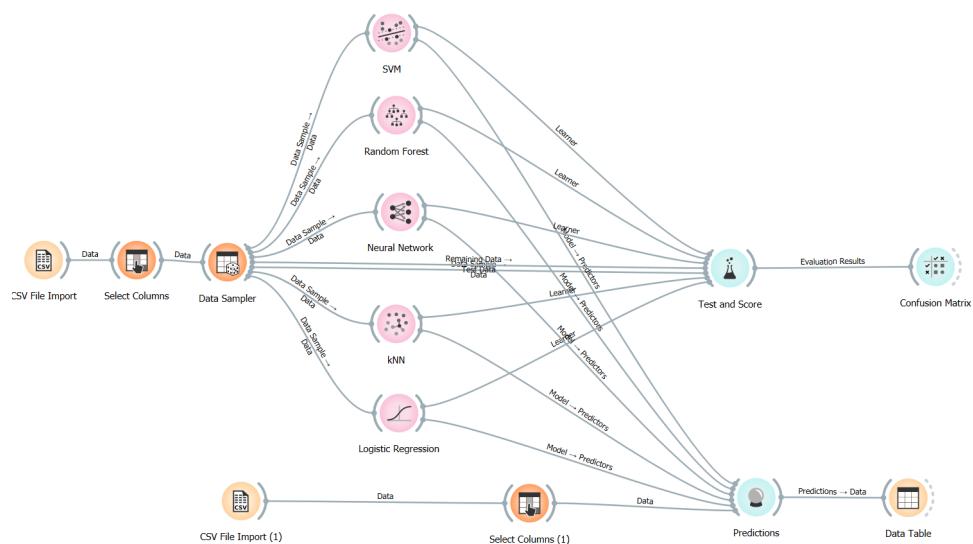
Hình 2.37: Lựa chọn Test on test data

- (f) Nhập dữ liệu cần dự đoán bằng CSV File Import
- (g) Sử dụng Select Columns để chọn ra các thuộc tính cho việc dự đoán
- (h) Kết nối dữ liệu vào Predictions



Hình 2.38: Kết nối dữ liệu vào Predictions

- (i) Kết nối dữ liệu từ Data Sampler vào mô hình và từ mô hình đến Predictions



Hình 2.39: Kết nối mô hình với Predictions

- (j) Kết nối output của Predictions với Data Table để xem dữ liệu dự đoán



Hình 2.40: Kết nối output của Predictions với Data Table để xem dữ liệu dự đoán

Kết quả

Model	AUC	CA	F1	Prec	Recall	MCC
Random Forest	0.754	0.693	0.693	0.693	0.693	0.384
SVM	0.529	0.534	0.512	0.564	0.534	0.108
kNN	0.675	0.637	0.638	0.639	0.637	0.274
Logistic Regression	0.493	0.496	0.483	0.511	0.496	0.015
Neural Network	0.639	0.598	0.592	0.617	0.598	0.219

Hình 2.41: Kết quả đánh giá mô hình

Đây là một bảng kết quả từ công cụ Orange Data Mining, hiển thị các thông tin về các mẫu dữ liệu và kết quả phân loại của các mô hình.

Các cột chính bao gồm:

- Selected: Chỉ định các dòng dữ liệu được chọn.
- shv256, sha1, md5, pkg_name: Các chỉ số xác thực cho ứng dụng.
- Logistic Regression, kNN, Random Forest, SVM, Neural Network: Các mô hình đã được huấn luyện.
- apk_size, dex_size, vercode: Các đặc trưng số lượng lớn.
- apk_size, dex_size, vercode: Các đặc trưng số lượng lớn.
- apk_size, dex_size, vercode: Các đặc trưng số lượng lớn.

Trong bảng, có một số dòng đặc biệt:

- Đông 1: Đầu tiên là dòng tiêu đề.
- Đông 2: Đầu tiên là dòng tiêu đề.
- Đông 3: Đầu tiên là dòng tiêu đề.
- Đông 4: Đầu tiên là dòng tiêu đề.
- Đông 5: Đầu tiên là dòng tiêu đề.
- Đông 6: Đầu tiên là dòng tiêu đề.
- Đông 7: Đầu tiên là dòng tiêu đề.
- Đông 8: Đầu tiên là dòng tiêu đề.
- Đông 9: Đầu tiên là dòng tiêu đề.
- Đông 10: Đầu tiên là dòng tiêu đề.
- Đông 11: Đầu tiên là dòng tiêu đề.
- Đông 12: Đầu tiên là dòng tiêu đề.
- Đông 13: Đầu tiên là dòng tiêu đề.
- Đông 14: Đầu tiên là dòng tiêu đề.
- Đông 15: Đầu tiên là dòng tiêu đề.
- Đông 16: Đầu tiên là dòng tiêu đề.
- Đông 17: Đầu tiên là dòng tiêu đề.
- Đông 18: Đầu tiên là dòng tiêu đề.
- Đông 19: Đầu tiên là dòng tiêu đề.
- Đông 20: Đầu tiên là dòng tiêu đề.
- Đông 21: Đầu tiên là dòng tiêu đề.
- Đông 22: Đầu tiên là dòng tiêu đề.
- Đông 23: Đầu tiên là dòng tiêu đề.
- Đông 24: Đầu tiên là dòng tiêu đề.
- Đông 25: Đầu tiên là dòng tiêu đề.
- Đông 26: Đầu tiên là dòng tiêu đề.
- Đông 27: Đầu tiên là dòng tiêu đề.
- Đông 28: Đầu tiên là dòng tiêu đề.
- Đông 29: Đầu tiên là dòng tiêu đề.
- Đông 30: Đầu tiên là dòng tiêu đề.
- Đông 31: Đầu tiên là dòng tiêu đề.
- Đông 32: Đầu tiên là dòng tiêu đề.
- Đông 33: Đầu tiên là dòng tiêu đề.
- Đông 34: Đầu tiên là dòng tiêu đề.
- Đông 35: Đầu tiên là dòng tiêu đề.
- Đông 36: Đầu tiên là dòng tiêu đề.

Hình 2.42: Kết quả dự đoán

Nhận xét :

- Kết quả đánh giá tất cả các mô hình đều cho kết quả thấp, với các chỉ số AUC chỉ nằm trong khoảng 0.493 - 0.754, Recall chỉ nằm trong khoảng 0.496 - 0.693. Các chỉ số này là rất thấp và mô hình không đáng tin cậy
- Vì thế nên tìm cách tiếp cận khác, bằng cách dịch ngược file APK, biểu diễn dưới dạng đồ thị và phân tích các đặc trưng để có thể phân loại chính xác hơn.

2. Thực hiện nội dung 2

3. Thực hiện nội dung 3

=> Kết luận:

Tuần Topic 5 (20/10 - 26/10)

Chủ đề tìm hiểu tuần 5: Các thuật toán phân loại và phân cụm sử dụng Orange data mining

Mục tiêu tuần Topic 5: Sử dụng Orange data mining để phân cụm dữ liệu dựa trên các đặc trưng như apk_size, vt_detection, dexsize và vercode. Trong đó :

- apk_size: Tổng kích thước của tệp APK tính theo byte, bao gồm mã nguồn của ứng dụng, tài nguyên, tệp tài sản, và bất kỳ tệp bổ sung nào được đóng gói vào APK.
- dex_size: Kích thước của tệp classes.dex, đây là tệp Dalvik Executable (DEX) chính chứa bytecode Java đã được biên dịch, chạy trên hệ điều hành Android. Trường này chỉ tính kích thước của tệp DEX đầu tiên

(classes.dex), không bao gồm các tệp DEX bổ sung.

- vercode: Mã phiên bản được xác định trong tệp manifest của ứng dụng Android. Đây là một giá trị số nguyên đại diện cho phiên bản của APK, được tăng dần mỗi khi ứng dụng được cập nhật. Nó có thể không khớp với chuỗi phiên bản hiển thị (ví dụ: “1.2.3”).
- vt_detection: Số lượng công cụ diệt virus (AV) trên VirusTotal đã gắn cờ APK này là phần mềm độc hại. VirusTotal tổng hợp kết quả từ nhiều công cụ diệt virus để đánh giá xem tệp có độc hại hay an toàn. Trong lần thử nghiệm này, sẽ thử tập dữ liệu không có đặc trưng vt_detection vì để có vt_detection thì phải dùng ứng khác quét

Dữ liệu huấn luyện : 2031 bản ghi, đã được gắn nhãn benign/ malware

Kết quả của tuần Topic 5:

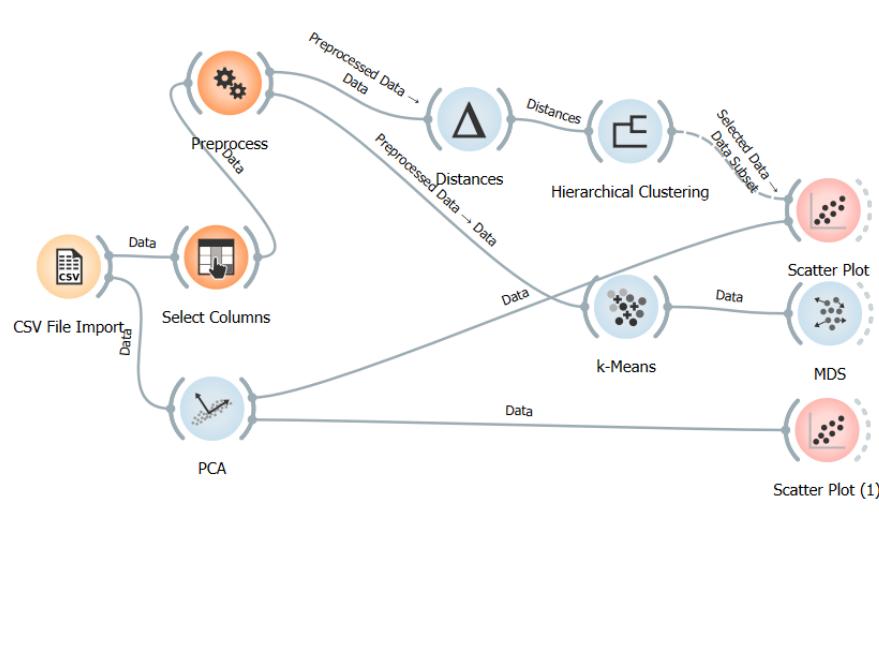
Thành viên tham gia

- Đỗ Đăng Khoa

1. Giới thiệu thuật toán phân cụm:

Là một kỹ thuật thuộc học không giám sát (unsupervised learning) trong machine learning. Mục tiêu chính của phân cụm là nhóm các đối tượng (dữ liệu) vào các cụm (cluster) sao cho những đối tượng trong cùng một cụm có sự tương đồng cao với nhau, trong khi các đối tượng thuộc các cụm khác nhau có sự khác biệt đáng kể. Mục tiêu là áp dụng các thuật toán phân cụm lên dữ liệu để phân các file ra các cụm nhằm xác định các file này có chứa mã độc hay không.

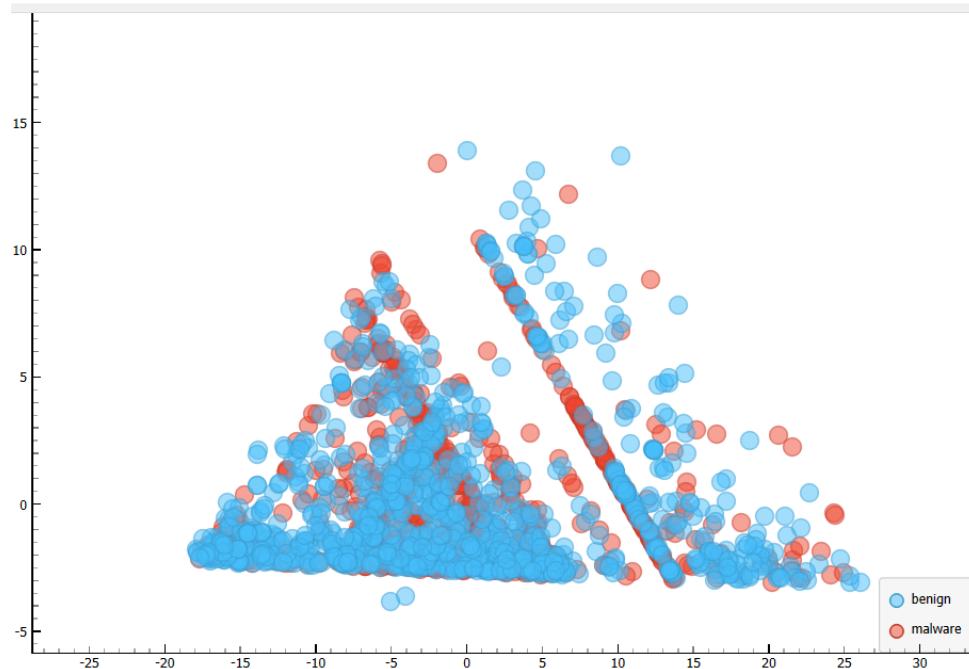
2. Mô hình Orange data mining



Hình 2.43: Mô hình tổng quan

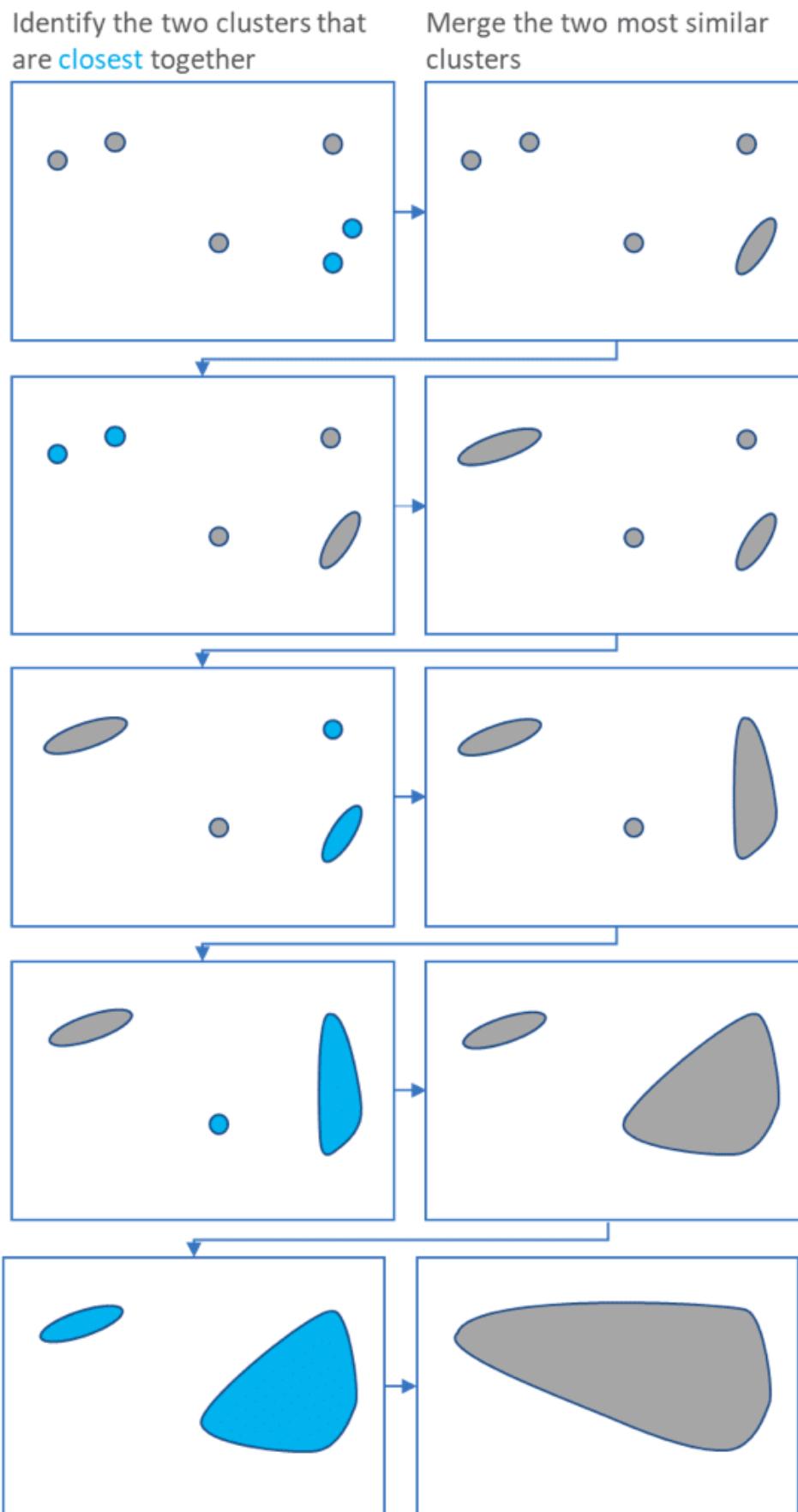
- Principal component analysis (PCA): Là một thuật toán giảm chiều dữ liệu được sử dụng phổ biến. Do dữ liệu gốc gồm nhiều chiều nên ta áp dụng thuật toán này đưa dữ liệu về dạng 2 chiều nhằm thuận tiện

cho việc hình dung trên scatter plot



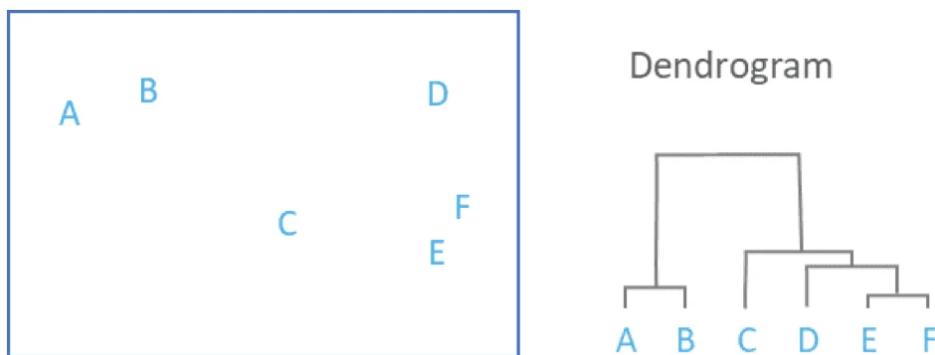
Hình 2.44: Kết quả của PCA

- (b) Hierarchical clustering: Là một thuật toán phân cụm theo cấp độ từ một ma trận khoảng cách (distance matrix). Bắt đầu với mỗi điểm dữ liệu là một cụm, thuật toán bao gồm 2 bước: (1) xác định 2 cụm gần nhau nhất, (2) hợp 2 cụm này. Quá trình này lặp lại cho đến khi chỉ còn một cụm duy nhất.



Hình 2.45: Ví dụ về Hierarchical clustering

Sau quá trình này ta thu được biểu đồ cho thấy cấp bậc của các điểm dữ liệu.

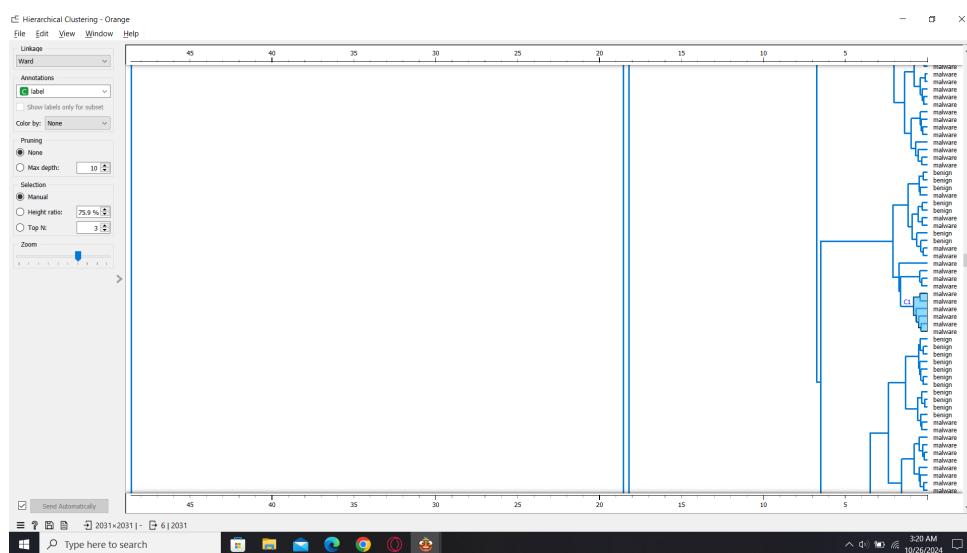


Hình 2.46: Cấp bậc trong dữ liệu

Trong Orange data mining, ta có thể áp dụng thuật toán này sau khi có được ma trận khoảng cách của dữ liệu.

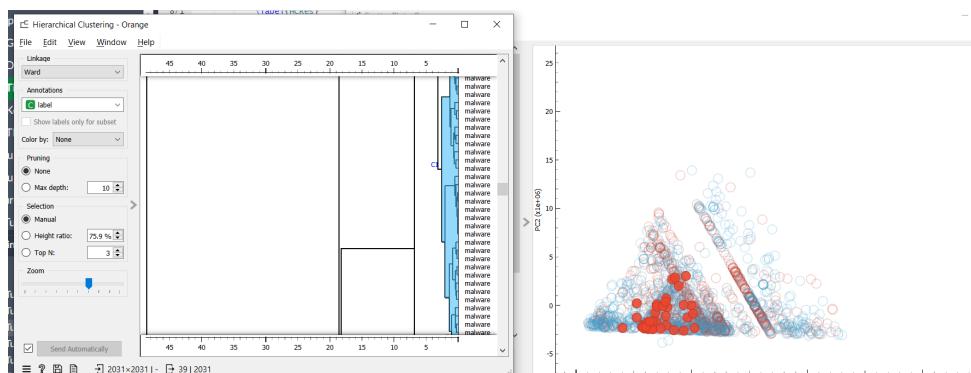


Hình 2.47: Hierarchical clustering trong Orange data mining



Hình 2.48: Kết quả của Hierarchical clustering

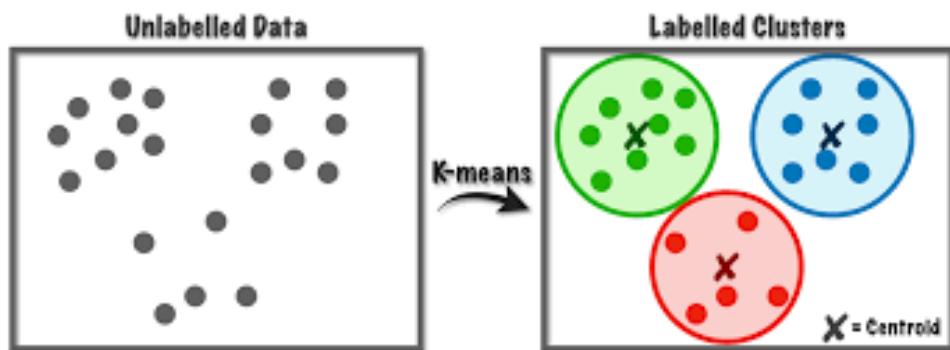
Lựa chọn các nhóm trên Hierarchical clustering, ta có thể xem được vị trí các điểm này trên scatter plot và đánh giá quá trình phân cụm dựa theo label của dữ liệu.



Hình 2.49: Hierarchical clustering trên scatterplot

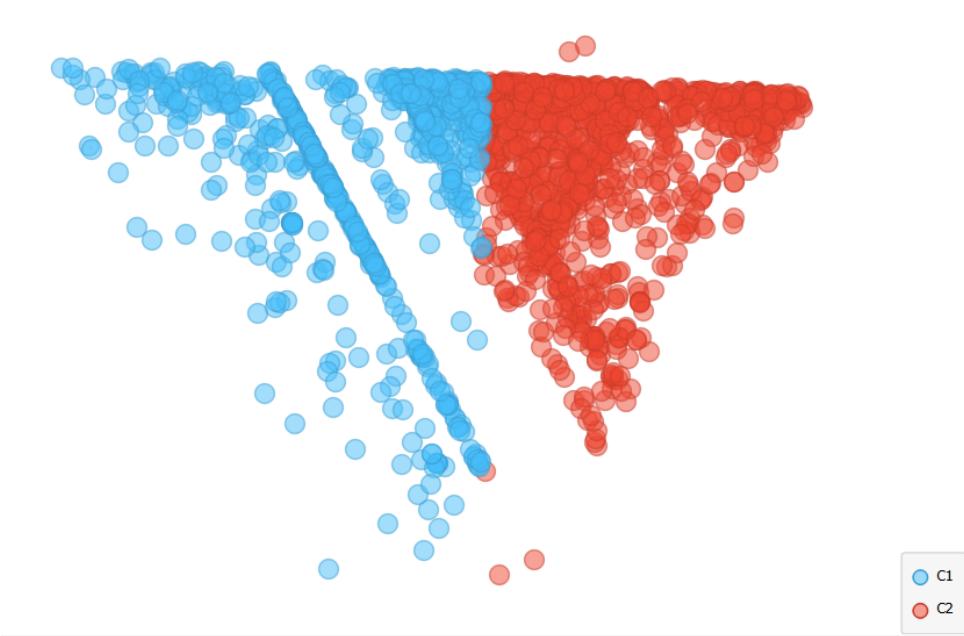
(c) K-means clustering: K-means clustering là một trong những thuật toán unsupervised learning cơ bản nhất được sử dụng rộng rãi trong các bài toán phân cụm với dữ liệu không có nhãn. Thuật toán này gồm 3 bước:

- i. Khởi tạo n điểm ngẫu nhiên làm tâm của cụm
- ii. Phân loại các điểm theo tâm gần nhất và cập nhật lại tọa độ của tâm là giá trị trung bình của các điểm hiện tại thuộc cụm tương ứng
- iii. Lặp lại quá trình trên cho đến khi ra kết quả



Hình 2.50: K-means

Trong orange data mining đã cài đặt sẵn thuật toán K-means, kết quả có thể sử dụng MDS để hình dung trên mặt phẳng 2D. Ta lựa chọn số cụm là 2 tương ứng với số nhãn của dữ liệu



Hình 2.51: Kết quả của K-mean trên dữ liệu apk

→ Kết luận: Các thuật toán phân cụm trên đã hoạt động trên tập dữ liệu apk, tuy nhiên vẫn còn nhiều điểm hạn chế. Khi hình dung dữ liệu sử dụng PCA, ta không quan sát được sự phân cụm rõ ràng của dữ liệu. Điều này cho thấy các thuật toán phân cụm không phù hợp khi áp dụng với thông số file từ csv. Để phân loại dữ liệu apk có thể cần đến thông tin từ nội dung file (không chứa trong csv) và áp dụng các thuật toán phức tạp hơn.

Tuần Topic 6 (27/10 - 02/11)

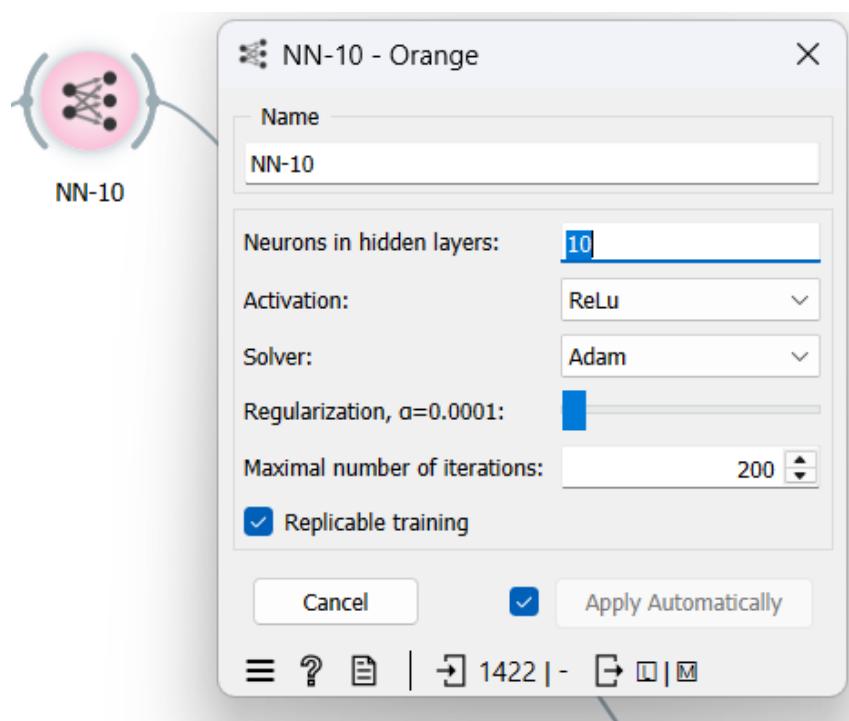
Chủ đề tìm hiểu tuần 6: Đánh giá hiệu suất của các cấu hình khác nhau khi áp dụng mạng nơ-ron (NN) trên tập dữ liệu.

Mục tiêu tuần Topic 6: Thực hiện triển khai mô hình mạng nơ-ron (NN) trên tập dữ liệu, thử nghiệm với nhiều cấu hình khác nhau để đánh giá hiệu suất và khả năng dự đoán của mô hình.

Thành viên tham gia: Vũ Thành Tuyên

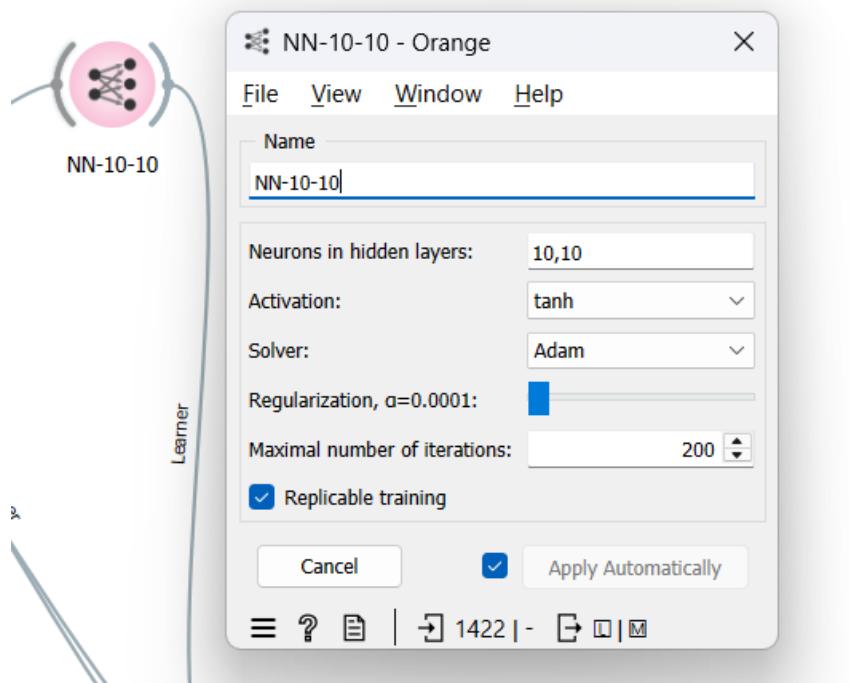
Kết quả của tuần Topic 6:

1. Thực hiện đánh giá hiệu quả của mô hình khi thay đổi số lượng lớp, với hàm kích hoạt ReLU và bộ tối ưu Adam.
 - (a) Với số lớp là 1, số đơn vị của lớp là 10



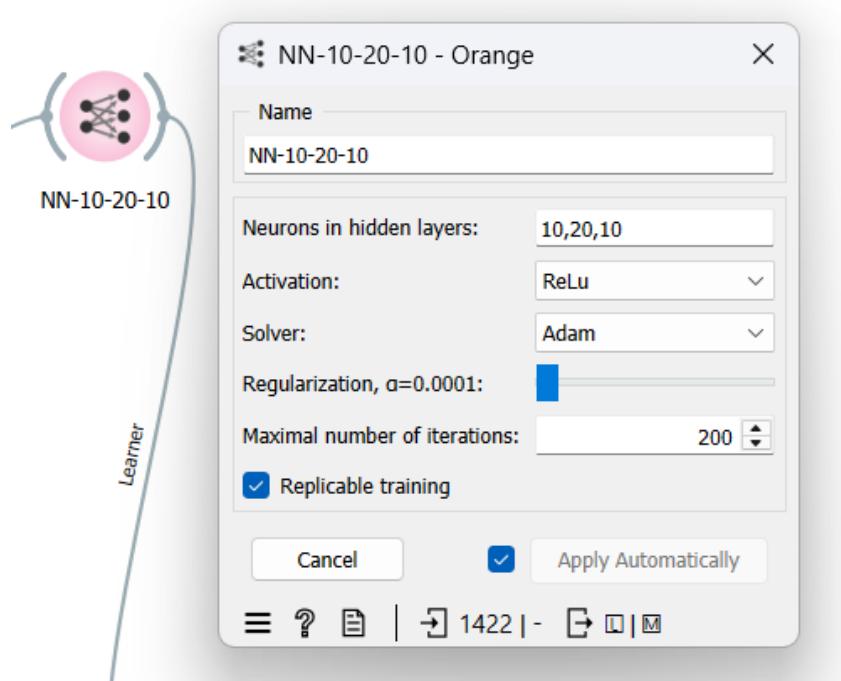
Hình 2.52: NN - 10

(b) Với số lớp là 2, số đơn vị của lớp là 10, 10



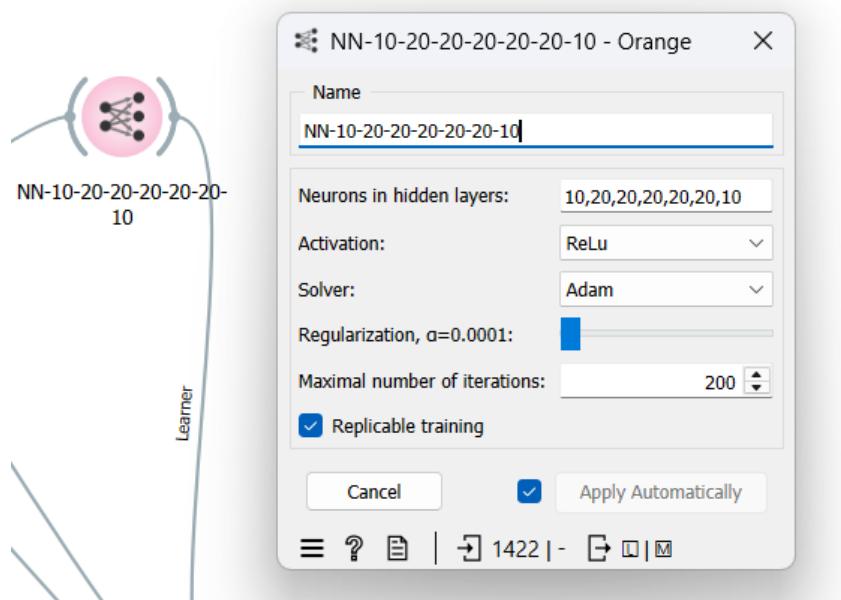
Hình 2.53: NN - 10, 10

(c) Với số lớp là 3, số đơn vị của lớp là 10, 20, 10



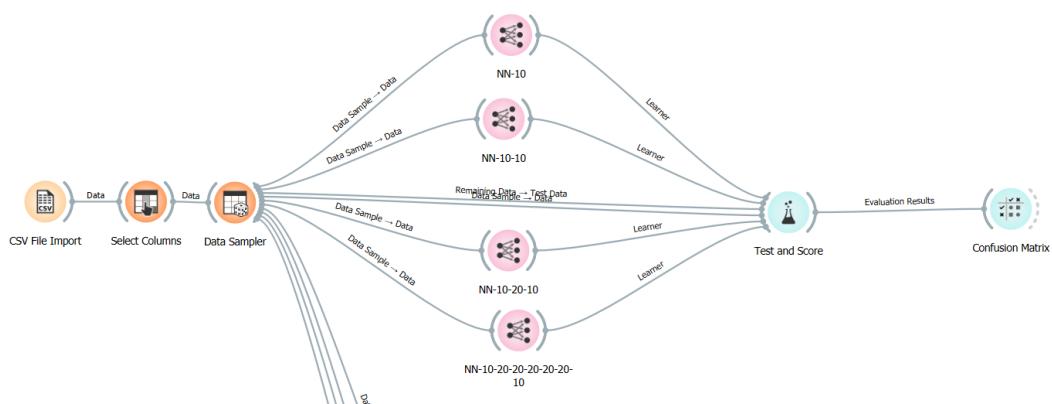
Hình 2.54: NN - 10, 20, 10

(d) Với số lớp là 7, số đơn vị của lớp là 10, 20, 20, 20, 20, 20, 10



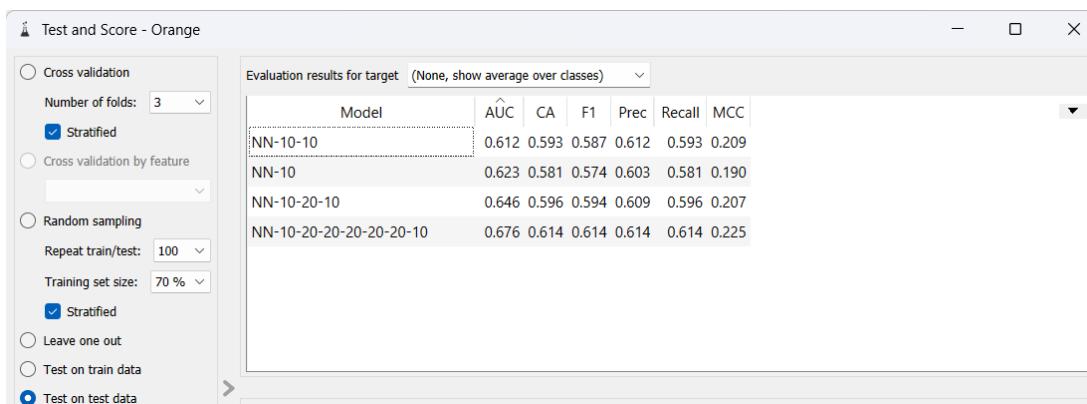
Hình 2.55: NN - 10, 20, 20, 20, 20, 20, 10

(e) Sơ đồ minh họa các kiến trúc mô hình được áp dụng trên cùng một tập dữ liệu



Hình 2.56: Sơ đồ minh họa cho thí nghiệm số lớp khác nhau

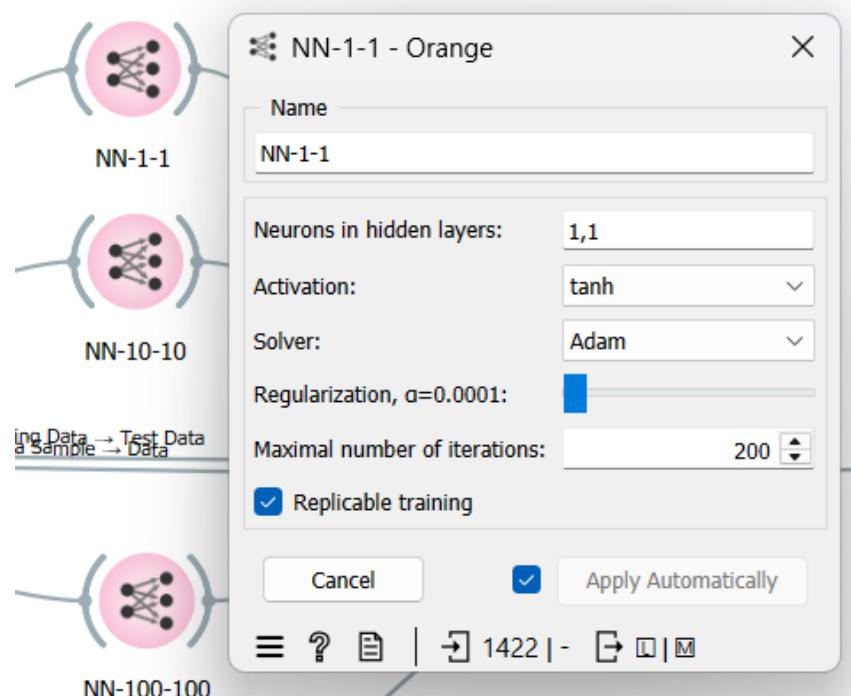
(f) Kết quả của các mô hình trên



Hình 2.57: Kết quả thí nghiệm số lớp khác nhau

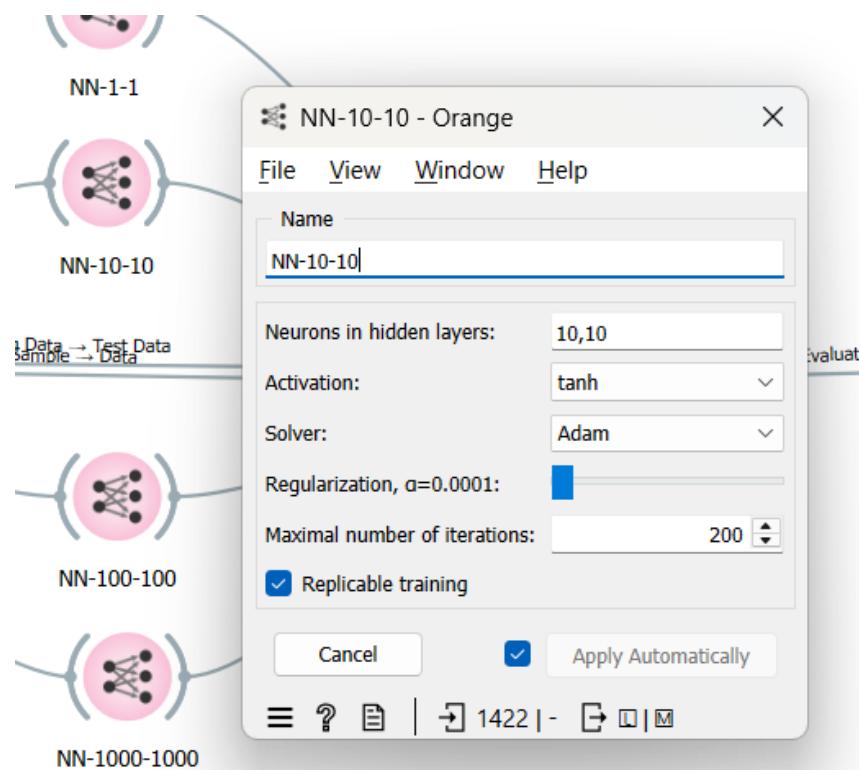
(g) Kết luận:

- Nhìn vào bảng kết quả trên, có thể nhận thấy rằng khi số lượng layer và số lượng neuron trong mỗi layer tăng lên, hiệu suất của mô hình cũng có xu hướng cải thiện theo. Các chỉ số đánh giá như AUC, CA (độ chính xác), F1, độ chính xác (Prec), Recall và MCC đều tăng dần khi mô hình có cấu trúc phức tạp hơn.
 - Cụ thể, mô hình với kiến trúc NN-10-20-20-20-20-20-10 cho kết quả tốt nhất trên các chỉ số đánh giá so với các kiến trúc đơn giản hơn. Điều này cho thấy việc sử dụng thêm nhiều layer có thể giúp mạng nơ-ron học tốt hơn từ dữ liệu, nhưng cũng đồng nghĩa với việc mô hình sẽ phức tạp hơn và có nguy cơ bị overfitting nếu không được điều chỉnh cẩn thận.
2. Đánh giá hiệu suất mô hình với các số lượng đơn vị (units) khác nhau trong khi giữ nguyên số lượng lớp (layers), với hàm kích hoạt ReLU và bộ tối ưu Adam, xét với 2 layer
- (a) Xét với số unit là 1



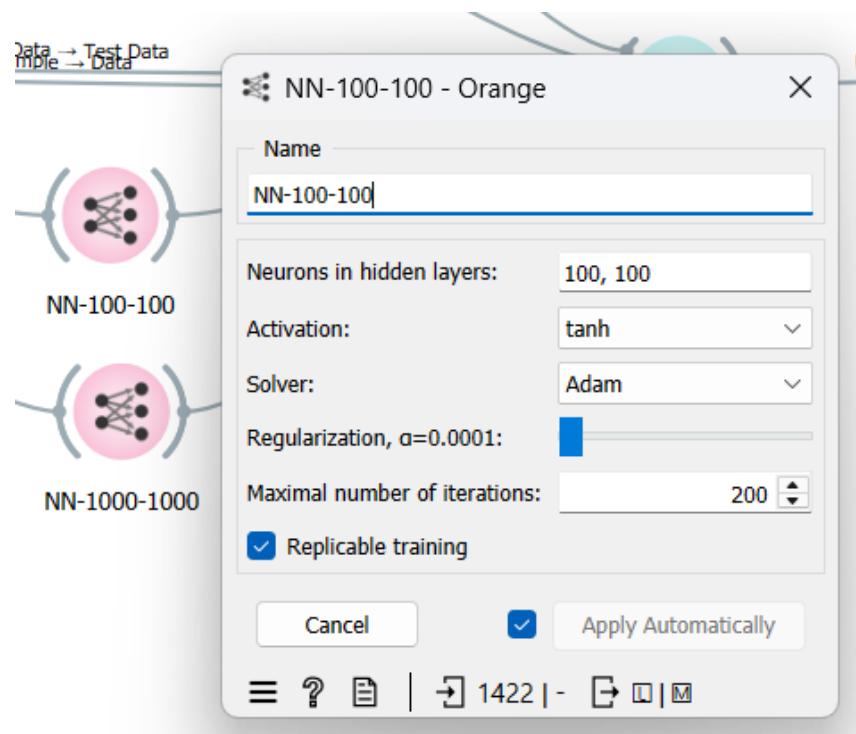
Hình 2.58: NN với số unit 1

(b) Xét với số unit là 10



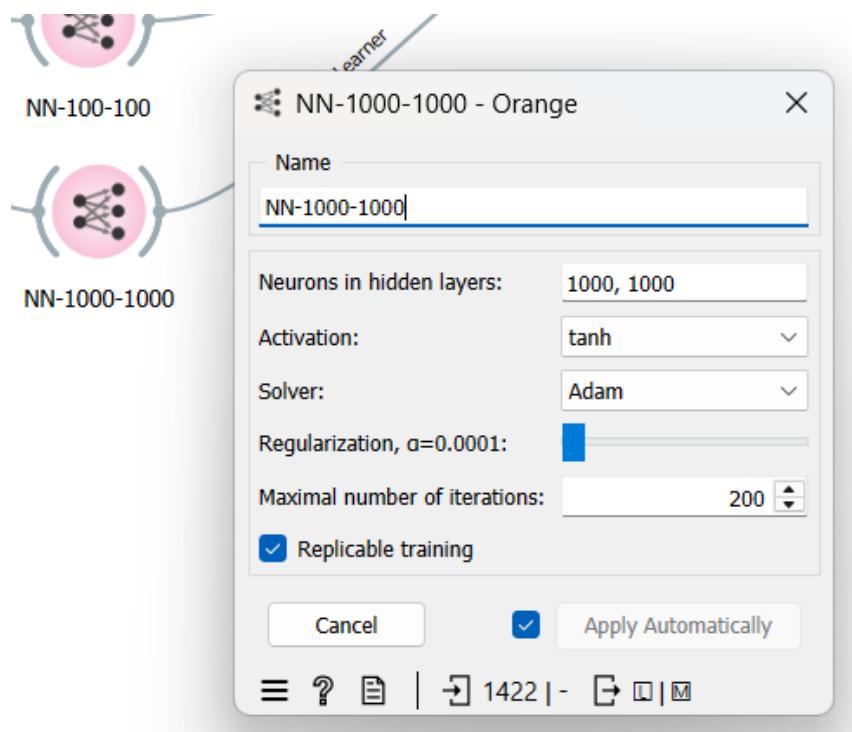
Hình 2.59: NN với số unit 10

(c) Xét với số unit là 100



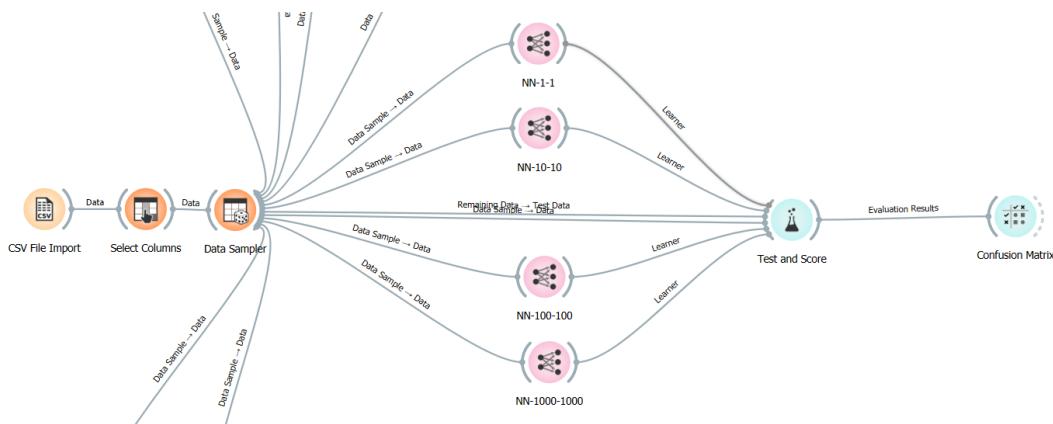
Hình 2.60: NN với số unit 100

(d) Xét với số unit là 1000



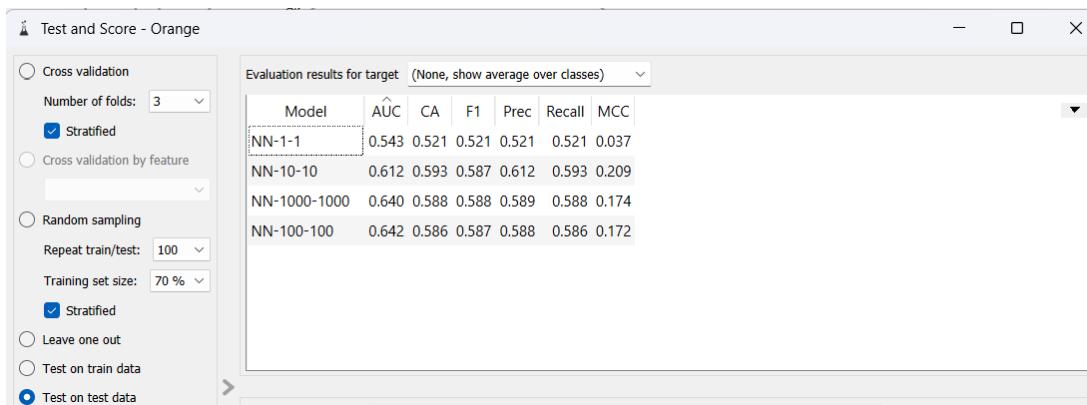
Hình 2.61: NN với số unit 1000

(e) Sơ đồ minh họa các kiến trúc mô hình được áp dụng trên cùng một tập dữ liệu



Hình 2.62: Sơ đồ minh họa cho thí nghiệm số lớp khác nhau

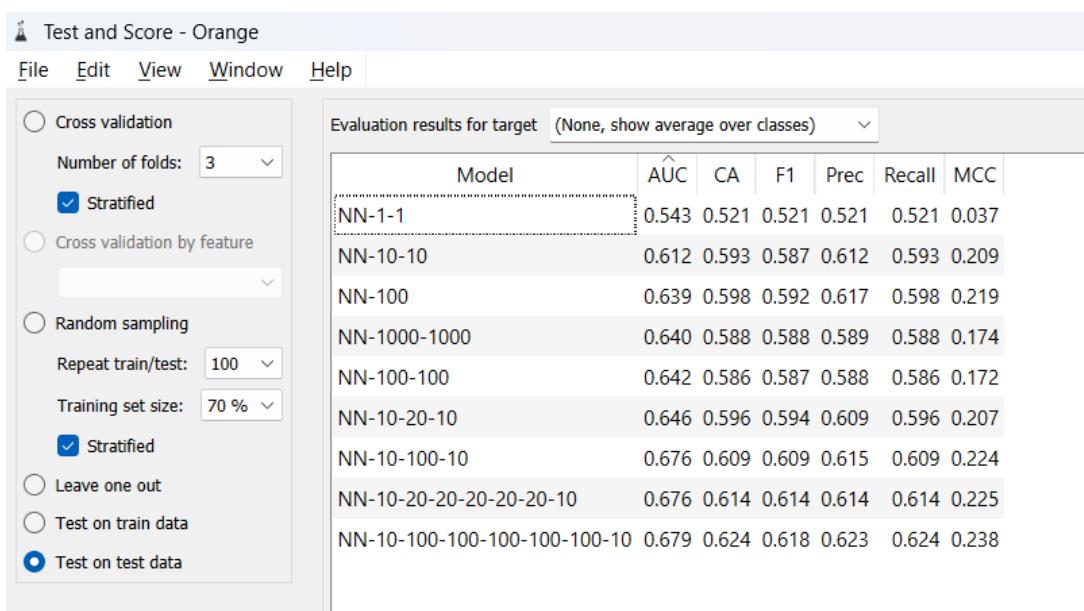
(f) Kết quả của các mô hình trên



Hình 2.63: Kết quả thí nghiệm số unit khác nhau

(g) Kết luận:

- Số lượng unit rất ít (ví dụ: NN-1-1): Mô hình với số lượng unit cực thấp dẫn đến các chỉ số đánh giá như AUC, CA, F1 và MCC đều thấp. Điều này cho thấy mô hình không đủ khả năng nắm bắt các mẫu phức tạp trong dữ liệu do quá ít tham số.
 - Số lượng unit vừa phải (ví dụ: NN-10-10): Khi tăng số lượng unit lên 10 cho mỗi layer, hiệu suất của mô hình cải thiện rõ rệt. Các chỉ số đều tăng đáng kể, cho thấy mô hình có khả năng học tốt hơn từ dữ liệu khi số unit vừa phải.
 - Số lượng unit lớn (ví dụ: NN-100-100 và NN-1000-1000): Khi tăng số lượng unit lên nhiều hơn nữa, hiệu suất của mô hình không tăng đáng kể so với cấu hình NN-10-10. Điều này có thể là do mô hình đạt đến ngưỡng bão hòa, khi số lượng unit thêm vào không mang lại cải thiện đáng kể trong việc học từ dữ liệu, đồng thời còn làm tăng nguy cơ overfitting và độ phức tạp của mô hình.
3. Kết hợp tối ưu giữa số lượng lớp và số lượng đơn vị (units) phù hợp.
- (a) Kết quả của sự kết hợp tương đối giữa số layer và số unit



Hình 2.64: Kết quả thí nghiệm

(b) Kết luận:

- Các mô hình với số lượng unit và layer thấp như NN-1-1 hoặc NN-10-10 có chỉ số AUC và độ chính xác (CA) khá thấp. Điều này cho thấy rằng khi mô hình có quá ít unit hoặc layer, khả năng học hỏi và phân biệt của nó bị hạn chế, dẫn đến hiệu quả dự đoán thấp.
- Khi số lượng unit được tăng lên (như NN-100, NN-100-100), các chỉ số AUC, CA, và F1 tăng lên, cho thấy mô hình có khả năng học và phân tích dữ liệu tốt hơn. Tuy nhiên, hiệu quả vẫn chưa đạt mức cao nhất có thể, cho thấy cần có thêm các lớp (layer) hoặc số unit lớn hơn để cải thiện.
- Khi tăng thêm số lượng layer hoặc unit ở các lớp ẩn (như mô hình NN-1000-1000 hoặc NN-10-100-100-10), các chỉ số về độ chính xác, AUC, và MCC được cải thiện đáng kể. Điều này cho thấy rằng mô hình có khả năng phân tích dữ liệu phức tạp hơn khi có cấu trúc nhiều lớp và nhiều unit hơn.
- Các mô hình kết hợp nhiều lớp với số unit khác nhau trong mỗi lớp (như NN-10-20-20-20-20-10) mang lại kết quả tương đối tốt, với các chỉ số khá cao. Điều này gợi ý rằng việc sử dụng các lớp với số unit khác nhau có thể tạo ra mô hình có khả năng học linh hoạt và hiệu quả hơn so với mô hình chỉ sử dụng một số unit cố định cho mỗi lớp.

→ Kết luận: Tóm lại, mô hình càng phức tạp (nhiều layer và nhiều unit hơn) thường có hiệu quả cao hơn, nhưng việc tối ưu hóa số lượng layer và unit cần được cân nhắc để đạt hiệu quả tốt nhất mà không bị quá khớp (overfitting).

Tuần Topic 7 (03/11 - 10/11)

Chủ đề tìm hiểu tuần 7: Thực hiện doc2vec cho dữ liệu java được tạo từ apk và công cụ dịch ngược apk thành mã nguồn(java)

Thành viên tham gia:

- Đào Duy Thông

- Vũ Thành Tuyên **Mục tiêu tuần Topic 7:** Trình bày khái quát về lý thuyết, cách áp dụng doc2vec vào dữ liệu ở

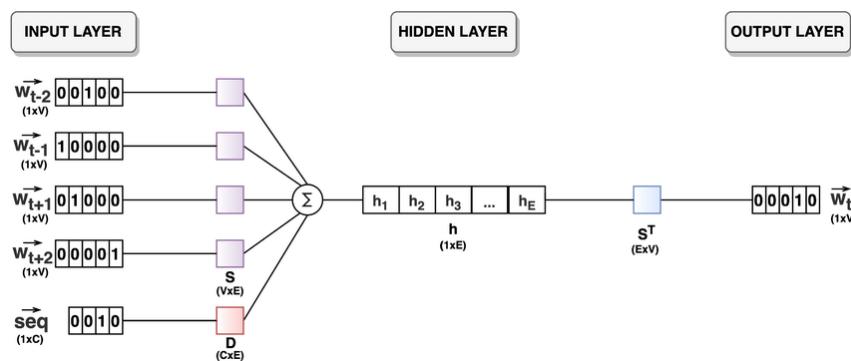
dạng java (do được tạo ra từ các file apk)

1. Tìm hiểu về lý thuyết doc2vec
2. Lý do áp dụng kĩ thuật doc2vec vào bài toán phân loại apk
3. Cách sử dụng kĩ thuật doc2vec vào bài toán phân loại apk
4. Áp dụng doc2vec vào dự án

Kết quả của tuần Topic:

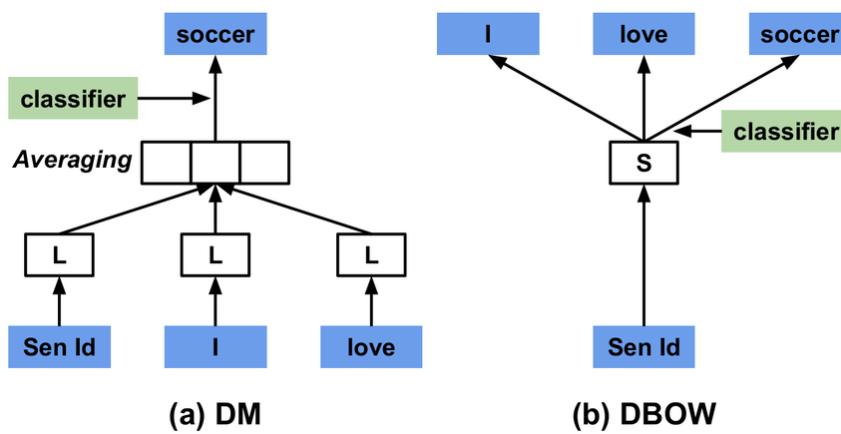
1. Tìm hiểu về lý thuyết doc2vec

- (a) Mục tiêu của Doc2Vec là học các biểu diễn vector cho các đoạn văn, câu, hoặc tài liệu, thay vì chỉ học vector cho các từ đơn lẻ như trong Word2Vec. Bằng cách này, Doc2Vec không chỉ nắm bắt ý nghĩa của từng từ mà còn hiểu được ngữ cảnh tổng thể của một đoạn văn hoặc tài liệu, từ đó giúp mô hình có khả năng biểu diễn ngữ nghĩa của các văn bản dài.



Hình 2.65: Architecture Doc2vec

- (b) Có hai kiến trúc chính trong Doc2Vec là Distributed Memory (DM) và Distributed Bag of Words (DBOW). Với DM, mô hình học cách dự đoán từ tiếp theo trong một câu dựa trên các từ trước đó và vector của tài liệu, kết hợp cả thông tin về ngữ cảnh và cấu trúc của văn bản. Điều này giúp vector biểu diễn của tài liệu giữ được ngữ cảnh mà các từ xuất hiện. Ngược lại, DBOW hoạt động tương tự như mô hình Skip-gram của Word2Vec, khi nó học cách dự đoán các từ ngẫu nhiên trong tài liệu mà không sử dụng thông tin ngữ cảnh. Dù cách tiếp cận khác nhau, cả hai đều nhằm mục đích tạo ra một biểu diễn vector cho toàn bộ tài liệu.
- (c) Doc2Vec đã được sử dụng rộng rãi trong nhiều ứng dụng, từ phân loại văn bản, tìm kiếm thông tin, đến phân tích cảm xúc và khuyến nghị. Nó giúp tạo ra các biểu diễn vector có khả năng nắm bắt ngữ nghĩa và mối quan hệ giữa các tài liệu, từ đó giúp các mô hình học máy có thể dễ dàng xử lý văn bản hơn. Một trong những lợi ích chính của Doc2Vec là khả năng làm việc với các văn bản có độ dài và ngữ cảnh khác nhau, điều mà Word2Vec không thể làm được. Điều này làm cho Doc2Vec trở thành một công cụ mạnh mẽ trong xử lý ngôn ngữ tự nhiên.



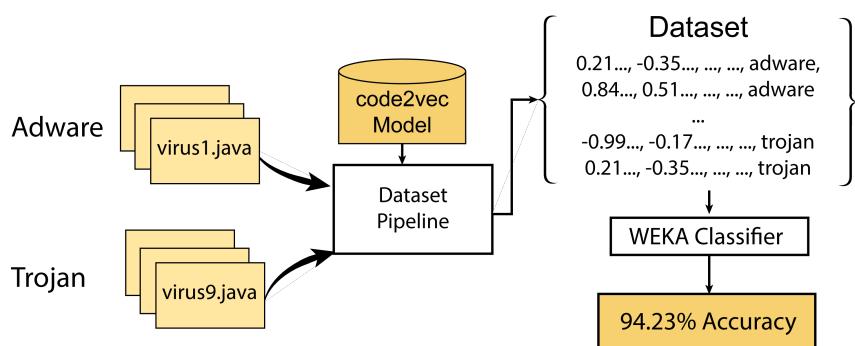
Hình 2.66: DM and DBOW

2. **Lý do áp dụng kỹ thuật doc2vec vào bài toán phân loại apk:** khi mà phân tích file apk ta nhận thấy ta có thể dịch ngược file apk thành file java, và từ đó ta có thể làm giàu đồ thị hơn với các đặc trưng xuất từ file ngôn ngữ java

- (a) Doc2Vec vào ngôn ngữ Java, chúng ta có thể xây dựng các ứng dụng phân tích văn bản và xử lý ngôn ngữ tự nhiên (NLP) mạnh mẽ. Doc2Vec giúp chuyển các đoạn văn bản, tài liệu, hoặc đoạn mã nguồn thành các vector có kích thước cố định, biểu diễn ngữ nghĩa của nội dung. Điều này có thể áp dụng cho nhiều bài toán liên quan đến mã nguồn trong Java, chẳng hạn như phân loại, tìm kiếm và gợi ý mã, phát hiện lỗi, và phân tích mã nguồn.
- (b) Doc2Vec mang lại nhiều lợi ích khi áp dụng vào việc phân tích và xử lý mã nguồn Java. Nó giúp biểu diễn mã nguồn dưới dạng các vector số học, cho phép các mô hình học máy hiểu được ngữ nghĩa của các đoạn mã, từ đó hỗ trợ phân loại các file apk

3. **Cách sử dụng kỹ thuật doc2vec vào bài toán phân loại apk:** Kỹ thuật Doc2Vec có thể được sử dụng để phân loại các tệp APK (Android Package) bằng cách chuyển đổi mã nguồn java của APK thành các biểu diễn vector, sau đó sử dụng các vector này để huấn luyện mô hình phân loại.

- (a) APK chứa mã nguồn (thường là mã Java hoặc mã Smali)
- (b) Mã nguồn: Mã nguồn Java hoặc Smali cần được chuyển thành dạng văn bản, tương tự như một tài liệu để có thể áp dụng Doc2Vec. Có thể coi mỗi đoạn mã (class, method) là một "tài liệu".
- (c) Sau khi có thể chuyển đổi thành các file java thì ta có thể nhúng các đoạn văn bản đó cho một mô hình đã được đào tạo từ trước trên huggingface để thu được các vector đặc trưng cho file apk của ta



Hình 2.67: Embedding java with doc2vec

4. Áp dụng doc2vec vào dự án:

(a) Thực hiện sử dụng kiến trúc đã được xây sẵn trong package **gensim**

(b) Thực hiện extract các hàm từ file java đã được dịch ngược từ trước đó:

```

1 def extract_functions_from_java_file(file_path):
2     """Function to extract functions from a Java file"""
3     with open(file_path, 'r') as file:
4         lines = file.readlines()
5
6     functions = []
7     current_function = []
8     inside_function = False
9     brace_count = 0
10
11    for line in lines:
12        stripped_line = line.strip()
13
14        # Skip lines that define the class
15        if stripped_line.startswith('class '):
16            continue
17
18        # Check if the line starts with "public", "private", "protected",
19        # or "static"
20        if stripped_line.startswith(('public ', 'private ', 'protected ', 'static ')) and ' ' in stripped_line:
21            if current_function:
22                functions.append(''.join(current_function))
23                current_function = []
24            inside_function = True
25
26            # Add line to the current function
27            if inside_function:
28                current_function.append(line)
  
```

```

28         brace_count += line.count('{')
29         brace_count -= line.count('}')
30
31     # If brace_count is 0, the function has ended
32     if brace_count == 0:
33         functions.append(''.join(current_function))
34         current_function = []
35         inside_function = False
36
37     return functions

```

- (c) Sau khi mà ta đã thực hiện xong việc trích xuất các hàm từ các file java, ta tiến hành mang vào training model doc2vec

```

1 def train_doc2vec_model(tokenized_functions):
2     """Function to train a Doc2Vec model and vectorize functions"""
3     documents = [TaggedDocument(words=tokens, tags=[f"function_{i}"]) for i
4         , tokens in enumerate(tokenized_functions)]
5     model = Doc2Vec(vector_size=100, window=5, min_count=1, workers=4,
6         epochs=50)
7
8     # Build vocabulary
9     model.build_vocab(documents)
10
11    # Train the model
12    model.train(documents, total_examples=model.corpus_count, epochs=model.
13        epochs)
14
15    return model

```

- (d) Model được training xong ta mang vào thực hiện nhúng từng hàm một mà ta trích xuất từ java ở trên qua mô hình doc2vec ta thu vector của một hàm là 200

```

1 # Step 3: load model doc2vec
2 from gensim.models.doc2vec import Doc2Vec
3 model = Doc2Vec.load("/kaggle/input/doc2vec/other/default/1/
4     java_8m_methods_doc2vec.model")
5
6 # Java file
7 # Find all .java files in the directory
8 java_files = []
9 for root, _, files in os.walk(folder_path):
10     for file in files:
11         if file.endswith('.java'):
12             java_files.append(os.path.join(root, file))

```

```

12
13 # Step 4: Save vectors to .txt files with corresponding names
14 for idx, file_path in enumerate(java_files):
15     print("Idx:", idx)
16     funcs = docs[idx]
17     embs = []
18     for func in funcs:
19         emb = model.infer_vector(func)
20         embs.append(emb.tolist())
21     output_file = os.path.join('/kaggle/working', f"{os.path.splitext(os.
22                                     path.basename(file_path))[0]}.txt")
23     save_vectors_to_file(embs, output_file)

```

-> Kết luận:

1. Log khi nhúng các hàm trong file java qua doc2vec:

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** Doc2vecMalware
- Description:** Copied from Đào Duy Thông (+1, 1)
- Logs Tab:** Selected tab.
- Log Content:**

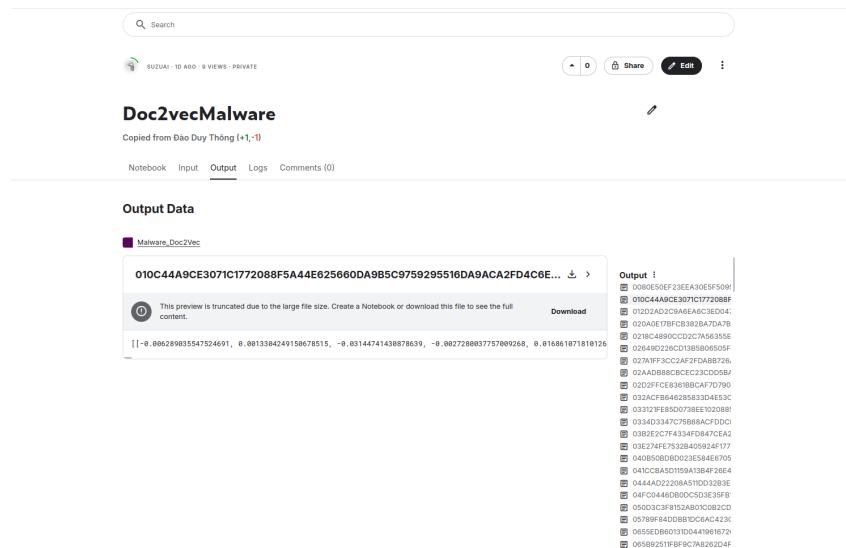
```

Successfully ran in 826.0s
[Time] [#] [Log Message]
20.2s 1 Dang xu ly: /kaggle/input/malwarejava/984906CA73E15CB37B42F0F21BF096888F2C731268B079024C389A48E3295F7.java
20.2s 2 Dang xu ly: /kaggle/input/malwarejava/6B8A7551F6A0B13F9030F8A880CF840919FD7F84B6A97701116051093F0500.java
20.2s 3 Dang xu ly: /kaggle/input/malwarejava/51EC5C181C5F58B06FF6811BC362021176F34CE5509863D11613C489839D1088A.java
20.2s 4 Dang xu ly: /kaggle/input/malwarejava/7F6899A3E8F6F9090E94785778B440116A2B0D7A66FFAAEFA4B1B1D9F314E36.java
20.2s 5 Dang xu ly: /kaggle/input/malwarejava/48873116393EAD0CD512E2A042A69753D08836711BC579FA14F147D530B97841C.java
20.2s 6 Dang xu ly: /kaggle/input/malwarejava/1C80DC41E579CA6524224180773D99F9C980BF2E1FA130C490C8BEAD68989050F8.java
20.2s 7 Dang xu ly: /kaggle/input/malwarejava/8C850C902E824F47E7C9C9088E6E8938479B6014245A5A8970F43C7035C9C989840.java
20.4s 8 Dang xu ly: /kaggle/input/malwarejava/7EE00E80278E70C5DF4A312E2B98611A82F80973EF9E0DC3914F43599BA820CF.java
20.4s 9 Dang xu ly: /kaggle/input/malwarejava/1EFC9C9CC2E25434804F17A73F77511EA9F5E7753A1B0C0F677CE2F1177901.java
20.4s 10 Dang xu ly: /kaggle/input/malwarejava/AD6910E503721ED424387491A57F119197893177F78966847D0FA164420.java
20.7s 11 Dang xu ly: /kaggle/input/malwarejava/2FF7898558F773400CF7E193129821CE558B781CE692FF5FE3948B14E64FF5.java
20.7s 12 Dang xu ly: /kaggle/input/malwarejava/12B9537E801625185EFD463A59163381D485188AC7C7E0D8AC243236984011.java
20.7s 13 Dang xu ly: /kaggle/input/malwarejava/42C9DCT2B84618AC489WF80CE87774CF6E619198E4242470CF49AE74221C008.java
20.7s 14 Dang xu ly: /kaggle/input/malwarejava/585D4671E9950A7FB510A86468B0164C7F9882C8BF78EC8A6BD4A49E11E05224.java

```
- Accelerator:** None
- Environment:** Latest Container Image
- Output:** 5.3 GB

Hình 2.68: Log Embedding java qua doc2vec

2. Dữ liệu sau khi nhúng:



Hình 2.69: Kết quả Embedding java qua doc2vec

Tuần Topic 8 (10/11 - 16/11)

Chủ đề tìm hiểu tuần topic 8: Thực hiện thiết kế và triển khai CSDL noSQL trên tập dữ liệu. Viết query và thống kê đối với dữ liệu MongoDB

Thành viên tham gia:

- Nguyễn Việt Khiêm
- Trần Sỹ Tiên

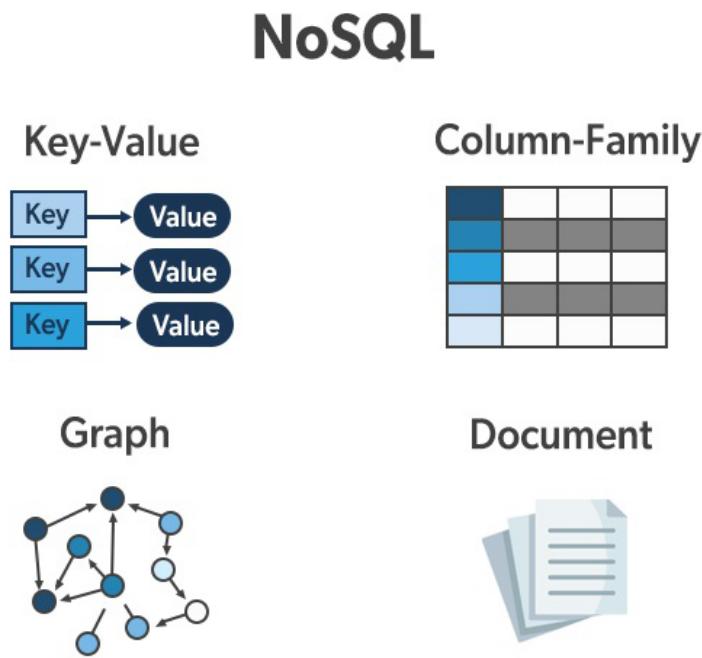
Mục tiêu tuần Topic 8: Trình bày khái quát về cách tổ chức dữ liệu thô - xử lý, query và thống kê đối với CSDL noSQL MongoDB.

1. Tìm hiểu về cơ sở dữ liệu noSQL. Tìm hiểu về MongoDB.
2. Cấu trúc tổ chức dữ liệu
3. Query
4. Thống kê dữ liệu

Kết quả của tuần Topic 8:

1. Tìm hiểu về cơ sở dữ liệu noSQL. Tìm hiểu về MongoDB.

- (a) Cơ sở dữ liệu NoSQL (Not Only SQL) là một loại cơ sở dữ liệu được thiết kế để xử lý dữ liệu phi cấu trúc hoặc bán cấu trúc, thường không yêu cầu sử dụng mô hình quan hệ (bảng và cột) như cơ sở dữ liệu SQL. NoSQL phù hợp với các ứng dụng yêu cầu khả năng mở rộng cao, xử lý lượng lớn dữ liệu hoặc có cấu trúc dữ liệu không cố định.



Hình 2.70: noSQL database

(b) Các đặc điểm chính của noSQL:

- Linh hoạt: Không cần định nghĩa trước cấu trúc dữ liệu (schema-less).
- Hiệu suất cao: Tối ưu cho các truy vấn nhanh và khôi lượng lớn dữ liệu.
- Khả năng mở rộng: Hỗ trợ mở rộng ngang (thêm nhiều máy chủ).
- Hỗ trợ nhiều loại dữ liệu: Văn bản, JSON, XML, đồ thị, hoặc dữ liệu nhị phân.

(c) Phân loại chính:

- Key-Value Store: Lưu trữ dữ liệu dưới dạng cặp key-value (VD: Redis, DynamoDB).
- Document Store: Lưu trữ dữ liệu dưới dạng tài liệu (JSON, BSON) (VD: MongoDB, Couchbase).
- Column-Family Store: Dữ liệu được lưu thành cột (VD: Cassandra, HBase).
- Graph Database: Lưu trữ dữ liệu dưới dạng các nút và quan hệ (VD: Neo4j, ArangoDB).

(d) Trong báo cáo này, nhóm 2 sử dụng CSDL MongoDB. MongoDB là một cơ sở dữ liệu NoSQL phổ biến, sử dụng mô hình lưu trữ document-based (tài liệu). Dữ liệu được lưu trữ dưới dạng các tài liệu JSON hoặc BSON (Binary JSON). MongoDB phù hợp với các ứng dụng yêu cầu linh hoạt về cấu trúc dữ liệu, xử lý dữ liệu lớn, và hiệu năng cao.

(e) Đặc điểm chính của MongoDB:

- Schema-less: Không yêu cầu cấu trúc cố định cho dữ liệu (schema) và các tài liệu trong cùng một collection có thể có cấu trúc khác nhau.
- Mô hình tài liệu (Document Model): Dữ liệu được tổ chức dưới dạng tài liệu (documents) với các trường (fields), tương tự như một đối tượng JSON. Một tài liệu đại diện cho một bản ghi, và một

collection chứa nhiều tài liệu tương tự như một bảng trong SQL.

- iii. Hỗ trợ mở rộng ngang (Horizontal Scaling): MongoDB sử dụng sharding để phân phối dữ liệu trên nhiều máy chủ, đảm bảo khả năng mở rộng khi dữ liệu tăng lên.
- iv. Truy vấn linh hoạt: Hỗ trợ nhiều kiểu truy vấn, từ truy vấn cơ bản đến truy vấn phức tạp như lọc, sắp xếp, và kết hợp dữ liệu.
- v. Chỉ mục (Indexing): Hỗ trợ nhiều loại chỉ mục để tăng hiệu năng truy vấn, bao gồm chỉ mục đơn, chỉ mục phức hợp, chỉ mục văn bản.
- vi. Khả năng tích hợp cao: Tích hợp tốt với các ngôn ngữ lập trình như Python, Java, NodeJS, C#, và nhiều công nghệ hiện đại khác.

(f) Kiến trúc MongoDB:

- i. Database: Là nơi chứa các collection.
- ii. Collection: Tập hợp các tài liệu tương tự nhau, tương đương với một bảng trong SQL.
- iii. Document: Đơn vị cơ bản của MongoDB, được lưu dưới dạng JSON hoặc BSON.

```

1 {
2   "_id": "12345",
3   "name": "Nguyen Van A",
4   "age": 25,
5   "address": { "city": "Hanoi", "country": "Vietnam" }
6 }
```

(g) Ưu điểm của MongoDB:

- i. Linh hoạt trong cấu trúc dữ liệu.
- ii. Xử lý dữ liệu lớn với hiệu năng cao.
- iii. Dễ dàng mở rộng khi dữ liệu tăng lên.
- iv. Tích hợp tốt với các ứng dụng thời gian thực.

2. Cấu trúc tổ chức dữ liệu. Vì MongoDB lưu trữ dữ liệu theo các Collection (Tập hợp các tài liệu tương tự nhau, tương đương với một bảng trong SQL) được chứa trong database nên đây là các Collection nhóm 2 sẽ sử dụng để lưu trữ dữ liệu.

```

_id: ObjectId('6737678f8147701d0abb2743')
file_path : "https://www.kaggle.com/code/oduythng/doc2vecbenign-63c85c/EEDC5DD59556..."
size : "13415.65 kB"
date : "2024-11-15 11:23:29"

_id: ObjectId('673767908147701d0abb2744')
file_path : "https://www.kaggle.com/code/oduythng/doc2vecbenign-63c85c/24404571E1FC..."
size : "337.32 kB"
date : "2024-11-15 11:23:21"
```

Hình 2.71: Bên trong một Collection

Đối với collection ví dụ:

- (a) _id: ID của object
- (b) file_path: Đường dẫn đến file được lưu trữ trong Kaggle
- (c) size: Kích thước của file
- (d) date: Thời điểm xuất hiện file



Hình 2.72: Collections

- (a) Benign-apk (file)

1	6DF079464232EFF4E530F3D8F8FB8D2E5212E0F19DBC6D546F8093C2417E61B3.apk 1005.42 2024-11-15 11:31:22
2	0F885F09F01D2F857CFE08EF57D29A1E0ADBB7B8DB5936A9A5A8A54C03361950.apk 105.13 2024-11-15 11:31:22
3	CCC923788C620F15DD6A1ABE7B1A3EE631E722DA2FA69BE3A9D348A112218A5E.apk 158.22 2024-11-15 11:31:21

- (b) Benign-doc2vec

1	EEDC5DD595566F1E1F47C1888F05329819C2F27EEBD326981ACF3EE0649D5B76.txt 13415.65 2024-11-15 11:23:29
---	--

```

2 24404571E1FC6575A7DFC5BAEC78DACP74B873578DFCA3D290211DD3286CF91A.txt 337.32
    2024-11-15 11:23:21
3 3F6784B906C71CAA116BEDD87B912B752301467443587960C8AC17508FF889EF.txt
    1751.14 2024-11-15 11:23:23

```

(c) Benign-fcg

```

1 3810A835FA906DC35167A2970A18E5C77AA812C805703C59196EFD9F7289D3F5.fcg
    1070.60 2024-11-15 11:31:25
2 FD3156606D5B2244A9D485343785DDA3D2CA015218253EEE201319D56066D68C.fcg 5.66
    2024-11-15 11:31:25
3 4A7718AA41541DD7DE983E042A5D66EFD6B83311EC27F99AE6A5449FC284254D.fcg 95.54
    2024-11-15 11:31:25

```

(d) Benign-java

```

1 8B7F4AB46AA4EFAFCE49AEE91E48518E529DC4FDF73A1FC6EC8E37FAC90C985A.java
    558.99 2024-11-15 11:24:32
2 3249D58BBD8A8D4A583C8680CBF6F0A6DCA4C53EC97130CAD766DA0111351236.java 35.60
    2024-11-15 11:24:29
3 B83F94B9EE28E2C24DF6D8D1CE84713A2D9C3C074A889AEBC8AC8BD22F4AF8F6.java
    718.77 2024-11-15 11:24:34

```

3. Query

- Python cung cấp thư viện MongoClient cho phép truy vấn vào CSDL MongoDB thông qua các câu lệnh
- Code config connection MongoDB và các lệnh đọc ghi vào CSDL. Trong module này, nhóm sẽ truy xuất các thông tin trong malware-doc2vec.txt (sha, size, date) từ file text rồi insert thông tin đó vào MongoDB:

```

1 from pymongo import MongoClient
2 import os
3 from datetime import datetime
4
5 client = MongoClient("mongodb+srv://laohacbacho20032003:IgsOT049EKsoVVUC@khdl.
6     xez2i.mongodb.net/?retryWrites=true&w=majority&appName=KHDL")
7 database = client["KHDL"]
8 collection = database["Malware_Doc2Vec"]
9
10 collection.delete_many({})
11
12 file_names = []
13 with open('./malware-doc2vec.txt', 'r') as file:
14     file_names = file.readlines()
15
16 base_url = "https://www.kaggle.com/code/suzuai/doc2vecmalware/"
17 for line in file_names:

```

```

18     parts = line.strip().split()

19

20     file_name = parts[0]
21     file_size = (parts[1]) + " kB"
22     creation_time = " ".join(parts[2:])

23

24     full_path = f"{base_url}{file_name}"

25

26     # Insert into MongoDB
27     collection.insert_one({
28         "file_path": full_path,
29         "size": file_size,
30         "date": creation_time
31     })

```

- Query code ví dụ:

```

1 from pymongo import MongoClient
2 import json
3
4 uri = "mongodb+srv://laojacbacho20032003:1gsOT049EKsoVVUC@khdl.xez2i.mongodb.
5           net/?retryWrites=true&w=majority&appName=KHDL"
6
7 client = MongoClient(uri)
8
9 database = client['KHDL'] # Thay 'KHDL' bằng tên của
10          # bạn muốn
11 collection = database['Benign_APK'] # Tạo collection trong MongoDB
12
13 def convert_object_id(data):
14     for doc in data:
15         if "_id" in doc:
16             doc["_id"] = str(doc["_id"])
17     return data
18
19 def write_to_file(filename, data):
20     with open(filename, 'w', encoding='utf-8') as f:
21         json.dump(data, f, ensure_ascii=False, indent=4)
22
23 # 1.
24 file_name = "BC0F5D0D6B795D4E063DECE177D2CCDD0D0ADC27386D7EBFE46749F2578D78DC.
25 apk"
26 query1 = {"file_path": {"$regex": file_name}}
27 result1 = convert_object_id(list(collection.find(query1)))
28 write_to_file('query1.txt', result1)

```

```
26
27 # 2.
28 query2 = {"size": {"$gt": "150 kB"}}
29 result2 = convert_object_id(list(collection.find(query2)))
30 write_to_file('query2.txt', result2)
31
32 # 3.
33 query3 = {"date": {"$regex": "^2024-11-15"}}
34 result3 = convert_object_id(list(collection.find(query3)))
35 write_to_file('query3.txt', result3)
36
37 # 4.
38 query = { "size": { "$gte": "100 kB", "$lte": "200 kB" } }
39 result = list(collection.find(query))
40 write_to_file('query4.txt', convert_object_id(result))
41
42 # 5.
43 result5 = convert_object_id(list(collection.find().sort("size", -1)))
44 write_to_file('query5.txt', result5)
45
46 client.close()
```

- Kết quả:

```
1 # 1.
2 [
3 {
4     "_id": "6737663e8f408e82c7882de0",
5     "file_path": "https://www.kaggle.com/code/oduythng/craw-apk-androizoo/
6                 BC0F5D0D6B795D4E063DECE177D2CCDD0D0ADC27386D7EBFE46749F2578D78DC.
7                 apk",
8     "size": "873.23 kB",
9     "date": "2024-11-15 11:31:24"
10 }
11 ]
12 # 2.
13 [
14 {
15     "_id": "6737661a8f408e82c78829fe",
16     "file_path": "https://www.kaggle.com/code/oduythng/craw-apk-androizoo/
17                 CCC923788C620F15DD6A1ABE7B1A3EE631E722DA2FA69BE3A9D348A112218A5E.
18                 apk",
19     "size": "158.22 kB",
20     "date": "2024-11-15 11:31:21"
21 },
22 ]
```

```

18    {
19        "_id": "6737661a8f408e82c78829ff",
20        "file_path": "https://www.kaggle.com/code/oduythng/craw-apk-androizoo/4
21            B9BF66508D799291055438F627E8392E3A62EB2696D596AD55957F26216FA90.apk
22            ",
23        "size": "150.22 kB",
24        "date": "2024-11-15 11:31:22"
25    },
26    {
27        "_id": "6737661a8f408e82c7882a00",
28        "file_path": "https://www.kaggle.com/code/oduythng/craw-apk-androizoo/
29            C6ACABB9E733FE53E2A0A5C0C9AD45799D38A33C0CDF93D970CD4D7950C56B1C.
30            apk",
31        "size": "178.22 kB",
32        "date": "2024-11-15 11:31:24"
33    },
34 ]

```

4. Thống kê dữ liệu

- Code thống kê dữ liệu:

```

1 from pymongo import MongoClient
2 import json
3 from datetime import datetime
4
5 uri = "mongodb+srv://laozacbacho20032003:1gsOT049EKsoVVUC@khdl.xez2i.mongodb.
6             net/?retryWrites=true&w=majority&appName=KHDL"
7
8 client = MongoClient(uri)
9
10 database = client['KHDL']
11 collection = database['Benign_APK']
12
13 def convert_object_id(data):
14     for doc in data:
15         if "_id" in doc:
16             doc["_id"] = str(doc["_id"])
17
18     return data
19
20 def write_to_file(filename, data):
21     with open(filename, 'w', encoding='utf-8') as f:
22         json.dump(data, f, ensure_ascii=False, indent=4)
23
24 # 1.
25 count = collection.count_documents({})

```

```

24 write_to_file('thongke1.txt', [{"total_documents": count}])

25

26 # 2.

27 average_size = list(collection.aggregate([
28     { "$group": {
29         "average_size": { "$avg": "$size" }
30     } }
31 ]))

32 write_to_file('thongke2.txt', average_size)

33

34 # 3.

35 max_min_size = list(collection.aggregate([
36     { "$group": {
37         "max_size": { "$max": "$size" },
38         "min_size": { "$min": "$size" }
39     } }
40 ]))

41 write_to_file('thongke3.txt', max_min_size)

42

43 client.close()

```

- Kết quả thống kê:

```

1 #1 .
2 [
3     {
4         "total_documents": 1000
5     }
6 ]
7
8 #2 .
9 [
10    {
11        "average_size": 463.11818
12    }
13 ]
14
15 #3 .
16 [
17    {
18        "max_size": 1023.16,
19        "min_size": 5.55
20    }
21 ]

```

CHƯƠNG 3. BÁO CÁO DỰ ÁN

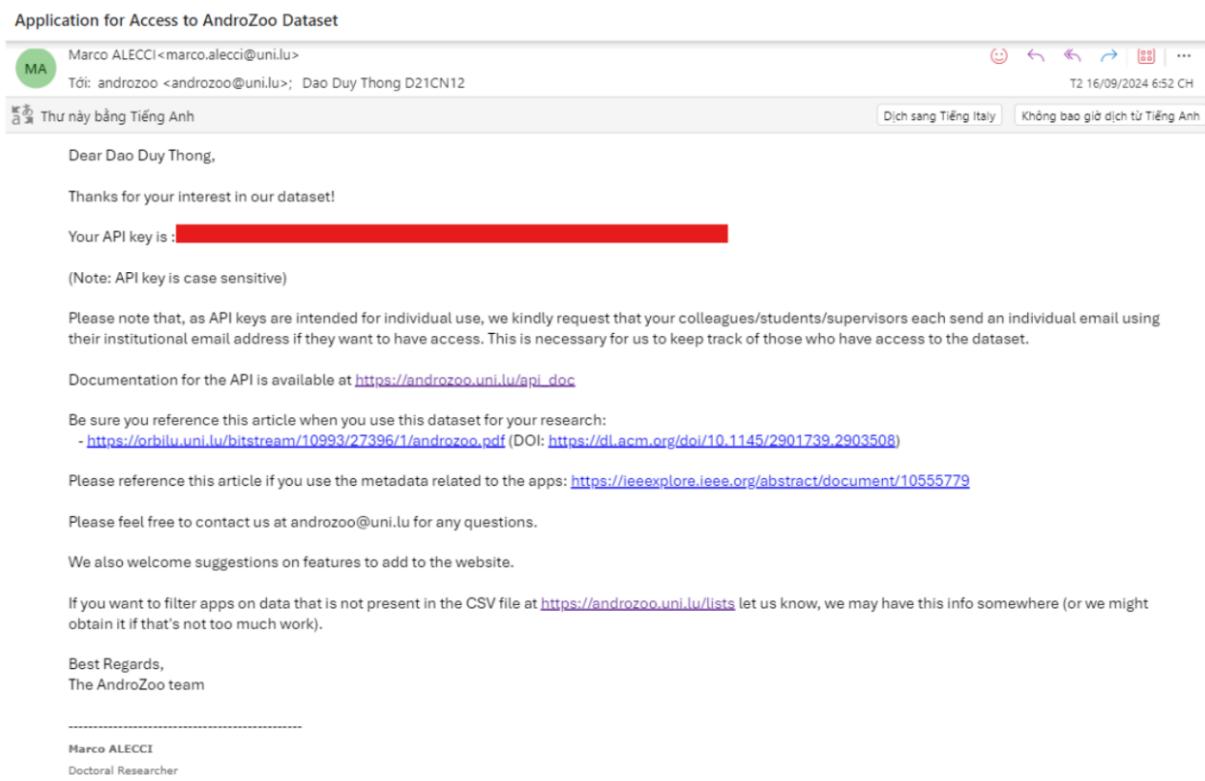
3.1 Thu thập dữ liệu

3.1.1 Nguồn dữ liệu

AndroZoo là một bộ sưu tập ứng dụng Android đang phát triển, được thu thập từ nhiều nguồn khác nhau, bao gồm cả cửa hàng ứng dụng chính thức Google Play. Hiện tại, nó chứa 24,718,475 tệp APK khác nhau, mỗi tệp đã được (hoặc sẽ sớm được) phân tích bởi nhiều sản phẩm phần mềm diệt virus để xác định ứng dụng nào bị phát hiện là phần mềm độc hại.

AndroZoo cung cấp API để người dùng có thể truy cập và tải xuống file APK: [https://androzoo.uni.lu/api/download?apikey=\\\${APIKEY}&sha256=\\\${SHA256}](https://androzoo.uni.lu/api/download?apikey=\${APIKEY}&sha256=\${SHA256}). Trong đó APIKEY là API Key của người dùng được cấp thông qua email, SHA256 là mã hash, mã này được lưu trữ trong 1 file CSV, mỗi hàng chứa thông tin của 1 file APK như : sha256 : mã hash để tải file APK, apksize : kích thước file APK, vt-tool-detection : số phần mềm virus phát hiện file APK là độc hại,...

Androzoo yêu cầu người dùng gửi email từ tài khoản.edu để lấy được API key. Sau khi email xin cấp API KEY, nhóm đã được phản hồi và cấp 1 API KEY để truy cập và tải xuống các file APK của Androzoo.



Hình 3.1: Email phản hồi của Androzoo

3.1.2 Tiền xử lý dữ liệu

Vì file csv có kích thước rất lớn, chứa rất nhiều mã hash để tải các file APK, cần lọc hoặc cắt chỉ lấy đủ số lượng file APK cần thiết cho quá trình làm dự án (khoảng 2000 file). Cụ thể :

- **Bước 1: Đọc tệp CSV**

- Sử dụng hàm `read_csv` của Pandas để đọc tệp `latest.csv` đã được giải nén trước đó.
 - Gán dữ liệu vào biến `df`.
- **Bước 2: Lọc bỏ hàng có giá trị 'snaggamea' trong cột 'pkg_name'**
- Lọc bỏ các hàng có chứa chuỗi 'snaggamea' trong cột 'pkg_name' bằng cách sử dụng toán tử phủ định ~ và hàm `str.contains()`.
 - Lưu kết quả vào biến `df_filtered`.
- **Bước 3: Chuyển đổi cột 'dex_date' sang định dạng datetime**
- Sử dụng hàm `pd.to_datetime()` để chuyển đổi giá trị của cột 'dex_date' sang định dạng ngày tháng.
 - Các giá trị không hợp lệ sẽ được xử lý thành NaT (bằng cách sử dụng `errors='coerce'`).
- **Bước 4: Lọc các hàng có giá trị 'apk_size' nhỏ hơn 1 MB**
- Lọc các hàng có giá trị của cột 'apk_size' nhỏ hơn 1048576 bytes (1 MB).
 - Lưu kết quả vào biến `df_filtered_size`.
- **Bước 5: Sắp xếp dữ liệu theo cột 'dex_date'**
- Sử dụng hàm `sort_values()` để sắp xếp dữ liệu theo cột 'dex_date' theo thứ tự giảm dần (các APK gần đây nhất sẽ ở trên cùng).
 - Lưu kết quả vào biến `df_sorted`.
- **Bước 6: Phân tách danh sách benign và malware dựa trên cột 'vt_detection'**
- Tạo danh sách các APK benign (*lành tính*) bằng cách lọc các hàng có giá trị 'vt_detection' bằng 0 và lấy 1000 hàng đầu tiên.
 - Tạo danh sách các APK malware (*có mã độc*) bằng cách lọc các hàng có giá trị 'vt_detection' lớn hơn 0 và lấy 1000 hàng đầu tiên.
 - Lưu kết quả vào biến `benign_list` và `malware_list`.
- **Bước 7: Lưu danh sách benign và malware vào tệp CSV riêng biệt**
- Sử dụng hàm `to_csv()` để lưu `benign_list` vào tệp `benign_list_1000.csv` và `malware_list` vào tệp `malware_list_1000.csv`.
 - Xác nhận việc lưu thành công bằng cách in đường dẫn lưu tệp ra màn hình.

Code xử lý file CSV

```

1 import pandas as pd
2
3 df = pd.read_csv('/kaggle/input/androzoolist/latest.csv')
4
5 df_filtered = df[~df['pkg_name'].str.contains('snaggamea', na=False)]
```

```

6
7     df_filtered['dex_date'] = pd.to_datetime(df_filtered['dex_date'], errors='coerce')
8
9     df_filtered_size = df_filtered[df_filtered['apk_size'] < 1048576]
10
11    df_sorted = df_filtered_size.sort_values(by='dex_date', ascending=False)
12
13    benign_list = df_sorted[df_sorted['vt_detection'] == 0].head(1000)
14    malware_list = df_sorted[df_sorted['vt_detection'] > 0].head(1000)
15
16    benign_file_path = '/kaggle/working/benign_list_1000.csv'
17    malware_file_path = '/kaggle/working/malware_list_1000.csv'
18
19    benign_list.to_csv(benign_file_path, index=False)
20    malware_list.to_csv(malware_file_path, index=False)
21
22    print("Benign list saved at:", benign_file_path)
23    print("Malware list saved at:", malware_file_path)

```

Sau bước này sẽ có 1 dataset gồm 2 file csv : benign_list_1000.csv và malware_list_1000.csv, mỗi file có 1000 dòng.

3.1.3 Tải dữ liệu từ Androizoo

Sử dụng API key và mã hash với cú pháp:

[https://androzoo.uni.lu/api/download?apikey=\\$APIKEY&sha256=\\$SHA256](https://androzoo.uni.lu/api/download?apikey=$APIKEY&sha256=$SHA256)

Duyệt qua các hàng trong file CSV, lấy mã sha256 và dùng request trong Python để tải file APK.

Do tổng dung lượng APK của benign và malware lớn hơn 19,5GB (dung lượng tối đa cho output của Kaggle), cần thực hiện tải xuống hai lần.

```

1   from concurrent.futures import ThreadPoolExecutor
2
3   import pandas as pd
4
5   import os
6
7   import requests
8
9
10
11
12
13
14

```

```

15     pathsave = "/kaggle/working/"

16

17     base_url = 'https://androzoo.uni.lu/api/download?apikey=6205
18                                     e414a89f470504a13bc12dfeecb11db866d927248f748213410cb168b6d2&sha256='

19

20     download_url = base_url + sha256_hash
21     response = requests.get(download_url, stream=True)

22

23     if response.status_code == 200:
24         file_name = pathsave + f"{sha256_hash}.apk"
25         with open(file_name, 'wb') as file:
26             for chunk in response.iter_content(chunk_size=8192):
27                 file.write(chunk)
28             print(f"File {cnt} - {file_name} ")
29     else:
30         print(f"Error {sha256_hash}: {response.status_code}")
31     data = pd.read_csv("/kaggle/input/d/suzuai/dataset-benign-malware/
32                         benign_list_100.csv")
33     hashList = data['sha256'].to_list()
34     hashList = [f"{{a}}#{{i}}" for i, a in enumerate(hashList)]
     multithread(download,hashList)

```

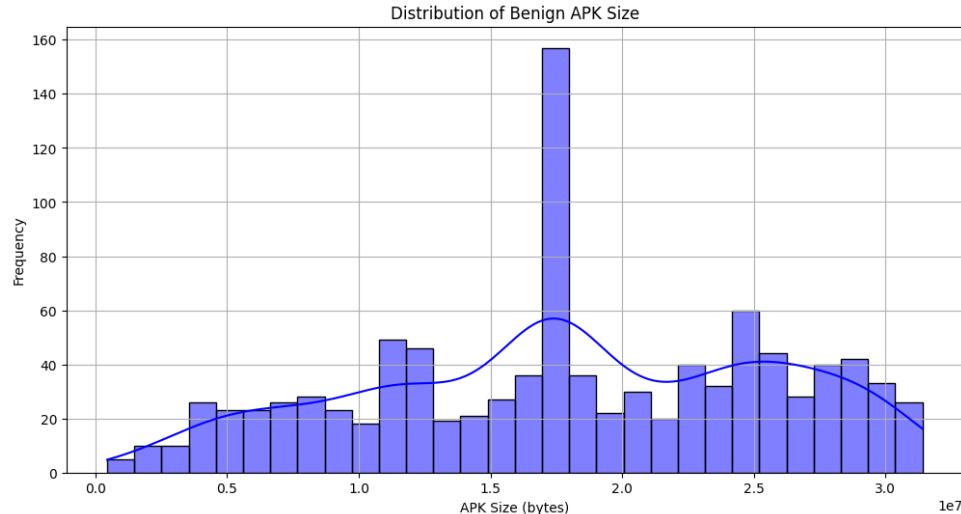
3.2 Thông kê và Trực quan hóa Dữ liệu

3.2.1 Biểu đồ

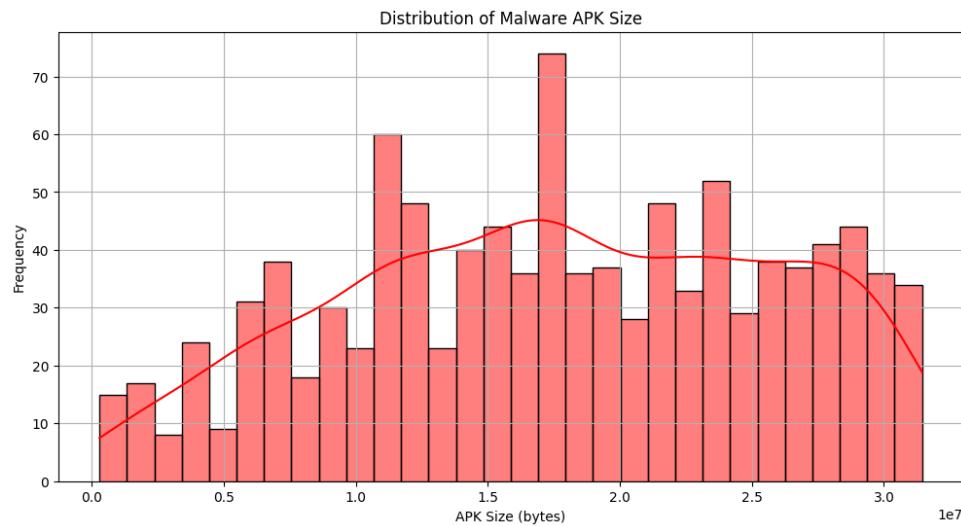
Trong phần này, chúng tôi trình bày các biểu đồ để trực quan hóa kích thước của các file APK thuộc hai loại: benign và malware. Các biểu đồ này bao gồm:

- **Biểu đồ Histogram:** Hiển thị phân bố kích thước APK, giúp nhận diện sự phân bố tổng quát của cả hai loại file.
- **Biểu đồ Boxplot:** Giúp xác định các giá trị ngoại lệ và khoảng phân bố của kích thước APK cho từng loại.
- **Biểu đồ KDE (Kernel Density Estimate):** Cung cấp một ước lượng mật độ cho phân phối kích thước APK, cho phép nhìn nhận rõ hơn về cấu trúc phân bố.
- **Biểu đồ Tần suất:** Hiển thị số lần xuất hiện của từng khoảng kích thước APK cho cả hai loại, từ đó giúp xác định các xu hướng kích thước.

3.2.2 Biểu Đồ Histogram



Hình 3.2: Benign histogram

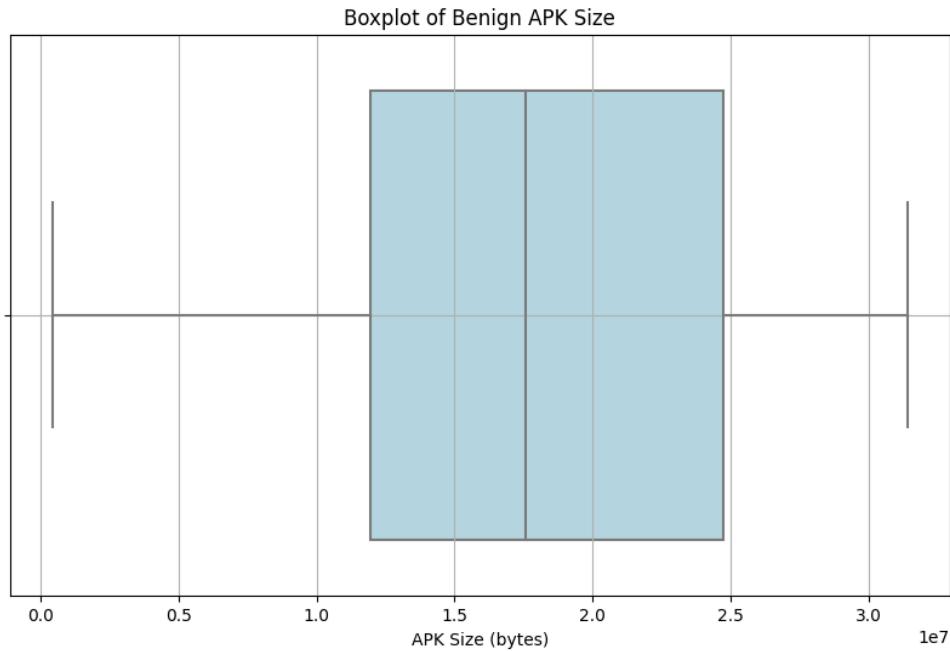


Hình 3.3: Malware histogram

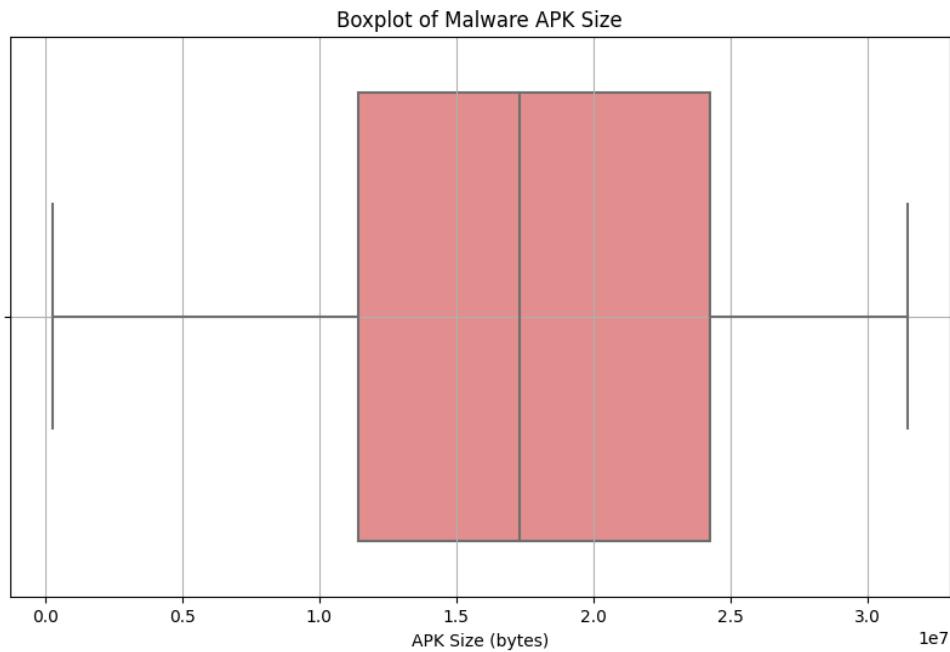
Biểu đồ histogram cho thấy sự phân bố kích thước của các file APK malware và benign. Qua phân tích, chúng tôi nhận thấy rằng:

- Đối với các file benign, kích thước thường tập trung trong khoảng từ 10 đến 20 MB, cho thấy đây là kích thước phổ biến của các ứng dụng an toàn.
- Ngược lại, kích thước của các file malware có sự phân bố rộng hơn, với nhiều file có kích thước nhỏ hơn 30 MB. Điều này cho thấy rằng nhiều malware được thiết kế để nhẹ hơn nhằm dễ dàng lén lút vào hệ thống.

3.2.3 Biểu Đồ Boxplot



Hình 3.4: Benign boxplot

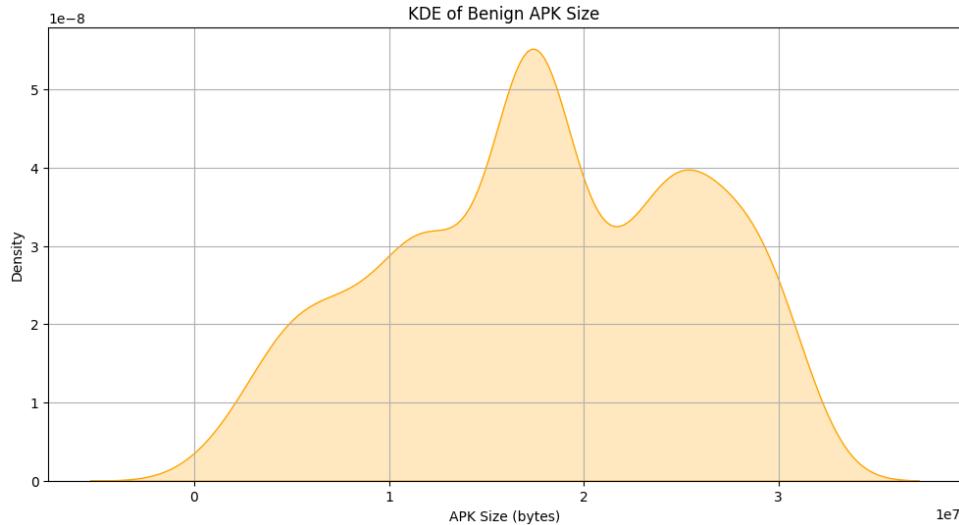


Hình 3.5: Malware boxplot

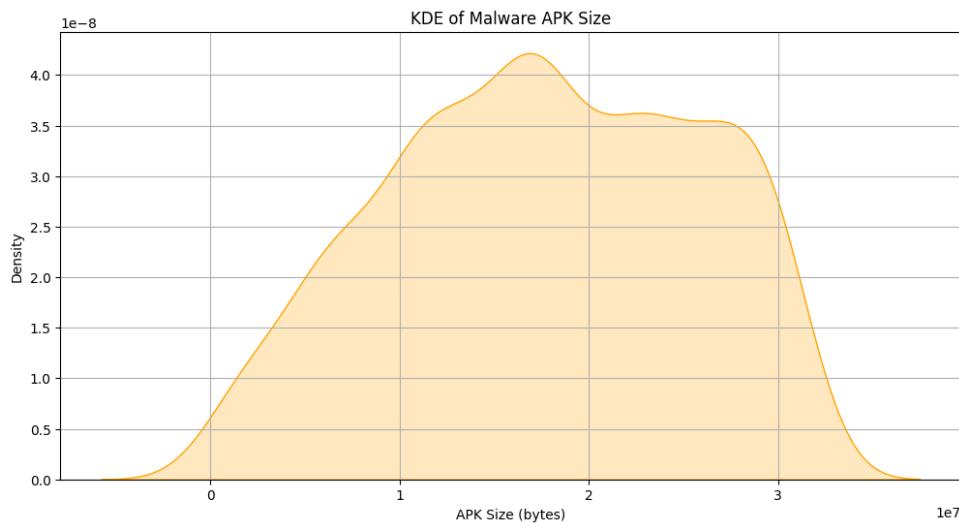
Biểu đồ boxplot cho thấy kích thước APK trung vị của benign khoảng 15 MB, trong khi malware có trung vị khoảng 10 MB. Phân tích sâu hơn cho thấy:

- Các giá trị ngoại lệ đáng chú ý ở cả hai loại cho thấy có nhiều file có kích thước lớn hơn, cụ thể là có một số file đạt đến kích thước lớn hơn 100 MB. Điều này có thể liên quan đến các ứng dụng đa chức năng hoặc các ứng dụng chứa nhiều tài nguyên.

3.2.4 Biểu Đồ KDE



Hình 3.6: Benign kde

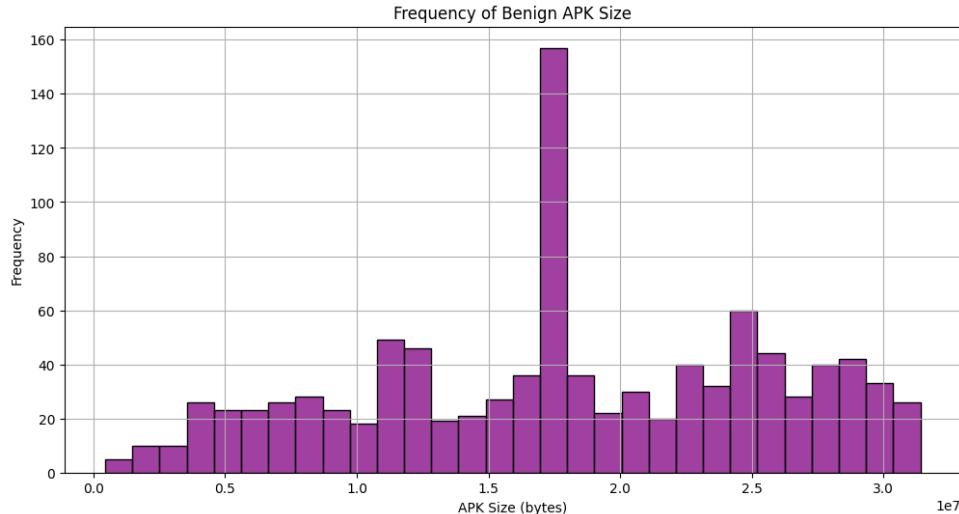


Hình 3.7: Malware kde

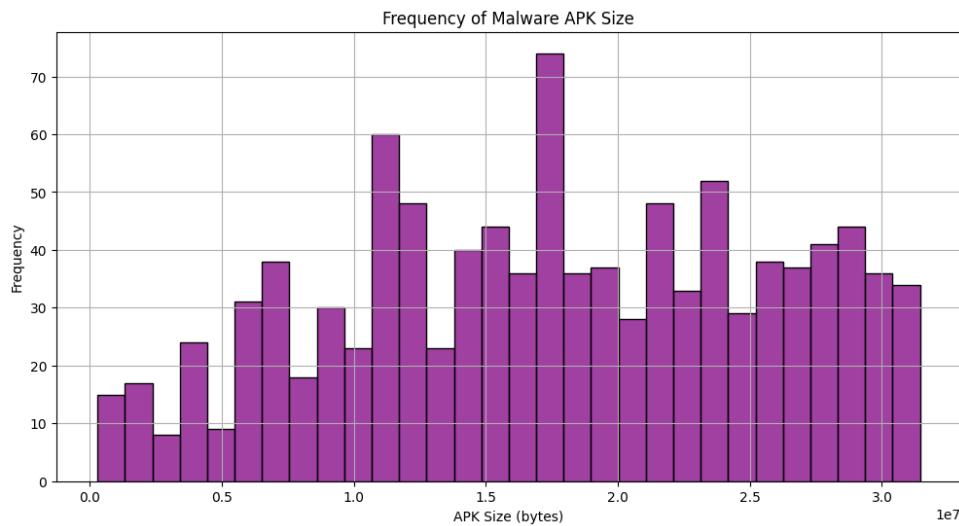
Biểu đồ KDE cung cấp cái nhìn mượt mà hơn về phân bố kích thước APK của cả hai loại. Phân tích cho thấy:

- Đường cong mật độ cho thấy rằng benign thường có kích thước lớn hơn malware. Điều này có thể phản ánh rằng các ứng dụng an toàn thường chứa nhiều chức năng và dữ liệu hơn.
- Hơn nữa, sự chồng chéo giữa hai phân bố cũng cho thấy một số file benign có thể bị nhầm lẫn với malware nếu chỉ dựa vào kích thước.

3.2.5 Biểu Đồ Tần Suất



Hình 3.8: Benign frequency



Hình 3.9: Malware frequency

Biểu đồ tần suất cho thấy số lượng file APK ở từng khoảng kích thước cho cả hai loại. Phân tích cho thấy:

- Các khoảng kích thước từ 10 đến 20 MB có tần suất cao nhất cho benign, cho thấy rằng đây là kích thước điển hình cho các ứng dụng an toàn.
- Đối với malware, tần suất cao hơn được ghi nhận ở khoảng kích thước nhỏ hơn 30 MB, nhấn mạnh việc thiết kế nhẹ hơn của nhiều malware.

3.2.6 Phân Tích

Qua việc trực quan hóa dữ liệu, rút ra một số nhận định quan trọng:

- Phân bố kích thước:** Phần lớn các file benign có kích thước lớn hơn so với malware. Điều này có thể gợi ý rằng nhiều malware được thiết kế nhẹ nhàng nhằm dễ dàng lén lút vào hệ thống.
- Giá trị ngoại lệ:** Cả hai loại file đều có giá trị ngoại lệ với một số file có kích thước lớn hơn 100 MB. Điều

này cần được xem xét kỹ lưỡng hơn để xác định chức năng của chúng, và có thể yêu cầu các bước kiểm tra bổ sung để xác định xem chúng có phải là nguy hiểm hay không.

- Tính khả thi trong phân tích:** Dữ liệu về kích thước APK có thể là một chỉ số hữu ích trong việc phân loại file. Tuy nhiên, cần kết hợp với các thông số khác như nội dung, độ phức tạp và hành vi để có được cái nhìn chính xác hơn về tính an toàn.

3.2.7 Kết Luận

Phân tích kích thước của các file APK malware và benign thông qua các biểu đồ trực quan hóa đã cung cấp cái nhìn sâu sắc về đặc điểm của cả hai loại. Việc sử dụng các phương pháp thống kê và trực quan hóa giúp chúng tôi hiểu rõ hơn về cấu trúc của dữ liệu, từ đó hỗ trợ cho quá trình phát triển mô hình phân loại. Trong tương lai, nhóm sẽ tiếp tục tìm cách cải thiện các tham số đầu vào và nghiên cứu sâu hơn về các giá trị ngoại lệ cũng như các yếu tố khác ảnh hưởng đến tính an toàn của các file APK.

3.3 Dịch ngược file APK thành file FCG

Thực hiện dịch ngược file APK thành file FCG

1. Tìm hiểu về lý thuyết và ứng dụng của việc dịch ngược APK thành FCG

- Dịch ngược (Reverse Engineering) là quá trình phân tích một chương trình, hệ thống hoặc phần mềm để hiểu rõ cấu trúc và cách thức hoạt động của nó. Đặc biệt, trong trường hợp này, chúng ta thực hiện dịch ngược tệp APK (Android Package) thành tệp FCG (Function Call Graph).
- Tệp APK chứa mã nguồn của ứng dụng Android, thường được biên dịch từ các tệp Java hoặc Smali. Việc chuyển đổi APK thành FCG giúp phân tích sự tương tác giữa các hàm trong mã nguồn, từ đó hỗ trợ tìm kiếm lỗ hổng bảo mật, phân tích hành vi của ứng dụng và kiểm tra các mối quan hệ trong mã nguồn.

2. Quá trình thực hiện dịch ngược APK thành FCG

- Đầu tiên, tệp APK được phân tích bằng công cụ Androguard, công cụ mạnh mẽ để phân tích các tệp APK và trích xuất các thành phần quan trọng như bytecode, các hàm, và đồ thị gọi hàm.
- Sau khi phân tích xong, chúng ta xây dựng đồ thị gọi hàm (Call Graph, CG) để biểu diễn các hàm và mối quan hệ gọi hàm giữa các thành phần trong ứng dụng.
- Cuối cùng, đồ thị gọi hàm sẽ được chuyển đổi thành một đồ thị DGL (Deep Graph Library) và lưu lại dưới dạng tệp FCG, có thể sử dụng cho các mục đích phân tích và học máy.

3. Cách sử dụng mã nguồn để dịch ngược APK thành FCG

- Đoạn mã dưới đây mô tả quá trình dịch ngược tệp APK thành tệp FCG:

```

1 import os
2
3 source_folder = "/kaggle/input/benign"
4 destination_folder = "/kaggle/working"
5
6 cnt = 0
7 for filename in os.listdir(source_folder):

```

```

8     cnt += 1
9     print("Idx: ", cnt)
10    file_path = os.path.join(source_folder, filename)
11    if os.path.isfile(file_path):
12        source_dest = (file_path, destination_folder)
13        process(source_dest)

```

- (b) Tiếp theo, chúng ta sử dụng công cụ Androguard để phân tích APK và trích xuất đồ thị gọi hàm:

```

1 from androguard.misc import AnalyzeAPK
2 import networkx as nx
3 from pathlib import Path
4 import dgl
5 def process(source_dest):
6     try:
7         source_file, dest_dir = source_dest
8         source_file = Path(source_file)
9         dest_dir = Path(dest_dir)
10        file_name = source_file.stem
11        _, _, dx = AnalyzeAPK(source_file)
12        cg = dx.get_call_graph()
13
14        cg = nx.convert_node_labels_to_integers(cg)
15        dg = dgl.from_networkx(cg, node_attrs[])
16        dest_dir = dest_dir / f'{file_name}.fcg'
17        dgl.data.utils.save_graphs(str(dest_dir), [dg])
18        print(f"Processed {source_file}")
19
20    except:
21        print(f"Error while processing {source_file}")

```

4. Áp dụng mã nguồn vào dự án

- (a) Đoạn mã trên sẽ duyệt qua tất cả các tệp APK trong thư mục source_folder, phân tích từng tệp và chuyển đổi đồ thị gọi hàm thành đồ thị DGL.
- (b) Các đồ thị DGL sẽ được lưu lại dưới dạng tệp .fcg trong thư mục destination_folder, với mỗi tệp APK tương ứng với một tệp FCG.

3.4 Dịch ngược file APK thành mã Java

Thực hiện dịch ngược file APK thành mã Java

1. Tìm hiểu về lý thuyết và ứng dụng của việc dịch ngược APK thành mã Java

- (a) Dịch ngược (Reverse Engineering) là quá trình phân tích một chương trình, hệ thống hoặc phần mềm để hiểu rõ cấu trúc và cách thức hoạt động của nó. Đặc biệt, trong trường hợp này, chúng ta thực hiện dịch ngược tệp APK (Android Package) thành mã Java.

- (b) Tệp APK chứa mã nguồn của ứng dụng Android, thường được biên dịch từ các tệp Java hoặc Smali. Việc chuyển đổi APK thành mã Java giúp phân tích hành vi của ứng dụng, tìm kiếm lỗ hổng bảo mật và hiểu rõ hơn về logic xử lý bên trong.

2. Quá trình thực hiện dịch ngược APK thành mã Java

- Đầu tiên, tệp APK được giải nén để trích xuất file classes.dex.
- File classes.dex được chuyển đổi thành file classes.jar bằng công cụ dex2jar.
- File classes.jar được dịch ngược thành mã Java bằng công cụ JADX.
- Toàn bộ mã Java được hợp nhất vào một tệp duy nhất để thuận tiện cho việc phân tích.

3. Cách sử dụng mã nguồn để dịch ngược APK thành mã Java

- (a) Đoạn mã dưới đây mô tả quá trình dịch ngược tệp APK:

```

1 import os
2 import subprocess
3 import shutil
4
5 APK_FILE = "your_apk_file.apk"
6 DEX2JAR_PATH = "d2j-dex2jar"
7 JADX_PATH = "jadx"
8
9 # Create temporary working directory
10 def create_temp_directory():
11     if not os.path.exists("temp_dir"):
12         os.makedirs("temp_dir")
13     return "temp_dir"
14
15 # Extract APK content
16 def extract_apk(apk_file, output_dir):
17     shutil.unpack_archive(apk_file, output_dir)
18     print(f"APK extracted to {output_dir}")
19
20 # Convert .dex to .jar using dex2jar
21 def convert_dex_to_jar(dex_path, output_jar):
22     dex2jar_command = [DEX2JAR_PATH, dex_path, "-o", output_jar]
23     subprocess.run(dex2jar_command, check=True)
24     print(f"DEX converted to JAR: {output_jar}")
25
26 # Decompile .jar to Java source using JADX
27 def decompile_jar(jar_file, output_dir):
28     jadx_command = [JADX_PATH, jar_file, "-d", output_dir]
29     subprocess.run(jadx_command, check=True)
30     print(f"JAR decompiled to Java source in: {output_dir}")
31

```

```

32 # Merge all Java files into one Java file
33 def merge_java_files(source_dir, output_java_file):
34     with open(output_java_file, 'w') as merged_file:
35         for root, _, files in os.walk(source_dir):
36             for file in files:
37                 if file.endswith(".java"):
38                     file_path = os.path.join(root, file)
39                     with open(file_path, 'r') as java_file:
40                         merged_file.write(java_file.read())
41                         merged_file.write("\n")
42     print(f"All Java files merged into {output_java_file}")
43
44 # Main workflow
45 def main():
46     temp_dir = create_temp_directory()
47     extracted_dir = os.path.join(temp_dir, "extracted_apk")
48     jar_output_path = os.path.join(temp_dir, "classes.jar")
49     jadx_output_dir = os.path.join(temp_dir, "jadex_output")
50     merged_java_file = "merged_output.java"
51
52     try:
53         extract_apk(APK_FILE, extracted_dir)
54         dex_path = os.path.join(extracted_dir, "classes.dex")
55         convert_dex_to_jar(dex_path, jar_output_path)
56         decompile_jar(jar_output_path, jadx_output_dir)
57         merge_java_files(jadx_output_dir, merged_java_file)
58     except Exception as e:
59         print(f"Error: {e}")
60     finally:
61         if os.path.exists(temp_dir):
62             shutil.rmtree(temp_dir)
63             print("Temporary directory cleaned up")
64
65 if __name__ == "__main__":
66     main()

```

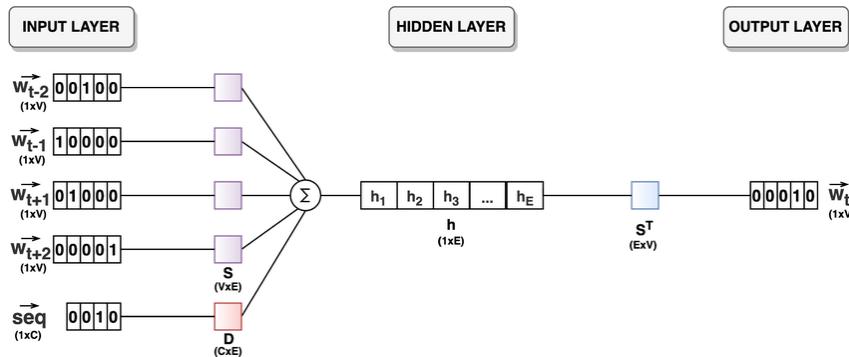
4. Áp dụng mã nguồn vào dự án

- (a) Đoạn mã trên thực hiện toàn bộ quá trình xử lý file APK, từ giải nén, chuyển đổi file DEX, dịch ngược file JAR đến hợp nhất mã Java.
- (b) Các tệp mã nguồn Java sau khi trích xuất có thể được sử dụng để phân tích bảo mật, nghiên cứu cấu trúc ứng dụng hoặc kiểm tra logic hoạt động.

3.5 Nhúng code Java bằng Doc2vec

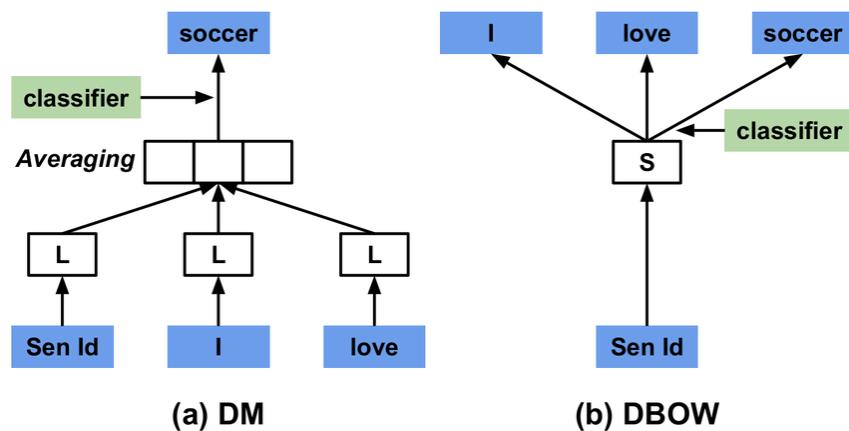
1. Lý thuyết doc2vec

- (a) Mục tiêu của Doc2Vec là học các biểu diễn vector cho các đoạn văn, câu, hoặc tài liệu, thay vì chỉ học vector cho các từ đơn lẻ như trong Word2Vec. Bằng cách này, Doc2Vec không chỉ nắm bắt ý nghĩa của từng từ mà còn hiểu được ngữ cảnh tổng thể của một đoạn văn hoặc tài liệu, từ đó giúp mô hình có khả năng biểu diễn ngữ nghĩa của các văn bản dài.



Hình 3.10: Architecture Doc2vec

- (b) Có hai kiến trúc chính trong Doc2Vec là Distributed Memory (DM) và Distributed Bag of Words (DBOW). Với DM, mô hình học cách dự đoán từ tiếp theo trong một câu dựa trên các từ trước đó và vector của tài liệu, kết hợp cả thông tin về ngữ cảnh và cấu trúc của văn bản. Điều này giúp vector biểu diễn của tài liệu giữ được ngữ cảnh mà các từ xuất hiện. Ngược lại, DBOW hoạt động tương tự như mô hình Skip-gram của Word2Vec, khi nó học cách dự đoán các từ ngẫu nhiên trong tài liệu mà không sử dụng thông tin ngữ cảnh. Dù cách tiếp cận khác nhau, cả hai đều nhằm mục đích tạo ra một biểu diễn vector cho toàn bộ tài liệu.



Hình 3.11: DM and DBOW

2. Áp dụng kỹ thuật doc2vec vào bài toán phân loại apk: khi mà phân tích file apk ta nhận thấy ta có thể dịch ngược file apk thành file java, và từ đó ta có thể làm giàu đồ thị hơn với các đặc trưng trích xuất từ file ngôn ngữ java

- (a) Doc2Vec vào ngôn ngữ Java, chúng ta có thể xây dựng các ứng dụng phân tích văn bản và xử lý ngôn ngữ tự nhiên (NLP) mạnh mẽ. Doc2Vec giúp chuyển các đoạn văn bản, tài liệu, hoặc đoạn mã nguồn

thành các vector có kích thước cố định, biểu diễn ngữ nghĩa của nội dung. Điều này có thể áp dụng cho nhiều bài toán liên quan đến mã nguồn trong Java, chẳng hạn như phân loại, tìm kiếm và gọi ý mã, phát hiện lỗi, và phân tích mã nguồn.

- (b) Doc2Vec mang lại nhiều lợi ích khi áp dụng vào việc phân tích và xử lý mã nguồn Java. Nó giúp biểu diễn mã nguồn dưới dạng các vector số học, cho phép các mô hình học máy hiểu được ngữ nghĩa của các đoạn mã, từ đó hỗ trợ phân loại các file apk

3. Quá trình sử dụng kỹ thuật doc2vec vào bài toán phân loại apk: Kỹ thuật Doc2Vec có thể được sử dụng để phân loại các tệp APK (Android Package) bằng cách chuyển đổi mã nguồn java của APK thành các biểu diễn vector, sau đó sử dụng các vector này để huấn luyện mô hình phân loại.

- (a) APK chứa mã nguồn (thường là mã Java hoặc mã Smali)
- (b) Mã nguồn: Mã nguồn Java hoặc Smali cần được chuyển thành dạng văn bản, tương tự như một tài liệu để có thể áp dụng Doc2Vec. Có thể coi mỗi đoạn mã (class, method) là một "tài liệu".
- (c) Sau khi chuyển đổi thành các file java ta tiến hành đào tạo (training) một mô hình Doc2Vec sử dụng kỹ thuật DM của Doc2Vec với bộ dữ liệu java đã tạo ở trên.
- (d) Thu được kết quả mô hình ở trên, ta thực hiện nhúng từng đoạn văn bản (một hàm trong file java, là một node trong đồ thị GCN để training mô hình) thành một vector mang đặc trưng của node đó trong đồ thị GCN.
- (e) Ví dụ một số hàm trong java (nội dung một vài node trong đồ thị):

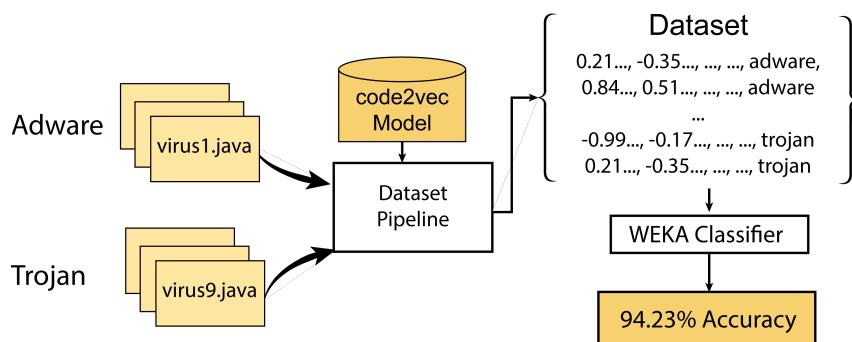
```

1  public String getObfuscatedAccountId()
2  {
3      return this.zza;
4  }
5
6
7  public String getObfuscatedProfileId()
8  {
9      return this.zzb;
10 }
11
12
13 private AcknowledgePurchaseParams$Builder()
14 {
15     return;
16 }
17
18
19 synthetic AcknowledgePurchaseParams$Builder(com.android.billingclient.
20     api.zza p1)
21 {
22     return;
23 }
```

```

22     }
23
24
25     public com.android.billingclient.api.AcknowledgePurchaseParams build()
26     {
27         if (this.zza != null) {
28             com.android.billingclient.api.AcknowledgePurchaseParams v0_3 =
29                 new com.android.billingclient.api.AcknowledgePurchaseParams
30                 (0);
31             com.android.billingclient.api.AcknowledgePurchaseParams.zza(
32                 v0_3, this.zza);
33             return v0_3;
34         } else {
35             throw new IllegalArgumentException(Purchase token must be set);
36         }
37     }

```



Hình 3.12: Embedding java with doc2vec

4. Thực thi áp dụng doc2vec vào dự án:

- (a) Thực hiện sử dụng kiến trúc đã được xây sẵn trong package **gensim**
- (b) Thực hiện extract các hàm từ file java đã được dịch ngược từ trước đó:

```

1 def extract_functions_from_java_file(file_path):
2     """Function to extract functions from a Java file"""
3     with open(file_path, 'r') as file:
4         lines = file.readlines()
5
6     functions = []
7     current_function = []
8     inside_function = False
9     brace_count = 0
10
11    for line in lines:
12        stripped_line = line.strip()

```

```
13
14     # Skip lines that define the class
15
16     if stripped_line.startswith('class '):
17         continue
18
19
20     # Check if the line starts with "public", "private", "protected",
21     # or "static"
22
23     if stripped_line.startswith(('public ', 'private ', 'protected ',
24                               'static ')) and ' ' in stripped_line:
25
26         if current_function:
27
28             functions.append(''.join(current_function))
29
30         current_function = []
31
32         inside_function = True
33
34
35     # Add line to the current function
36
37     if inside_function:
38
39         current_function.append(line)
40
41         brace_count += line.count('{')
42
43         brace_count -= line.count('}')
44
45
46     # If brace_count is 0, the function has ended
47
48     if brace_count == 0:
49
50         functions.append(''.join(current_function))
51
52         current_function = []
53
54         inside_function = False
55
56
57
58     return functions
```

- (c) Sau khi mà ta đã thực hiện xong việc trích xuất các hàm từ các file java, ta tiến hành mang vào training model doc2vec

```
1 def train_doc2vec_model(tokenized_functions):  
2     """Function to train a Doc2Vec model and vectorize functions"""  
3     documents = [TaggedDocument(words=tokens, tags=[f"function_{i}"]) for i  
4                   , tokens in enumerate(tokenized_functions)]  
5     model = Doc2Vec(vector_size=100, window=5, min_count=1, workers=4,  
6                      epochs=50)  
7  
8  
9     # Build vocabulary  
10    model.build_vocab(documents)  
11  
12  
13    # Train the model  
14    model.train(documents, total_examples=model.corpus_count, epochs=model.  
15                epochs)
```

```
12     return model
```

- (d) Model được training xong ta mang vào thực hiện nhúng từng hàm một mà ta trích xuất từ java ở trên qua mô hình doc2vec ta thu vector của một hàm là 200

```
1 # Step 3: load model doc2vec
2 from gensim.models.doc2vec import Doc2Vec
3 model = Doc2Vec.load("/kaggle/input/doc2vec/other/default/1/
4                 java_8m_methods_doc2vec.model")
5
6 # Java file
7 # Find all .java files in the directory
8 java_files = []
9 for root, _, files in os.walk(folder_path):
10     for file in files:
11         if file.endswith('.java'):
12             java_files.append(os.path.join(root, file))
13
14 # Step 4: Save vectors to .txt files with corresponding names
15 for idx, file_path in enumerate(java_files):
16     print("Idx:", idx)
17     funcs = docs[idx]
18     embs = []
19     for func in funcs:
20         emb = model.infer_vector(func)
21         embs.append(emb.tolist())
22     output_file = os.path.join('/kaggle/working', f"{os.path.splitext(os.
23         path.basename(file_path))[0]}.txt")
24     save_vectors_to_file(embs, output_file)
```

-> Kết luận: Ta thu được bộ dữ liệu đã được thực hiện nhúng quá Doc2Vec mang ý nghĩa của từng node trong từng file apk tương ứng, từ đó bộ dữ liệu bổ sung đặc trưng cho từng node của đồ thị GCN.

1. Log khi nhúng các hàm trong file java qua doc2vec:

The screenshot shows a Jupyter Notebook interface with the title "Doc2vecMalware". Below the title, it says "Copied from Đào Duy Thông (+1, -1)". There are tabs for Notebook, Input, Output, Logs, and Comments (0). The Logs tab is selected, showing the following log messages:

```

Logs
Download Logs

Successfully ran in 826.0s

Time # Log Message
20.2s 1 Dang xú ly: /kaggle/input/malwarejava/9849c6ca73e15cb37842f0f210fd86bb0f2c731268bd788246c389a40e3295f7.java
20.2s 2 Dang xú ly: /kaggle/input/malwarejava/680aa7551f6ad03f9038fb08cf040919fd7f84b6a97fd1110501093f050d.java
20.2s 3 Dang xú ly: /kaggle/input/malwarejava/51ec5c181fc588b0f6811bc362021176f34ce5e530b63011613c48939310888a.java
20.2s 4 Dang xú ly: /kaggle/input/malwarejava/f76899a38ef6f8990e94788440116a280d446f44eeff4a1810af314e36.java
20.2s 5 Dang xú ly: /kaggle/input/malwarejava/a8073116399e6a0cd3c12ea024a97530d8836711bc5750fa14f10753d007841c.java
20.2s 6 Dang xú ly: /kaggle/input/malwarejava/8c880d4416e79ca65242241bc77308fc98bf2e1fa04090c88eade088603f.java
20.2s 7 Dang xú ly: /kaggle/input/malwarejava/bc850c9692e84f47e7c5908e6e893e47966b1a2a5a84970fa3c7c936c988940.java
20.4s 8 Dang xú ly: /kaggle/input/malwarejava/7ee0de0207e07c50fa312e056611a82f0973ef9ed0c3914fa3569fa828cf.java
20.4s 9 Dang xú ly: /kaggle/input/malwarejava/1efc9c9bcc22e5438484f17a75ff7511ea9f5e77f5fa1bc6f677ef2f1177901.java
20.4s 10 Dang xú ly: /kaggle/input/malwarejava/4d910e5037217ed4243687f491a57f1191e978933f7ff96684470fa16420.java
20.7s 11 Dang xú ly: /kaggle/input/malwarejava/2f7989558f773400cf70e1932129821ce558781fe062ff5ff3948814ef64ff5.java
20.7s 12 Dang xú ly: /kaggle/input/malwarejava/825377e801625185eed43aa591633810485184c/eed6d8ac24323684911.java
20.7s 13 Dang xú ly: /kaggle/input/malwarejava/42c8027884618ac8a9f980e87799cef61916e44247dcf934e74210808.java
20.7s 14 Dang xú ly: /kaggle/input/malwarejava/50504671e8950a7f8510ab64880164c7f982c88f78ec8a68d0a449e11e0d224.java

```

Hình 3.13: Log Embedding java qua doc2vec

2. Dữ liệu sau khi nhúng:

The screenshot shows a Jupyter Notebook interface with the title "Doc2vecMalware". Below the title, it says "Copied from Đào Duy Thông (+1, -1)". There are tabs for Notebook, Input, Output, Logs, and Comments (0). The Output tab is selected, showing the following output data:

Output Data

Malware_Doc2Vec

This preview is truncated due to the large file size. Create a Notebook or download this file to see the full content.

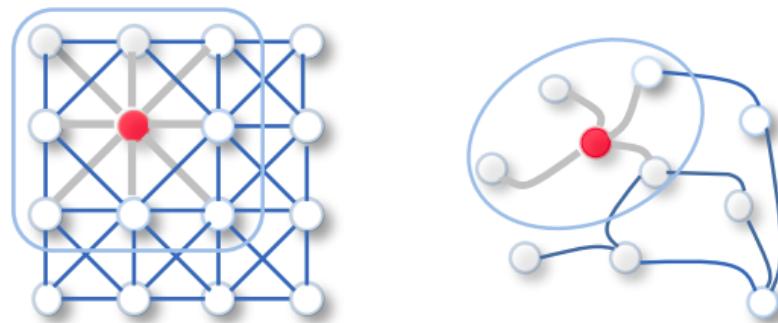
Output :

- 0000E50E0F238EEA30E5F5001
- 010C44A9CE3071C1772088F5A44E625660DA9B5C9759295516DA9ACA2FD4C6E... ↴ >
- 010C44A9CE3071C1772088F
- 0102042A02D2C9A6E46C43ED044
- 020204179FC9382BA70A7B
- 020304C48904C0242283935E
- 020404C48904C0242283935F
- 0224MF13CC2AF7D488778
- 024A0D9EBC0EC23CD058F
- 0202042F1C5E397889CAF7D790
- 0324CFBF49285833D04E53C
- 033102F8E9500758EEF0202088P
- 0334D1347C7C58848CF0D01
- 02032C074934F0D47CA2
- 03E274FE7F5328405924F177
- 04095080BD023584E6705
- 041CCB5A5019419A184F428E4
- 0444A0222098A51D03283E
- 04F0C0446080DC503E33EF9
- 05003C3F81524B01C092C0
- 05709F540D9B91C6AC423C
- 0655E5E66013D044961672
- 065B92511FB9C7A8262D4F

Hình 3.14: Kết quả Embedding java qua doc2vec

3.6 Graph convolutional network

1. Khái niệm: Graph convolutional network (GCN) là một loại neural network được thiết kế để làm việc với dữ liệu đồ thị. Mạng CNN truyền thống hoạt động trên dữ liệu dạng grid bằng cách di chuyển một kernel qua dữ liệu, thu thập thông tin các điểm dữ liệu trong kernel đó. GCN cũng hoạt động theo ý tưởng tương tự CNN, tuy nhiên kernel thông thường không thể hoạt động trên dữ liệu biểu đồ. Thay vì sử dụng kernel, GCN sẽ duyệt qua các node, trích xuất thông tin của node đó và các node lân cận. Bằng phương pháp này, GCN có thể học được thông tin của các node cùng với thông tin môi trường quanh node đó trong biểu đồ.



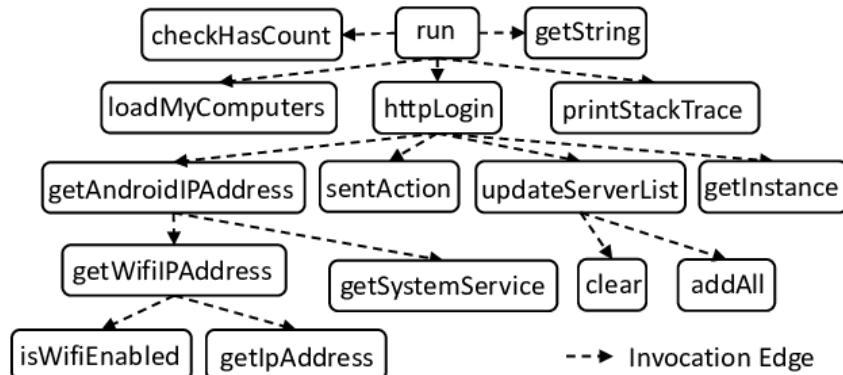
(a) 2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes the weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size.

(b) Graph Convolution. To get a hidden representation of the red node, one simple solution of the graph convolutional operation is to take the average value of the node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size.

Hình 3.15: CNN vs GCN

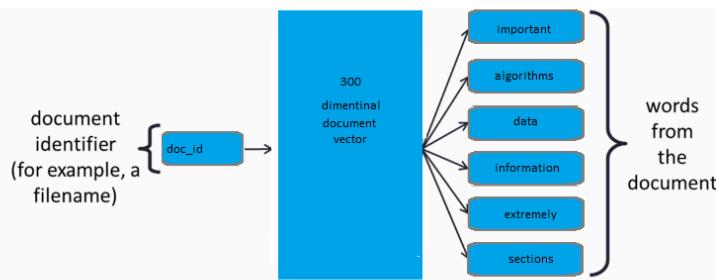
2. **Dữ liệu đầu vào:** Sau khi xử lý file apk ta sẽ có dữ liệu gồm:

- (a) Function call graph (FCG): một biểu đồ với các node tương ứng với các hàm trong chương trình, các cạnh tương ứng với các lời gọi hàm.



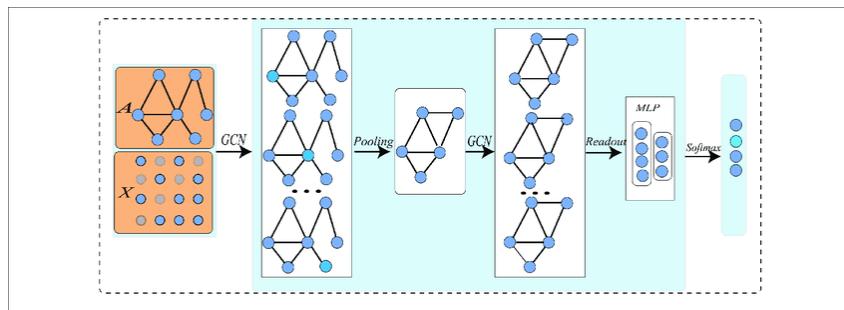
Hình 3.16: Function call graph

- (b) Các vector tương ứng cho feature của các hàm trong FCG



Hình 3.17: Doc2Vec

(c) **Mô hình:** Mô hình gồm 2 lớp GCN để trích xuất thông tin từ biểu đồ. Sau khi đi qua các lớp GCN ta cần dự đoán một nhãn cho toàn bộ biểu đồ thay vì từng node. Quy trình trên được gọi là "readout". Một phương pháp đơn giản để thực hiện điều này là lấy giá trị trung bình từ features của các node sử dụng dgl.mean_nodes.



Hình 3.18: Tổng quan mô hình

```

1 class GCN(nn.Module):
2     def __init__(self, in_feats, h_feats, num_classes):
3         super(GCN, self).__init__()
4         self.conv1 = GraphConv(in_feats, h_feats)
5         self.conv2 = GraphConv(h_feats, num_classes)
6
7     def forward(self, g, in_feat):
8         h = self.conv1(g, in_feat)
9         h = F.relu(h)
10        h = self.conv2(g, h)
11        g.ndata['h'] = h
12        return dgl.mean_nodes(g, 'h')

```

3. **Output:** Kết quả của mô hình là một trong hai nhãn Benign (Không có mã độc) và Malware (Mã độc)

3.7 Xây dựng mô hình

Mô hình xây dựng trên Kaggle

1. Chuẩn bị

Sử dụng Pytorch để xây dựng mô hình; Kiểm tra các GPU có thể sử dụng

```

1 import torch
2 print(torch.__version__)
3
4 !nvidia-smi

```

```

Mon Nov 18 13:20:50 2024
+-----+
| NVIDIA-SMI 550.90.07      Driver Version: 550.90.07    CUDA Version: 12.4 |
|-----+
| GPU  Name           Persistence-M  Bus-Id     Disp.A | Volatile Uncorr. ECC | | | | | | |
| Fan  Temp  Perf        Pwr:Usage/Cap | Memory-Usage | GPU-Util  Compute M. |
| |      |             |          |              |      |          |       |
|-----+
| 0   Tesla P100-PCIE-16GB     Off | 00000000:00:04.0 Off |          0 | | | | | |
| N/A   38C   P0            27W / 250W |    0MiB / 16384MiB |    0%     Default |
| |      |             |          |              |      |          |       |
+-----+
+-----+
| Processes:
| GPU  GI CI      PID  Type  Process name                  GPU Memory |
| ID   ID
|-----|
| No running processes found
+-----+

```

Cài đặt thư viện dgl (dgl là một thư viện mã nguồn mở được sử dụng để xây dựng và huấn luyện các mô hình học sâu trên đồ thị (graph), ví dụ như GCN).

```

1 !pip install dgl -f https://data.dgl.ai/wheels/torch-2.4/cu124/repo.html

```

Test đầu ra cho dgl

```

1 import dgl
2 g, label_dict = dgl.load_graphs("/kaggle/input/benign-fcg/08
D75AF7C56DD80D2712344279998C4082BD17C591ABEE4F29F4A81D59128E3D.fcg")
3 g[0]

```

```

Graph(num_nodes=205, num_edges=435,
      ndata_schemes={}
      edata_schemes={})

```

Test đầu ra cho Pytorch

```

1 features = []
2 with open('/kaggle/input/benign-doc2vec/08
D75AF7C56DD80D2712344279998C4082BD17C591ABEE4F29F4A81D59128E3D.txt', 'r')
as f:
3     features = ast.literal_eval(f.read().strip()) # Converts the nested list
                                                string into an actual Python list

```

```

4
5 # Step 2: Convert the list to a PyTorch tensor
6 features_tensor = torch.tensor(features, dtype=torch.float32)
7
8 # Step 3: Add the feature tensor to the graph
9 # Assuming you already have a DGL graph `g` with matching node count
10 # g[0].ndata['feat'] = features_tensor
11 features_tensor.shape

```

torch.Size([51, 200])

Import các thư viện cần thiết

```

1 import dgl
2 import torch
3 import torch.nn as nn
4 import torch.nn.functional as F
5 import os
6 import ast

```

2. Tiền xử lý dữ liệu

Khai báo hàm load_graphs_with_labels_and_padding(graph_folders, feature_folders, labels)

- Đọc và tải đồ thị từ các tệp .fcg. (Duyệt qua các thư mục đồ thị và đặc trưng: Hàm thực hiện duyệt qua từng thư mục chứa tệp đồ thị (.fcg) và tệp đặc trưng (.txt), đồng thời lấy nhãn tương ứng từ danh sách labels.)
- Đọc và thêm đặc trưng (features) từ các tệp .txt. (Tải đồ thị: Sử dụng dgl.load_graphs để tải đồ thị từ các tệp .fcg.; Tải đặc trưng: Đọc tệp .txt và chuyển nội dung thành tensor PyTorch.)
- Xử lý sự không khớp kích thước giữa số lượng node trong đồ thị và số lượng dòng trong tensor đặc trưng (bằng cách bổ sung hoặc cắt bớt).
 - Nếu thiếu node: Thêm các hàng vector toàn số 0.
 - Nếu thừa node: Cắt tensor để khớp với số lượng node.
- Gắn đặc trưng vào đồ thị và gán nhãn tương ứng: Đặc trưng được gắn vào đồ thị bằng g.ndata['feat'].
- Trả về danh sách chứa các tuple (đồ thị, nhãn) để sử dụng trong huấn luyện mô hình. (Nhãn được chuyển thành tensor PyTorch và cùng đồ thị tạo thành một tuple (graph, label). Tuple này được thêm vào danh sách kết quả dataset.)
- Các tham số đầu vào:
 - graph_folders: Danh sách các thư mục chứa các tệp đồ thị (.fcg).

- feature_folders: Danh sách các thư mục chứa các tệp đặc trưng (.txt).
- labels: Danh sách các nhãn tương ứng với từng tập dữ liệu đồ thị và đặc trưng.

```

1 def load_graphs_with_labels_and_padding(graph_folders, feature_folders, labels):
2     :
3     dataset = []
4     labels_list = []
5     total_read = 0
6     for graph_folder, feature_folder, label in zip(graph_folders,
7             feature_folders, labels):
8         # List all graph and feature files
9         graph_files = sorted([f for f in os.listdir(graph_folder) if f.endswith
10             ('.fcg')])
11         feature_files = sorted([f for f in os.listdir(feature_folder) if f.
12             endswith('.txt')])
13         print(graph_folder)
14         for graph_file, feature_file in zip(graph_files, feature_files):
15             # if total_read >= 10:
16             #     return dataset
17             # Load the graph
18
19             graph_path = os.path.join(graph_folder, graph_file)
20             graphs, _ = dgl.load_graphs(graph_path)
21             g = graphs[0] # Assuming each file contains a single graph
22
23             # Load the features
24             feature_path = os.path.join(feature_folder, feature_file)
25             with open(feature_path, 'r') as f:
26                 features = ast.literal_eval(f.read().strip()) # Convert string
27                 to list
28             features_tensor = torch.tensor(features, dtype=torch.float32)
29
30             # Handle mismatched dimensions
31             num_nodes = g.num_nodes()
32             num_features = features_tensor.shape[1]
33             if features_tensor.shape[0] < num_nodes:
34                 # Pad with zeros
35                 padding = torch.zeros((num_nodes - features_tensor.shape[0],
36                     num_features), dtype=torch.float32)
37                 features_tensor = torch.cat([features_tensor, padding], dim=0)
38             elif features_tensor.shape[0] > num_nodes:
39                 # raise ValueError(f"Feature tensor has more rows ({{
40                     features_tensor.shape[0]}} than graph nodes ({num_nodes}){{
41                         .}")
42                 features_tensor = features_tensor[:num_nodes, :]

```

```

35
36     # Add features to the graph
37     g.ndata['feat'] = features_tensor
38
39     # Add the graph and label to the dataset
40     # Create the label tensor
41     label_tensor = torch.tensor(label, dtype=torch.long)
42
43     # Add the (graph, label) tuple to the dataset
44     dataset.append((g, label_tensor))
45     print(total_read,"Num nodes:",num_nodes, "Feature shapes:",
46             features_tensor.shape, "Label:", label)
47     total_read += 1
48
49
50 return dataset

```

Khai báo các đường dẫn tới các tệp dữ liệu và gọi hàm

```

1 benign_graph_folder = '/kaggle/input/benign-fcg/'
2 malware_graph_folder = '/kaggle/input/malware-fcg/'
3 benign_feature_folder = '/kaggle/input/benign-doc2vec/'
4 malware_feature_folder = '/kaggle/input/malware-doc2vec/'
5
6 graph_folders = [benign_graph_folder, malware_graph_folder]
7 feature_folders = [benign_feature_folder, malware_feature_folder]
8 labels = [0, 1] # Assign 0 for benign, 1 for malware
9
10 # Load dataset and labels
11 # dataset, labels = load_graphs_with_labels_and_padding(graph_folders,
12 #                                                       feature_folders, labels)
12 dataset = load_graphs_with_labels_and_padding(graph_folders,
13                                             feature_folders, labels)

```

```

/kaggle/input/benign-fcg/
0 Num nodes: 1586 Feature shapes: torch.Size([1586, 200]) Label: 0
1 Num nodes: 1467 Feature shapes: torch.Size([1467, 200]) Label: 0
2 Num nodes: 691 Feature shapes: torch.Size([691, 200]) Label: 0
3 Num nodes: 3400 Feature shapes: torch.Size([3400, 200]) Label: 0
4 Num nodes: 1341 Feature shapes: torch.Size([1341, 200]) Label: 0
5 Num nodes: 5999 Feature shapes: torch.Size([5999, 200]) Label: 0
6 Num nodes: 6824 Feature shapes: torch.Size([6824, 200]) Label: 0
7 Num nodes: 191 Feature shapes: torch.Size([191, 200]) Label: 0
8 Num nodes: 7689 Feature shapes: torch.Size([7689, 200]) Label: 0
9 Num nodes: 11586 Feature shapes: torch.Size([11586, 200]) Label: 0
10 Num nodes: 15181 Feature shapes: torch.Size([15181, 200]) Label: 0
11 Num nodes: 295 Feature shapes: torch.Size([295, 200]) Label: 0
12 Num nodes: 12051 Feature shapes: torch.Size([12051, 200]) Label: 0

```

Hình 3.19: Áp dụng cho 1771 file

```

1 for i, (graph, label) in enumerate(dataset[:10]):
2     print(f"Graph {i}: Label {label}, {graph}")

Labels: [0, 1]
Graph 0: Label 0, Graph(num_nodes=1586, num_edges=4193,
    ndata_schemes={'feat': Scheme(shape=(200,), dtype=torch.float32)}
    edata_schemes={})
Graph 1: Label 0, Graph(num_nodes=1467, num_edges=3612,
    ndata_schemes={'feat': Scheme(shape=(200,), dtype=torch.float32)}
    edata_schemes={})
Graph 2: Label 0, Graph(num_nodes=691, num_edges=1770,
    ndata_schemes={'feat': Scheme(shape=(200,), dtype=torch.float32)}
    edata_schemes={})
Graph 3: Label 0, Graph(num_nodes=3400, num_edges=11158,
    ndata_schemes={'feat': Scheme(shape=(200,), dtype=torch.float32)}
    edata_schemes={})
Graph 4: Label 0, Graph(num_nodes=1341, num_edges=3002,
    ndata_schemes={'feat': Scheme(shape=(200,), dtype=torch.float32)}
    edata_schemes={})
Graph 5: Label 0, Graph(num_nodes=5999, num_edges=10117,
    ndata_schemes={'feat': Scheme(shape=(200,), dtype=torch.float32)}
    edata_schemes={})
Graph 6: Label 0, Graph(num_nodes=6824, num_edges=23481,
    ndata_schemes={'feat': Scheme(shape=(200,), dtype=torch.float32)}
    edata_schemes={})
Graph 7: Label 0, Graph(num_nodes=191, num_edges=340,
    ndata_schemes={'feat': Scheme(shape=(200,), dtype=torch.float32)}
    edata_schemes={})
Graph 8: Label 0, Graph(num_nodes=7689, num_edges=27994,
    ndata_schemes={'feat': Scheme(shape=(200,), dtype=torch.float32)}
    edata_schemes={})
Graph 9: Label 0, Graph(num_nodes=11586, num_edges=37794,
    ndata_schemes={'feat': Scheme(shape=(200,), dtype=torch.float32)}
    edata_schemes={})

```

Chia dữ liệu (dataset) thành hai phần: dữ liệu huấn luyện và dữ liệu kiểm tra, sau đó tạo các DataLoader để nạp dữ liệu trong quá trình huấn luyện mô hình đồ thị.

- Chia dữ liệu theo tỷ lệ 80:20 giúp tạo tập huấn luyện và kiểm tra để đánh giá mô hình.
- Sử dụng sampler ngẫu nhiên đảm bảo mỗi lần chạy mô hình, các batch có thể khác nhau, giúp mô hình huấn luyện tốt hơn.

```

1 from dgl.dataloading import GraphDataLoader
2 from torch.utils.data.sampler import SubsetRandomSampler
3
4 num_examples = len(dataset)
5 num_train = int(num_examples * 0.8)
6
7 train_sampler = SubsetRandomSampler(torch.arange(num_train))
8 test_sampler = SubsetRandomSampler(torch.arange(num_train, num_examples))
9
10 train_dataloader = GraphDataLoader(

```

```

11     dataset, sampler=train_sampler, batch_size=5, drop_last=False)
12 test_dataloader = GraphDataLoader(
13     dataset, sampler=test_sampler, batch_size=5, drop_last=False)

```

```

1 it = iter(train_dataloader)
2 batch = next(it)
3 print(batch)

```

```

[Graph(num_nodes=9336, num_edges=20066,
      ndata_schemes={'feat': Scheme(shape=(200,), dtype=torch.float32)},
      edata_schemes={}), tensor([0, 0, 0, 0, 0])]

```

Thực hiện các thao tác trên một batch dữ liệu đồ thị (graph) đã được tạo ra từ DataLoader của DGL. Hiển thị thông tin:

- Thông tin số lượng node và cạnh trong batch.
- Tách (unbatch) batch thành các đồ thị gốc.
- In ra danh sách các đồ thị đã tách.

Batched đồ thị: Trong DGL, nhiều đồ thị có thể được gộp lại (batched) thành một đồ thị lớn duy nhất để xử lý hiệu quả hơn trong mô hình học sâu. batched_graph là đồ thị lớn này.

Unbatch đồ thị: Việc tách batch thành các đồ thị ban đầu (từng đồ thị riêng lẻ) giúp thực hiện các phân tích cụ thể hoặc kiểm tra trên từng đồ thị.

```

1 batched_graph, labels = batch
2 print('Number of nodes for each graph element in the batch:', batched_graph.
      batch_num_nodes())
3 print('Number of edges for each graph element in the batch:', batched_graph.
      batch_num_edges())
4
5 # Recover the original graph elements from the minibatch
6 graphs = dgl.unbatch(batched_graph)
7 print('The original graphs in the minibatch:')
8 print(graphs)

```

```

Number of nodes for each graph element in the batch: tensor([5525, 1485, 1288, 812, 226])
Number of edges for each graph element in the batch: tensor([9439, 3409, 2928, 3858, 432])
The original graphs in the minibatch:
[Graph(num_nodes=5525, num_edges=9439,
      ndata_schemes={'feat': Scheme(shape=(200,), dtype=torch.float32)},
      edata_schemes={}), Graph(num_nodes=1485, num_edges=3409,
      ndata_schemes={'feat': Scheme(shape=(200,), dtype=torch.float32)},
      edata_schemes={}), Graph(num_nodes=1288, num_edges=2928,
      ndata_schemes={'feat': Scheme(shape=(200,), dtype=torch.float32)},
      edata_schemes={}), Graph(num_nodes=812, num_edges=3858,
      ndata_schemes={'feat': Scheme(shape=(200,), dtype=torch.float32)},
      edata_schemes={}), Graph(num_nodes=226, num_edges=432,
      ndata_schemes={'feat': Scheme(shape=(200,), dtype=torch.float32)},
      edata_schemes={})]

```

Ép kiểu sang mảng numpy

```

1 batched_graph.ndata["feat"]

        tensor([[-0.0085,  0.0039, -0.0473, ...,  0.0258,  0.0032,  0.0052],
       [-0.0146,  0.0070, -0.0405, ...,  0.0196,  0.0044,  0.0099],
       [-0.0033, -0.0016, -0.0198, ...,  0.0023,  0.0045,  0.0033],
       ...,
       [ 0.0000,  0.0000,  0.0000, ...,  0.0000,  0.0000,  0.0000],
       [ 0.0000,  0.0000,  0.0000, ...,  0.0000,  0.0000,  0.0000],
       [ 0.0000,  0.0000,  0.0000, ...,  0.0000,  0.0000,  0.0000]])
```

3. Xây dựng mô hình Graph Convolutional Network (GCN)

- **Class GCN:**

- Đây là một lớp biểu diễn mô hình GCN với hai tầng (layers) Graph Convolution.
- Các tham số chính:
 - * in_feats: Số lượng đặc trưng đầu vào của mỗi node.
 - * h_feats: Số lượng đặc trưng ẩn (hidden features) ở tầng giữa.
 - * num_classes: Số lớp đầu ra (ứng với các nhãn dự đoán).

- **Hai lớp Graph Convolution:**

- self.conv1: Lớp Graph Convolution đầu tiên, từ đầu vào in_feats sang h_feats.
- self.conv2: Lớp Graph Convolution thứ hai, từ h_feats sang num_classes.
- allow_zero_in_degree=True: Cho phép xử lý các node không có kết nối đầu vào.

```

1 from dgl.nn import GraphConv
2
3 class GCN(nn.Module):
4     def __init__(self, in_feats, h_feats, num_classes):
5         super(GCN, self).__init__()
6         self.conv1 = GraphConv(in_feats, h_feats, allow_zero_in_degree=True)
7         self.conv2 = GraphConv(h_feats, num_classes, allow_zero_in_degree=True)
8
9     def forward(self, g, in_feat):
10        h = self.conv1(g, in_feat)
11        h = F.relu(h)
12        h = self.conv2(g, h)
13        g.ndata['h'] = h
14        return dgl.mean_nodes(g, 'h')
```

```

1 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
2 model = GCN(200, 300, 2).to(device)
3 optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

```
4 | epochs = 1000
```

- Khởi tạo mô hình: Mô hình được tạo với các tham số đầu vào (số lượng đặc trưng của nodes), số lượng lớp ẩn, và số lượng lớp đầu ra (số lớp phân loại).
- Vòng lặp huấn luyện:
 - Trong mỗi epoch, một vòng lặp huấn luyện được thực hiện qua các batch dữ liệu từ train_dataloader.
 - Mỗi batch bao gồm các đồ thị (dưới dạng batched_graph) và nhãn của các nodes trong đồ thị.
 - Dữ liệu và mô hình được chuyển sang thiết bị (GPU hoặc CPU) tương ứng.
 - Dữ liệu đầu vào (features) được lấy từ batched_graph.ndata['feat'].
 - Mô hình thực hiện dự đoán (prediction) cho mỗi batch dữ liệu.
 - Hàm mất mát (loss) được tính bằng cross_entropy giữa dự đoán và nhãn thực tế.
 - Mô hình được tối ưu hóa qua việc tính toán gradient và cập nhật các tham số bằng cách sử dụng optimizer.step().
- Tính toán và hiển thị kết quả:
 - Tổng kết lại mất mát và độ chính xác của mô hình trong mỗi epoch.
 - Kết quả trung bình của mất mát (avg_loss) và độ chính xác huấn luyện (train_accuracy) được in ra mỗi epoch.
- Lưu lại checkpoint: Sau mỗi 100 epoch, trạng thái của mô hình và bộ tối ưu hóa được lưu lại dưới dạng checkpoint, cho phép tiếp tục huấn luyện từ điểm đã dừng mà không phải bắt đầu lại từ đầu.

Train mô hình

```

1 # Create the model with given dimensions
2 # model = GCN(dataset.dim_nfeats, 16, dataset.gclasses)
3     for epoch in range(epochs):
4         total_loss = 0
5         total_correct = 0
6         total_samples = 0
7
8         # Training loop
9         model.train()
10        for batched_graph, labels in train_dataloader:
11            pred = model(batched_graph, batched_graph.ndata['attr'].float())
12            batched_graph = batched_graph.to(device)
13            labels = labels.to(device)
14            feat = batched_graph.ndata['feat'].float().to(device)
15            pred = model(batched_graph, feat)
16            loss = F.cross_entropy(pred, labels)
17            optimizer.zero_grad()
18            loss.backward()
```

```
19     optimizer.step()

20

21     # Accumulate loss and accuracy
22     total_loss += loss.item()
23     total_correct += (pred.argmax(1) == labels).sum().item()
24     total_samples += len(labels)

25

26     # Calculate average loss and accuracy for the epoch
27     avg_loss = total_loss / len(train_dataloader)
28     train_accuracy = total_correct / total_samples

29

30     # Print the training metrics for each epoch
31     print(f"Epoch {epoch+1}/{epochs}, Loss: {avg_loss:.4f}, Accuracy: {
32         train_accuracy:.4f}")

33     if (epoch + 1) % 100 == 0:
34         checkpoint_path = os.path.join("/kaggle/working/", f'epoch_{epoch+1}.

35         pth')
36         torch.save({
37             'epoch': epoch + 1,
38             'model_state_dict': model.state_dict(),
39             'optimizer_state_dict': optimizer.state_dict(),
40             'loss': loss,
41         }, checkpoint_path)
42         print(f"Checkpoint saved to {checkpoint_path}")
```

```

Epoch 1/1000, Loss: 0.1117, Accuracy: 0.9562
Epoch 2/1000, Loss: 0.1042, Accuracy: 0.9626
Epoch 3/1000, Loss: 0.1130, Accuracy: 0.9569
Epoch 4/1000, Loss: 0.1012, Accuracy: 0.9633
Epoch 5/1000, Loss: 0.1140, Accuracy: 0.9562
Epoch 6/1000, Loss: 0.1039, Accuracy: 0.9668
Epoch 7/1000, Loss: 0.1090, Accuracy: 0.9576
Epoch 8/1000, Loss: 0.0990, Accuracy: 0.9633
Epoch 9/1000, Loss: 0.0968, Accuracy: 0.9654
Epoch 10/1000, Loss: 0.1099, Accuracy: 0.9562
Epoch 11/1000, Loss: 0.0970, Accuracy: 0.9689
Epoch 12/1000, Loss: 0.0964, Accuracy: 0.9619
Epoch 13/1000, Loss: 0.0924, Accuracy: 0.9640
Epoch 14/1000, Loss: 0.1020, Accuracy: 0.9647
Epoch 15/1000, Loss: 0.0903, Accuracy: 0.9703
Epoch 16/1000, Loss: 0.0985, Accuracy: 0.9696
Epoch 17/1000, Loss: 0.0953, Accuracy: 0.9626
Epoch 18/1000, Loss: 0.0866, Accuracy: 0.9661
Epoch 19/1000, Loss: 0.0922, Accuracy: 0.9640
Epoch 20/1000, Loss: 0.0901, Accuracy: 0.9641

```

Đánh giá mô hình

```

1   # Testing loop
2 model.load_state_dict(torch.load("/kaggle/working/epoch_100.pth", weights_only=
3     True) ['model_state_dict'])
4 model.eval()
5 num_correct = 0
6 num_tests = 0
7 # model.load
8 with torch.no_grad():
9     for batched_graph, labels in test_dataloader:
10         batched_graph = batched_graph.to(device)
11         labels = labels.to(device)
12         feat = batched_graph.ndata['feat'].float().to(device)
13         pred = model(batched_graph, feat)
14         pred = model(batched_graph, batched_graph.ndata['feat'].float())
15         num_correct += (pred.argmax(1) == labels).sum().item()
16         num_tests += len(labels)
17
18     # Calculate and print final test accuracy
19 test_accuracy = num_correct / num_tests
20 print('Test accuracy:', test_accuracy)

```

Test accuracy: 0.8309859154929577

Lưu lại mô hình

```
1 torch.save(model.state_dict(), "gcn_model_final.pth")
2 print("Model saved successfully.")
```

TÀI LIỆU THAM KHẢO

PHỤ LỤC

PHỤ LỤC A - MỘT SỐ QUY ĐỊNH

PHỤ LỤC B: ĐẶC TẢ USE CASE