ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA CÔNG NGHỆ PHẦN MỀM

PHẠM HỮU DANH

KHÓA LUẬN TỐT NGHIỆP

PHÁT HIỆN MÃ ĐỘC BẰNG PHƯƠNG PHÁP MÁY HỌC TRÊN HỆ ĐIỀU HÀNH WINDOWS

MALWARE DETECTION USING MACHINE LEARNING IN WINDOWS OPERATING SYSTEMS

KỸ SƯ NGÀNH KỸ THUẬT PHẦN MỀM

TP. HỒ CHÍ MINH, 2018

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA CÔNG NGHỆ PHẦN MỀM

PHẠM HỮU DANH - 14520134

KHÓA LUẬN TỐT NGHIỆP

PHÁT HIỆN MÃ ĐỘC BẰNG PHƯƠNG PHÁP MÁY HỌC TRÊN HỆ ĐIỀU HÀNH WINDOWS

MALWARE DETECTION USING MACHINE LEARNING IN WINDOWS OPERATING SYSTEMS

KỸ SƯ NGÀNH KỸ THUẬT PHẦN MỀM

GIẢNG VIÊN HƯỚNG DẪN
PGS. TS. VŨ THANH NGUYÊN

TP. HỒ CHÍ MINH, 2018

Lời Cảm Ơn

Đầu tiên và trên hết, tôi xin gửi lời cảm ơn chân thành nhất đến giảng viên, PGS. TS. Vũ Thanh Nguyên. Tôi chân thành biết ơn và cảm ơn thầy vì những chỉ dẫn vô giá, những hướng dẫn giá trị và những động lực từ tinh thần nghiên cứu liên tục của thầy. Những kinh nghiệm và sự giảng dạy của thầy trong lĩnh vực khóa học máy tính nói chung, và công nghệ phần mềm nói riêng, tôi có được những kiến thức mới, đầy giá trị và niềm đam mê trong lĩnh vực máy học và nhận dạng mã độc. Thầy là nguồn cảm hứng của tôi qua những trở ngại trong thời gian làm khóa luận tốt nghiệp. Nếu không có sự giúp đỡ của thầy, tôi sẽ không thể hoàn thành khóa luận này.

Tôi cũng gửi lời cảm ơn đến thầy Huỳnh Nguyễn Khắc Huy để về sự hướng dẫn của thầy về học máy và bảo mật máy tính.

Tôi cũng biết ơn tất cả các giảng viên của khoa Kỹ thuật Phần mềm tại trường Đại học Công nghệ Thông tin. Nhờ sự cống hiến của họ, tôi có một nền tảng vững chắc về khoa học máy tính nói chung và kiến thức về công nghệ phần mềm nói riêng. Nhờ các thầy cô, tôi học được khả năng học hỏi và áp dụng các công nghệ mới một cách hiệu quả hơn.

Cuối cùng nhưng không kém phần quan trọng, tôi xin cảm ơn gia đình đã khuyến khích và hỗ trợ tài chính lúc tôi cần. Tôi mãi mãi biết ơn tình yêu vô điều kiện của họ.

Tóm Tắt Khoá Luận

Phát hiện mã độc tĩnh là một lớp cần thiết trong các hệ thống bảo mật, nó cố gắng phân loại các mẫu là độc hại hoặc lành tính trước khi thực thi chúng. Tuy nhiên, hầu hết các nghiên cứu liên quan gặp các vấn đề về khả năng mở rộng, ví dụ, các phương pháp sử dụng neural networks thường mất rất nhiều thời gian đào tạo [41] hoặc sử dụng các tập dữ liệu mất cân bằng [45, 54], làm cho các số liệu đánh giá bị gây hiểu nhầm trong thực tế.

Trong nghiên cứu này, chúng tôi tìm hiểu về cả hai phương pháp nhận diện mã độc trên hệ điều hành Windows: phát hiện mã độc tính (static malware detection) và phát hiện mã độc động (dynamic malware detection). Chúng tôi tiến hành các thí nghiệm để áp dụng các phương pháp học máy trong phát hiện mã độc tĩnh. Hơn nữa, chúng tôi đề xuất phương pháp phát hiện phần mềm độc hại tĩnh sử dụng Portable Executable analysis và thuật toán học máy Gradient Boosting Decision Tree. Chúng tôi tập trung vào việc giảm thời gian đào tạo bằng cách giảm số lượng tính năng một cách thích hợp. Các kết quả thử nghiệm cho thấy phương pháp đề xuất của chúng tôi có thể đạt được tỷ lệ phát hiện lên tới 99,394 % với tỷ lệ cảnh báo sai là 1%, và khi giới hạn tỷ lệ báo động giả dưới 0,1% thì tỷ lệ phát hiện là 97,572 %, dựa trên hơn 600.000 mẫu đào tạo và 200.000 mẫu kiểm thử từ bộ dữ liệu Endgame Malware BEnchmark for Research (EMBER) [3].

Mục lục

Danh sách hình vẽ

Danh sách bảng

Chương 1

Đặt Vấn Đề

Tóm tắt chương

Chương 1 đặt vấn đề cho đề tài, bao gồm sự nguy hiểm của phần mềm độc hại, tổng quan về tình hình nghiên cứu, ứng dụng các phương pháp máy học vào nhận diện mã độc, và đặc biệt là trình bày những hạn chế của các phương pháp đã có, cũng chính là lí do của việc chọn đề tài.

Malware thường được sử dụng như một thuật ngữ chung để chỉ bất kỳ phần mềm nào được thiết kế nhằm mục đích gây thiệt hại cho một máy tính, máy chủ hoặc mạng máy tính [37]. Một sự cố từ malware có thể gây thiệt hại hàng triệu đô la, cụ thể, mã độc zero-day ransomware WannaCry đã gây ra thảm họa trên toàn thế giới từ việc đánh sập hệ thống của các Bệnh viện Dịch vụ Y tế Quốc gia Vương quốc Anh, cho đến việc tắt toàn bộ hệ thống sản xuất của công ty Honda tại Nhật Bản [11]. Hơn nữa, phần mềm độc hại ngày càng trở nên tinh vi và đa dạng hơn mỗi ngày [51]. Theo đó, bài toán phát hiện phần mềm độc hại là một vấn đề quan trọng trong an ninh mạng, đặc biệt khi xã hội trở nên phụ thuộc nhiều vào các hệ thống máy tính.

Các sản phẩm nhận diện malware trước đó thường sử dụng các phương pháp dựa trên quy tắc (rule-based) hoặc dựa trên chữ ký (signature-based), yêu cầu các nhà phân tích xử lý các quy tắc thủ công (handcraft rules) có liên quan để phát hiện mã

độc. Cách tiếp cận này có độ chính xác cao. Tuy nhiên, các quy tắc này thường là cụ thể và không thể nhận ra phần mềm độc hại mới, ngay cả khi nó sử dụng cùng chức năng. Vì lý do này, ý tưởng phát hiện mã độc dựa trên các thuật toán học máy được phát sinh. Thuật toán học máy học các mẫu cơ bản (pattern) từ một tập huấn luyện nhất định, bao gồm cả các mẫu độc hại và lành tính. Những pattern phân biệt các phần mềm độc hại từ phần mềm lành tính. Kể từ khi Schultz và các cộng sự chó thấy sự hiệu quả của việc ứng dụng các thuật toán máy học vào nhận diện mã độc [47], máy học đang trở thành một trong những công cụ phổ biến và có nhiều ảnh hưởng trong việc đảm bảo an toàn cho hệ thống.

Một số phương pháp sử dụng học máy đã mang lại các mô hình quá tích cực, thể hiện độ chính xác dự báo đáng kể, nhưng đã dẫn đến nhiều những kết quả dương tính giả (false positives). Các kết quả dương tính giả làm tiêu cực trải nghiệm của người dùng, ngăn không cho triển khai các phần mềm mới. Theo khảo sát của các IT administrator năm 2017 [15], 42% các công ty cho rằng người dùng của họ bị mất năng suất là do liên quan đến những kết quả dương tính giả, tạo ra một điểm nghẹt cho các quản trị viên CNTT trong môi trường doanh nghiệp. Các kỹ sư bảo mật cũng thông báo các báo động giả này thường gây rối khi họ đang làm việc để phát hiện và loại bỏ phần mềm độc hại. Một báo cáo được công bố vào năm 2015 cũng cho thấy rằng nhiều tổ chức ở Hoa Kỳ đã tiêu thụ một lượng tiền khổng lồ để xử lý các cảnh báo phần mềm độc hại không chính xác [29]. Do đó, ngay cả khi một giải pháp có tỷ lệ phát hiện cao nhất, nếu nó có một số lượng lớn các kết quả dương tính giả, nó được xem là vô dụng hơn so với một giải pháp với các kết quả dương tính giả thấp và một tỷ lệ phát hiện vừa phải.

Chúng tôi chọn đề tài "Phát hiện Mã độc nằng phương pháp Máy học trên hệ điều hành Windows"với mong muốn đóng góp một phương pháp mới để giải quyết vấn đề xác định phần mềm độc hại, đạt được tỷ lệ phát hiện cao và tỷ lệ dương giả thấp.

Chương 2

Giới Thiệu Tổng Quan

Tóm tắt chương

Chương 2 trình bày tổng quan về việc áp dụng Học máy trong Phát hiện mã độc tĩnh trên các hệ điều hành Windows; động lực và mục tiêu của nghiên cứu này; liệt kê các nghiên cứu có liên quan của tác giả ở Việt Nam hoặc trên toàn thế giới.

2.1 Tổng quan

Phần mềm độc hại là phần mềm được thiết kế để xâm nhập hoặc gây hại cho hệ thống máy tính mà không có sự đồng ý của chủ sở hữu. Các phân loại đơn giản phần mềm độc hại là nhận dạng tệp phần mềm độc hại và tệp sạch. Phát hiện phần mềm độc hại tĩnh đang phân loại các mẫu là độc hại hoặc lành tính mà không cần thực thi chúng. Ngược lại, phát hiện phần mềm độc hại động phát hiện phần mềm độc hại dựa trên hành vi thời gian chạy của phần mềm độc hại [4, 16]. Mặc dù phát hiện phần mềm độc hại tĩnh được biết đến là không thể giải quyết được toàn bộ bài toán nhận dạng mã độc [13], nó là một lớp quan trọng trong một bộ bảo mật bởi vì khi thành công, nó cho phép xác định các tập tin độc hại trước khi thực thi.

Bên cạnh đó, máy học là một công cụ hấp dẫn cho một khả năng phát hiện các mẫu mới và khả năng phát hiện heuristic dựa trên mẫu đã có (phần 3.3.1). Các mô hình học tập được giám sát (Supervised learning models) sẽ tự động tìm ra mối quan hệ phức tạp giữa các thuộc tính tệp trong dữ liệu huấn luyện và phân biệt giữa các mẫu độc hại và lành tính (phần 3.3.2). Hơn nữa, các mô hình học máy sẽ khái quát hóa với tập dữ liệu mới có các tính năng và nhãn theo một mẫu tương tự với dữ liệu đào tạo.

Ngoài ra, trong các hệ điều hành Windows, định dạng chung cho phần mềm độc hại là định dạng Portable Executable (PE) (phần 3.2), đó là định dạng tệp cho các tệp thi hành, mã đối tượng, tệp DLL, tệp phông chữ FON và các tệp khác được sử dụng trong cả phiên bản 32 bit và 64 bit. Định dạng PE đóng gói thông tin cần thiết cho trình tải hệ điều hành Windows để quản lý mã thực thi được bao bọc.

Do đó, nhiều phương pháp phát hiện phần mềm độc hại PE dựa vào các phương pháp học máy đã được đề xuất bắt đầu từ năm 1995 [27, 47, 28, 49, 45]. Năm 2001, Schultz và cộng sự biểu diễn các tệp PE theo các tính năng bao gồm imported functions, strings và byte sequences [47]. Các mô hình bao gồm các quy tắc được tạo ra từ RIPPER [14], Naive Bayes và một ensemble classifier. Phương pháp này đã được mở rộng bởi Kolter et al. vào năm 2006 [28] by cách sử dụng byte-level N-grams và những kỹ thuật từ xử lý ngôn ngữ tự nhiên, bao gồm trọng số TFIDF cho các strings. Vào năm 2009, Shafiq và các cộng sự đề xuất chỉ sử dụng bảy tính năng từ PE header, do thực tế là hầu hết các ứng dụng phần mềm độc hại trong nghiên cứu của họ thường trình bày các yếu tố đó [49]. Năm 2015, Saxe và Berlin sử dụng two-dimensional byte entropy histograms và một multi-layer neural network cho việc nhận dạng mã độc [45].

2.2 Động lực

Mặc dù nhiều mô hình đã thực hiện độ chính xác dự báo nổi bật, chúng được huấn luyện và xác thực trên tập dữ liệu mất cân bằng vì bài toán phát hiện phần mềm độc hại không nhận được sự chú ý tương tự như các bài toán khác trong cộng đồng nghiên cứu mở. Những hạn chế pháp lý là thách thức chính cho việc công bố tập dữ liệu điểm chuẩn để phát hiện phần mềm độc hại, cụ thể, Saxe và Berlin không thể công bố dữ liệu hoặc mã nguồn cho dự án của họ do tính chất pháp lý và độc quyền của nó [45]. Ngoài ra, không giống như hình ảnh, văn bản và lời nói có thể được gắn nhãn gần như ngay lập tức và trong nhiều trường hợp không cần chuyên gia, việc phân loại tệp độc hại hay lành tính là quá trình tốn thời gian cho cả những chuyên viên được đào tạo tốt. Công việc ghi nhãn có thể được tự động thông qua phần mềm anti-malware, nhưng kết quả có thể là độc quyền hoặc được bảo vệ pháp lý khác. Do đó, các mô hình này có thể không gây ấn tượng với dữ liệu cân bằng [10].

Hơn nữa, một phương pháp hiệu quả, có độ chính xác ấn tượng và tỷ lệ dương tính giả rất thấp, sẽ ngăn chặn những tổn thất lớn từ phần mềm độc hại, cũng như mang lại trải nghiệm tốt cho người dùng và tiết kiệm tài nguyên cho nhiều tổ chức.

2.3 Mục tiêu

Như đã đề cập trong phần 2.2, kết quả từ nhiều phương pháp được đề xuất có thể không ấn tượng khi đánh giá với dữ liệu cân bằng. Vì vậy mục tiêu chính của luận án này là **áp dụng các phương pháp học máy trong phát hiện phần mềm độc hại với tập dữ liệu cân bằng**. Ngoài ra, mục tiêu dự kiến là đề xuất một phương pháp với **tỷ lệ phát hiện cao** và **tỷ lệ báo động giả rất thấp**. Cuối cùng, chúng tôi hướng đến kiến thức về học máy và cách áp dụng chúng để tăng cường bảo mật thông tin người dùng trong việc giải quyết vấn đề phát hiện phần mềm độc hại.

Công việc chi tiết mà chúng tôi đã thực hiện trong luận án này bao gồm:

- Nghiên cứu và xây dựng nền tảng về bảo mật thông tin, đặc biệt là phần mềm độc hại.
- Nghiên cứu và hiểu các kiến thức cơ bản trong Học máy bao gồm phân loại, thuật toán cây quyết định, thuật toán Random Forest, thuật toán Support Vector Machine, Neural Networks và Gradient-Boosting Decision Tree.
- Nghiên cứu và hiểu rõ hơn về thuật toán cây Gradient-Boosting Decision Tree.
- Áp dụng thuật toán Gradient-Boosting Decision Tree vào bài toán phát hiện phần mềm độc hại. Tiến hành các thí nghiệm để đánh giá hiệu suất của mô hình và tối ưu hóa các tham số.

2.4 Những nghiên cứu liên quan

Phát hiện phần mềm độc hại đã phát triển trong vài năm qua, do mối đe dọa ngày càng tăng gây ra bởi phần mềm độc hại cho các doanh nghiệp lớn và các cơ quan chính phủ. Tại Việt Nam, PGS.TS. Vũ Thanh Nguyên và cộng sự đề xuất một phương pháp kết hợp của thuật toán lựa chọn tiêu cực và mạng miễn dịch nhân tạo để phát hiện virus [39], và phương pháp phát hiện phần mềm độc hại biến chất bằng phân tích Portable Executable (PE) cùng Longest Common Sequence (LCS) [54]. Nguyen Van Nhuong và các cộng sự đã đề xuất phương pháp semantic để phát hiện phần mềm độc hại biến chất một cách hiệu quả [52].

Các phương pháp phát hiện phần mềm độc hại có thể được phân loại trong phát hiện phần mềm độc hại tĩnh hoặc phát hiện phần mềm độc hại động [18]. Về lý thuyết, tính năng phát hiện động cung cấp chế độ xem trực tiếp về hành động của phần mềm độc hại, ít bị ảnh hưởng hơn bởi việc làm xáo trộn mã thực thi và làm cho việc sử dụng lại phần mềm độc hại trở nên khó khăn hơn [38]. Tuy nhiên, trên thực tế, phần mềm độc hại có thể xác định liệu nó có đang chạy trong một hộp cát, và ngăn chính nó thực hiện hành vi nguy hiểm [53]. Điều này dẫn đến một cuộc chạy đua vũ trang

giữa các phương pháp phát hiện phần mềm độc hại động và phần mềm độc hại. Hơn nữa, trong nhiều trường hợp, phần mềm độc hại không hoạt động một cách chính xác do thiếu dependency hoặc cấu hình hệ thống không mong muốn. Những vấn đề này gây khó khăn cho việc thu thập tập dữ liệu về hành vi phần mềm độc hại.

Ngược lại, phân tích tĩnh không yêu cầu các thiết lập phức tạp, tốn kém để thu thập, nó có các tập dữ liệu khổng lồ có thể được tạo bằng cách tổng hợp các tệp nhị phân. Điều này làm cho phát hiện phần mềm độc hại tĩnh rất phổ biến với phương pháp học máy, có xu hướng hoạt động tốt hơn khi tăng kích thước dữ liệu [5]. Một số chương trình phát hiện phần mềm độc hại dựa trên học máy đã được giới thiệu từ ít nhất năm 1995 [27, 47, 28, 49, 45]. Nhiều tính năng tĩnh đã được đề xuất để trích xuất các tính năng từ tệp nhị phân: printable strings [47], import tables, opcodes, informational entropy, [55], byte n-grams [1], two dimensional byte entropy histograms [45]. Nhiều tính năng khác cũng đã được đề xuất trong Microsoft Malware Classification Challenge trên Kaggle [43], như là opcode images, various decompiled assembly features, và các thống kê tổng hợp. Ngoài ra, lấy cảm hứng từ sự thành công của các mô hình học tập sâu rộng đến đầu cuối trong xử lý hình ảnh và xử lý ngôn ngữ tự nhiên, Raff et al. giới thiệu phát hiện phần mềm độc hại từ raw byte sequences [41]. Có lẽ các mô hình state-of-the-art sẽ thay đổi thành các mô hình học sâu end-to-end trong những tháng hoặc năm tiếp theo, nhưng các hand-crafted features có thể tiếp tục có liên quan do định dạng có cấu trúc của phần mềm độc hại.

Không gian tính năng có thể trở nên lớn, trong trường hợp đó các phương pháp như locality-sensitive hashing [6], feature hashing [22] hoặc random projections [19]đã được áp dụng trong phát hiện phần mềm độc hại. Tuy nhiên, ngay cả sau khi áp dụng việc giảm kích thước, vẫn còn một số lượng lớn các tính năng, có thể gây ra các vấn đề về khả năng mở rộng cho một số thuật toán học máy. Neural networks đã nổi lên như một giải pháp thay thế có thể mở rộng do những tiến bộ đáng kể trong thuật toán đào tạo [2]. Nhiều phương pháp sử dụng neural networks đã được giới thiệu [45, 16, 27, 7] mặc dù không có cách so sánh kết quả rõ ràng vì các tập dữ liệu khác nhau.

Một lựa chọn phổ biến khác là ensemble of trees, có thể mở rộng hợp lý một cách hợp lý bằng cách lấy mẫu không gian tính năng trong mỗi lần lặp [9]. Cây quyết định có thể thích ứng tốt với các loại dữ liệu khác nhau và linh hoạt với nhiều tỷ lệ giá trị trong vectơ đặc trưng, vì vậy chúng mang lại hiệu suất tốt ngay cả khi không có tiêu chuẩn hóa dữ liệu.

Chương 3

Lý Thuyết

Tóm tắt chương

Đầu tiên, chương 3 trình bày các nền tảng lý thuyết được sử dụng trong đề tài, bao gồm các loại malware, PE file format và các kiến thức lý thuyết khác về học máy. Sau đó, các thước đo đánh giá và framework sử dụng cho thuật toán học máy đã được sử dụng trong việc implement thuật toán sẽ được giới thiệu.

3.1 Malware Types

Phân loại là cách tuyệt vời để hiểu rõ hơn về phần mềm độc hại. Một số loại phổ biến nhất bao gồm: adware, bots, rootkits, spyware, Trojan horses, viruses, and worms [17].

3.1.1 Virus

Virus là một dạng phần mềm độc hại có khả năng sao chép chính nó và lây lan sang các máy tính khác bằng cách đính kèm vào các ứng dụng khác nhau và thực thi khi người dùng khởi chạy một trong số đó. Virus cũng có thể lây lan qua các tài liệu, tệp kịch

bản và lỗ hổng tập lệnh script (cross-site scripting vulnerabilities) trong các ứng dụng web. Một số ví dụ nổi tiếng của virus trong những năm qua là virus Concept, virus Chernobyl (còn được gọi là CIH), virus Anna Kournikova, Brain và RavMonE.exe.

3.1.2 Worm

Worm là một phần mềm độc lập sao chép mà không cần nhắm mục tiêu và lây nhiễm các tệp cụ thể. Hãy nghĩ về worm như các chương trình nhỏ tự sao chép bản thân và phá hủy dữ liệu. Nó thường nhắm vào các tập tin hệ điều hành và làm việc cho đến khi ổ đĩa trở nên trống rỗng. Một số ví dụ bao gồm Melissa, Morris, Mydoom, Sasser và Blaster.

3.1.3 Trojan

Trojan là một ứng dụng độc hại giả dạng bản thân để trông hữu ích và đánh lừa người dùng tải xuống và cài đặt. Trojan có thể cung cấp quyền truy cập từ xa vào một máy tính bị nhiễm để kẻ tấn công có thể lấy cắp dữ liệu, cài đặt thêm phần mềm độc hại, theo dõi hoạt động của người dùng, v.v. Các ví dụ đáng chú ý cũng bao gồm những con Trojan được phát triển bởi các cơ quan chính phủ Hoa Kỳ như FBI và NSA. Những cái tên như Magic Lantern, FinFisher, Netbus, Beast, Gh0st RAT, Clickbot.A, và Zeus đã trở thành lý do của kinh sự kinh hoàng. Tương tự một trojan trên Android được phát hiện vào năm 2015, tên là Shedun, là một trong nhiều malware nhắm đến mục tiêu là thiết bị di động.

3.1.4 Ransomware

Ransomware, một trong những phần mềm độc hại nhất và liên tục xuất hiện, là một loại phần mềm độc hại thường giam giữ một hệ thống máy tính và yêu cầu một khoản tiền chuộc, ví dụ như chặn truy cập vào dữ liệu của nạn nhân hoặc đe dọa công khai nội dung của nó. Tệ hơn nữa, không có gì đảm bảo rằng việc thanh toán sẽ nhận được

quyền truy cập vào dữ liệu hoặc ngăn không cho công khai dữ liệu nhạy cảm. Các ransomware nổi tiếng như Reveton, CryptoLocker, CryptoWall, và gần đây hơn, cuộc tấn công WannaCry năm 2017, đã gây ra không một lượng không nhỏ tổn hại [11].

3.1.5 Rootkit

Rootkit là một tập hợp các phần mềm được thiết kế đặc biệt để cho phép phần mềm độc hại xâm nhập hệ thống của bạn và thu thập thông tin. Những công việc này ở chế độ nền để người dùng có thể không nhận thấy bất kỳ điều gì khác biệt. Nhưng bên trong môi trường, một rootkit sẽ cho phép một số loại phần mềm độc hại xâm nhập vào hệ thống. Rootkit đầu tiên có được danh tiếng trên Windows là NTRootkit vào năm 1999, nhưng phổ biến nhất là vụ bê bối Sony BMG copy protection rootkit scandal đã làm rung chuyển công ty trong năm 2005 [46].

3.1.6 Adware

Mặc dù phần mềm hỗ trợ quảng cáo (adware) hiện phổ biến hơn nhiều, adware đã được liên kết với phần mềm độc hại trong một thời gian dài. Trong khi phần mềm quảng cáo có thể tham chiếu đến bất kỳ ứng dụng nào được quảng cáo hỗ trợ, phần mềm quảng cáo độc hại thường hiển thị quảng cáo dưới dạng cửa sổ bật lên và ngăn cửa sổ không thể đóng. Đó có lẽ là phần mềm độc hại hiệu quả nhất và ít nguy hiểm nhất, được thiết kế với mục đích cụ thể là quảng bá quảng cáo trên máy tính của bạn.

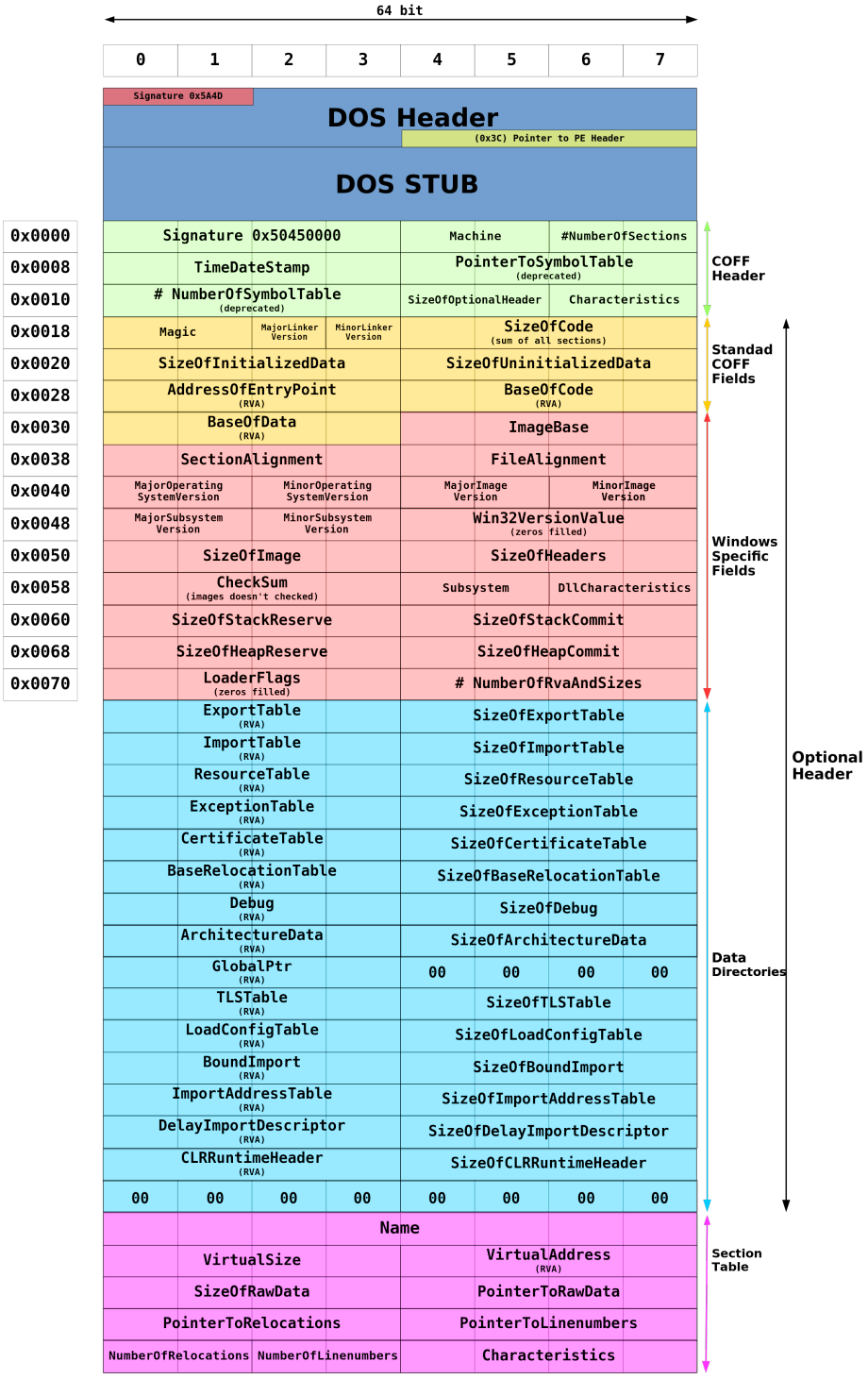
3.1.7 Bot

Bots là các chương trình phần mềm được tạo ra để thực hiện các hoạt động cụ thể một cách tự động. Trong khi một số bot được tạo ra cho mục đích vô hại, nó ngày càng trở nên phổ biến để xem bot đang được sử dụng độc hại. Bots có thể được sử dụng trong botnet (tập hợp các máy được kiểm soát bởi các bên thứ ba) để thực hiện các tấn công từ chối dịch vụ phân tán, gửi spam và ăn cắp dữ liệu.

3.2 PE File Format

Định dạng tệp Portable Executable (PE) mô tả định dạng thực thi chiếm ưu thế cho hệ điều hành Microsoft Windows và bao gồm tệp thực thi, thư viện liên kết động (DLL) và tệp phông chữ FON. Định dạng này hiện được hỗ trợ trên Intel, AMD và các biến thể của kiến trúc bộ lệnh ARM.

Tệp PE bao gồm một số header và section mô tả dynamic linker biết cách ánh xạ tệp vào bộ nhớ. Một tệp thực thi bao gồm một số vùng khác nhau, mỗi vùng đòi hỏi sự bảo vệ bộ nhớ khác nhau; do đó, bắt đầu của mỗi phần phải được căn chỉnh với một khung trang. Thông thường, các header bao gồm Common Object File Format (COFF) file header chứa các thông tin cần thiết như machine type, file type (DLL, EXE, OBJ), số lượng sections, số lượng symbols, v.v. Optional header xác định linker version, kích thước của code, kích thước của initialized data và uninitialized data, entry point address, v.v. Các data directory nằm optional header cung cấp con trỏ đến section chứa nó. Những section này bao gồm tables for exports, imports, resources, exceptions, debug information, certificate information, và relocation tables. Do đó, định dạng PE cung cấp một bản tổng hợp các thông tin hữu ích của một tệp thực thi [50].



Hình 3.1 Kết cấu của tệp Portable Executable 32-bit [60]

Các section của PE chứa code và initialized data mà Windows loader sẽ ánh xạ vào vùng thực thi hoặc vùng bộ nhớ đọc/viết, cũng như là imports, exports, và resources được định nghĩa trong tệp. Mỗi section chưa một header sẽ chỉ định kích cỡ và địa chỉ. Import address table chỉ thị loader function nào sẽ được import tĩnh.Resources section có thể chứa các resource cần thiết cho người dùng như là: cursors, fonts, bitmaps, icons, menus, v.v. Một tập PE cơ bản thường sẽ có `.text` code section và một hoặc nhiều data section (`.data`, `.rdata` hoặc `.bss`). Relocation tables thường được chứa trong `.reloc` section, và được sử dụng bởi Windows loader để reassign base address từ the executable's preferred base. Section `.tls` thường chứa special thread local storage (TLS) structure cho việc luuaw trữ các biến riêng cho thread. Các section name được gọi ngẫu nhiên từ phía Windows loader, nhưng các tên cụ thể đã được chấp nhận bởi tiền lệ và phổ biến rộng rãi.

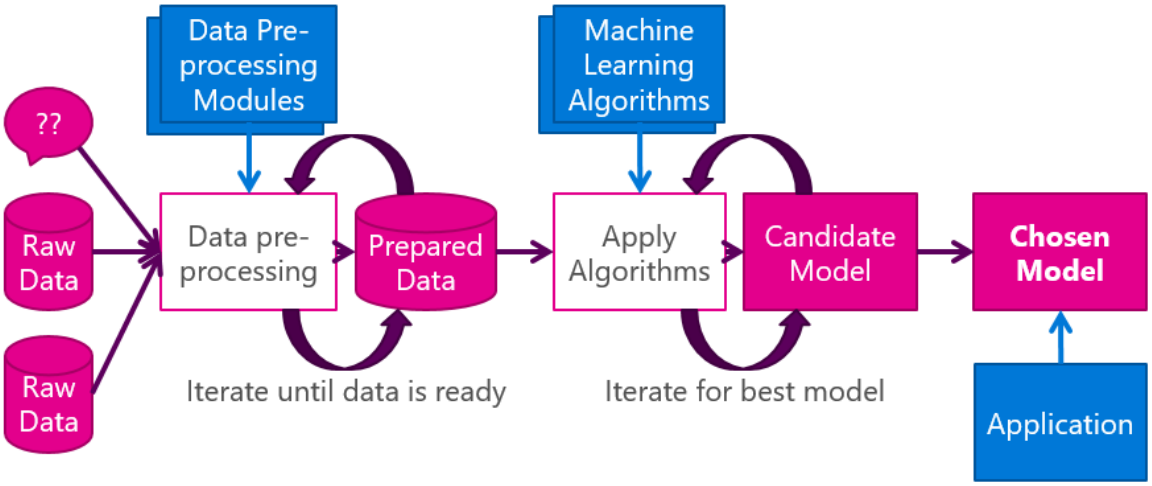
3.3 Machine Learning

3.3.1 Tổng quan

Trong những năm gần đây, hầu hết các nghiên cứu và tiến bộ sản xuất đều xuất phát từ tiểu ngành cuar Trí tuệ nhân tạo mang tên Machine Learning. Nguyên tắc Machine Learning rất đơn giản; Machine Learning là một phương pháp mà máy tính tìm thấy các pattern từ dữ liệu và đưa các pattern đó vào các ứng dụng. Sau đó, ứng dụng có thể có được thông tin chi tiết về dữ liệu mới dựa trên sự giống nhau với các pattern được xác định [26].

Hãy xem xet các công việc chung (Hình 3.2) của một quy trình học máy:

- Mục tiêu chính của quá trình này là xác định một mô hình. Mô hình là điều chính mà ứng dụng có thể gửi yêu cầu để có được thông tin chi tiết về dữ liệu mới.



Hình 3.2 Quy trình học máy [26]

- Trước khi đi vào thử nghiệm Machine Learning, chúng ta phải xác định mục đích và cách đánh giá kết quả.
- Quá trình bắt đầu với việc chuẩn bị dữ liệu. Dữ liệu được chuẩn bị là một hoặc nhiều tập dữ liệu đã được xử lý trước (định dạng, làm sạch và lấy mẫu) để sẵn sàng áp dụng thuật toán Machine Learning. Chuẩn bị dữ liệu có nghĩa là làm cho dữ liệu có hình dạng tốt nhất để rút ra kết luận khoa học.
- Bước tiếp theo là áp dụng một hoặc nhiều thuật toán Máy học tạo ra một Mô hình, đó là một quá trình lặp lại và chúng ta có thể lặp lại việc kiểm tra các thuật toán khác nhau cho đến khi chúng ta đạt được một mô hình đủ để đạt được mục đích.

Không phải tất cả các vấn đề đều là có thể áp dụng giải pháp học máy. Vấn đề này phải là một vấn đề có thể được giải quyết bằng dữ liệu, và có đủ số lượng dữ liệu có liên quan để được sử dụng. Như chúng ta sẽ thấy, nhiều vấn đề bảo mật phù hợp với yêu cầu này cực kỳ tốt.

3.3.2 Supervised Learning

Các thuật toán học được giám sát (supervised learning) đưa ra dự đoán dựa trên một tập hợp mẫu, ví dụ: giá cổ phiếu trong lịch sử có thể được sử dụng để đoán giá tương lai. Thuật toán học được giám sát tìm kiếm các pattern trong dữ liệu được gắn nhãn. Nó có thể sử dụng bất kỳ thông tin nào có thể có liên quan và mỗi thuật toán sẽ tìm các loại pattern khác nhau. Sau khi thuật toán đã phát hiện pattern tốt nhất có thể, nó sử dụng pattern đó để đưa ra các dự đoán cho dữ liệu không được dán nhãn.

Khi dữ liệu đang được sử dụng để dự đoán danh mục, học tập được giám sát được gọi là phân loại, ví dụ: chỉ định hình ảnh làm hình ảnh của chú mèo hoặc chó. Khi chỉ có hai lựa chọn, nó được gọi là phân loại hai lớp, nhị thức hoặc nhị phân. Khi có nhiều danh mục hơn, vấn đề này được gọi là phân loại nhiều lớp.

3.3.3 Feature Extraction

Như đã đề cập trong phần 3.3.1,chúng ta nên trích xuất các thuộc tính từ dữ liệu đầu vào để chúng ta có thể đưa nó vào thuật toán. Ví dụ, trong các trường hợp hình ảnh, dữ liệu có thể được biểu diễn dưới dạng giá trị RGB của mỗi pixel.

Các thuộc tính như vậy được gọi là **các đặc trưng**, và ma trận được gọi là vector đặc trưng. Quá trình trích xuất đặc trưng từ các tệp gọi là feature extraction. Mục đích của việc feature extraction là để có được một tập hợp dữ liệu nhiều thông tin và không dư thừa.

Các đặc trưng phải thể hiện thông tin cần thiết và có liên quan về tập dữ liệu của chúng ta vì chúng ta không thể đưa ra dự đoán chính xác mà không có nó. Đó là lí do tại sao feature extraction thường là một nhiệm vụ không rõ ràng và phụ thuộc vào lĩnh vực cụ thể, đòi hỏi nhiều thử nghiệm và nghiên cứu.

Một yêu cầu quan trọng khác đối với một bộ đặc trưng phù hợp là không dư thừa. Việc xuất hiện các đặc trưng dư thừa, cụ thể là các yếu tố nằm ngoài vùng thông tin

hoặc các yếu tố không liên quan, có thể làm cho thuật toán thiên vị, theo đó là việc cung cấp kết quả không chính xác.

3.3.4 Classification, Regression và Thresholding

Classification là công việc xấp xỉ hàm ánh xạ (f) từ các biến đầu vào (X) để đưa ra biến đầu ra rời rạc (y). Các biến đầu ra thường được gọi là nhãn hoặc danh mục. Hàm ánh xạ dự đoán lớp hoặc nhóm cho một quan sát đã cho. Ví dụ: một email có thể được phân loại là thuộc một trong hai loại: "spam"và "không phải spam".

Regression là công việc xấp xỉ hàm ánh xạ (f) từ các biến đầu vào (X) để đưa ra biến đầu ra liên tục (y). Biến đầu ra là giá trị thực, chẳng hạn như giá trị số nguyên hoặc số thực. Đây thường là số định lượng, chẳng hạn như số lượng và kích cỡ. Ví dụ: một căn nhà có thể được dự đoán sẽ bán cho một giá trị đô la cụ thể, có thể trong phạm vi từ 100,000 đô la đến 200.000 đô la.

Các vấn đề classification thường khác các vấn đề regression. Classification là công việc dự đoán nhãn lớp rời rạc, còn regression là công việc dự đoán số lượng liên tục. Tuy nhiên, có một số chồng chéo giữa các thuật toán để phân loại và hồi quy. Ví dụ, chúng ta có thể sử dụng xác suất, được trả về từ regression, trực tiếp hoặc chuyển nó thành một giá trị nhị phân. Để ánh xạ một giá trị logistic regression sang danh mục nhị phân, chúng ta phải định nghĩa một **classification threshold** (còn gọi là decision threshold). Có thể giả định rằng classification threshold luôn là 0,5, nhưng threshold phụ thuộc vào các vấn đề cụ thể và do đó đây là giá trị mà chúng ta phải điều chỉnh.

3.3.5 Ensemble, Bagging và Boosting

Khi chúng ta cố gắng dự đoán biến mục tiêu sử dụng bất kỳ phương pháp học máy nào, nguyên nhân hàng đầu của sự khác biệt trong giá trị ban đầu và được dự đoán là noise, variance, và bias. Ensemble giúp giảm hai yếu tố phía sau.

Một ensemble là một tập hợp các mô hình dự đoán để cùng đưa ra dự đoán cuối cùng. Lý do chính là nhiều mô hình dự đoán khác nhau cố gắng dự đoán cùng một

biến mục tiêu sẽ thực hiện một công việc tốt hơn so với bất kỳ một đơn lẻ nào. Kỹ thuật ensemble được phân loại thêm vào Bagging và Boosting.

Bagging là một kỹ thuật ensemble đơn giản bằng cách xây dựng nhiều mô hình độc lập và kết hợp chúng bằng những phương pháp lấy trung bình (ví dụ như, lấy trọng số trung bình, đa số phiếu bình chọn hoặc lấy trung bình bình thường). Chúng tôi thường sử dụng tập nhỏ ngẫu nhiên của dữ liệu cho mỗi mô hình, để tất cả các mô hình có chút khác biệt với nhau. Mỗi quan sát có cùng xác suất xuất hiện trong tất cả các mô hình. Bởi vì kỹ thuật này dùng nhiều dự báo không tương quan để tạo ra một mô hình cuối cùng, nó làm giảm lỗi bằng cách giảm variance. Một ví dụ nổi tiếng của kỹ thuật bagging ensemble là Random Forest (được đề cập ở phần 3.4.2).

Boosting là một kỹ thuật ensemble technique mà các mô hình được xây dựng một cách tuần tự. Kỹ thuật này áp dụng logic trong đó mô hình dự báo phía sa học hỏi từ những sai lầm của những mô hình trước đó. Theo đó, các quan sát có xác suất không đồng đều xuất hiện trong các mô hình tiếp theo. Mô hình dự báo có thể chọn từ nhiều thuật toán như decision trees, regressors, classifiers, v.v. Bởi vì các mô hình phía sau học tập từ những lỗi sai của mô hình phía trước, nó sử dụng ít số iteration để đạt được độ chính xác đã có. Những chúng ta phải chọn điểm dừng phù hợp, hoặc nó có thể dẫn đến việc overfitting trên tập dữ liệu huấn luyện. Gradient Boosting Decision Tree, được đề cập trong phần 3.4.3, là một ví dụ của kỹ thuật này.

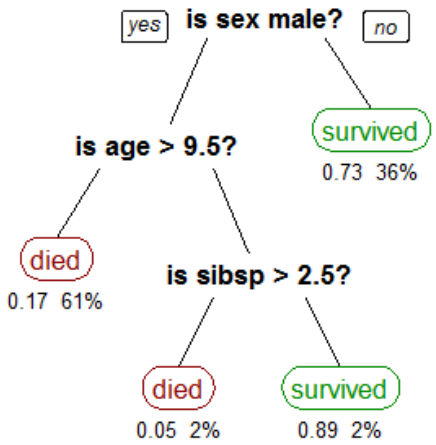
3.4 Các phương pháp Machine Learning

3.4.1 Decision Tree

Ngay từ tên của thuật toán, cây quyết định là cấu trúc dữ liệu có cấu trúc của cây. Bộ dữ liệu đào tạo được sử dụng để tạo ra cây, cây này được sử dụng để đưa ra các dự đoán về dữ liệu thử nghiệm. Trong thuật toán này, mục đích là đạt được kết quả chính xác nhất với số lượng ít nhất các quyết định phải được đưa ra. Chúng ta có thể sử dụng cây quyết định có thể cho cả vấn đề phân loại và hồi quy. Bảng 3.1 đưa ra một ví dụ.

Bảng 3.1 Một ví dụ về Decision Tree

Id	Name	Sex	Age	SibSp	Survived
1	Braund, Mr. Owen Harris	male	22.0	1	0
2	Cumings, Mrs. John Bradley	female	38.0	1	1
3	Heikkinen, Miss. Laina	female	26.0	0	1
...	...	...	...	...	...



Hình 3.3 Một ví dụ về Decision Tree [59]

Trong Hình 3.3, mô hình được đào tạo dựa trên tập dữ liệu và bây giờ có thể phân loại hành khách trong Titanic là sống sót hay không. Cây bao gồm các nút quyết định và các nút lá, và các nút quyết định có thể có nhiều nhánh dẫn đến các nút lá. Nút lá đại diện cho các quyết định hoặc phân loại. Nút đầu tiên đầu tiên được gọi là nút gốc.

Phương pháp cây quyết định đã trở nên phổ biến vì tính đơn giản của nó. Nó có thể xử lý tốt với các bộ dữ liệu lớn và có thể xử lý nhiễu trong bộ dữ liệu rất tốt. Một ưu điểm khác là không giống như các thuật toán khác, chẳng hạn như SVM hoặc KNN, cây quyết định hoạt động trong một hộp màu trắng, có nghĩa là chúng ta có thể thấy kết quả thu được như thế nào và quyết định nào dẫn đến nó.

3.4.2 Random Forest

Random Forest là một trong những thuật toán học máy phổ biến nhất. Nó hầu như không có đòi hỏi việc chuẩn bị dữ liệu và mô hình hóa nhưng thường kết thúc trong các kết quả không chính xác. Random Forests dựa trên các cây quyết định được mô tả trong phần trước 3.4.1. Cụ thể hơn, Random Forest là tập hợp các cây quyết định, tạo ra độ chính xác dự đoán tốt hơn. Đó là lí do tại sao nó được gọi là rừng - nó là bộ cây quyết định.

Ý tưởng thiết yếu là phát triển nhiều cây quyết định dựa trên các tập con độc lập của tập dữ liệu. Ở mỗi nút, *n variables* trong số các tính năng được chọn ngẫu nhiên, và phân chia tốt nhất trên các biến này được sử dụng.

Chúng ta có thể mô tả thuật toán như sau [8]:

1. Nhiều cây được xây dựng trên khoảng hai phần ba số liệu huấn luyện một cách ngẫu nhiên.

2. Một số biến được chọn ngẫu nhiên trong số tất cả các biến dự báo. Sau đó, sự phân chia tốt nhất trên những cái này được sử dụng để chia nút. Theo mặc định, số lượng các biến được chọn là căn bậc hai của tổng số của tất cả các dự đoán, và nó là hằng số cho tất cả các cây.

3. Với phần còn lại của dữ liệu, tỷ lệ phân loại sai được tính toán. Tổng tỷ lệ lỗi được tính là overall out-of-bag error rate.

4. Mỗi cây được đào tạo cho kết quả phân loại của nó và lớp được nhận điểm cao nhất được chọn là kết quả.

Vì chúng ta đang sử dụng nhiều cây quyết định, thuật toán này loại bỏ việc feature selection để xóa các tính năng không cần thiết - chúng sẽ không được tính đến trong mọi trường hợp. Nhu cầu duy nhất cho feature selection với các thuật toán random forest phát sinh khi có nhu cầu giảm số lượng các đặc trưng. Hơn thế nữa, out-of-bag error rate được coi là phương pháp xác thực chéo của thuật toán. Điều này loại bỏ nhu

cầu về các biện pháp xác thực chéo, mà sẽ phải được thực hiện nếu sử dụng phương pháp khác [36].

Random forest thừa hưởng nhiều ưu điểm của thuật toán cây quyết định. Chúng phù hợp cho cả vấn đề hồi quy và phân loại, chúng dễ tính toán và huấn luyện nhanh chóng để phù hợp. Nó cũng thường xuyên dẫn đến độ chính xác tốt hơn. Tuy nhiên, không giống như cây quyết định, nó không phải là rất dễ dàng để giải thích kết quả. Trong cây quyết định, bằng cách kiểm tra cây kết quả, chúng ta có thể thu được thông tin giá trị về các biến nào có liên quan và chúng ảnh hưởng như thế nào đến kết quả. Random forest cũng có thể được mô tả như một thuật toán vững chắc hơn so với cây quyết định vì nó là sự kết hợp của nhiều cây quyết định [33].

3.4.3 Gradient Boosting Decision Trees

Gradient Boosting Decision Tree (GDBT) is an ensemble model of decision trees, which are trained in sequence [20]. In each iteration, GBDT learns the decision trees by fitting the negative gradients (also known as residual errors). The main cost in GBDT lies in learning the decision trees, and the most time-consuming part of learning a decision tree is to find the best split points. One of the most common algorithms to find split points is the pre-sorted algorithm [34, 48], which lists all possible split points on the pre-sorted feature values. This algorithm is simple and can find the optimal split points. However, it is wasteful in both training speed and memory consumption. Another famous algorithm is the histogram-based algorithm [42, 23, 32]. Instead of finding the split points on the sorted feature values, histogram-based algorithm buckets continuous feature values into discrete bins and uses these bins to construct feature histograms during training.

Data: I: training data, d: max depth

Data: m: feature dimension

$nodeSet \leftarrow \{0\}$▷ tree nodes in current level

$rowSet \leftarrow \{\{0,1,2,...\}\}$▷ data indices in tree nodes

for $i = 1$ **to** d **do**

  for *node* ***in*** *nodeSet* **do**

    $usedRows \leftarrow rowSet[node]$

    for $k = 1$ **to** m **do**

      $H \leftarrow$ new Histogram()

      ▷ Build histogram

      for *j* ***in*** *usedRows* **do**

        bin ← I.f[k][j].bin

        H[bin].y ← H[bin].y + I.y[j]

        H[bin].n ← H[bin].n + 1

      end

    end

  end

  Update *rowSet* and *nodeSet* according to the best split points

end

Algorithm 1: Histogram-based Algorithm

As shown in Algorithm 1, the histogram-based algorithm finds the best split points based on the feature histograms. It costs $O(\#data \times \#feature)$ for building histogram and $O(\#bin \times \#feature)$ for finding the split point. Since $\#bin$ is usually much smaller than $\#data$, histogram building will control the computational complexity. If we can reduce $\#data$ or $\#feature$, we will be able to speed up the training of GBDT extensively.

3.4.4 Support Vector Machine

Support Vector Machines (SVM) is another machine learning algorithm that is commonly used for classification problems. The main idea relies on finding such a hyperplane, that would separate the classes in the best way. The term "support vectors"refers to the points lying closest to the hyperplane, that would change the hyperplane position if removed. The distance between the support vector and the hyperplane is referred to as margin.

Intuitively, we know that the further from the hyperplane our groups lie, the more accurate predictions we can get. That is why, although multiple hyperplanes can be found, the goal of the SVM algorithm is to find such a hyperplane that would result in the maximum margins.

SVMs are usually able to produce good accuracy, particularly on clean datasets. Further, it is good at working with the high-dimensional datasets, also when the number of dimensions is higher than the number of the samples. Additionally, for large datasets with a lot of noise or overlapping classes, it can be more effective. However, with more massive datasets training time can be extended [24].

3.4.5 K-Nearest Neighbors

The k-Nearest Neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. The k-NN does not make any assumptions about the data structure, which makes it become a good solution in the real world where most of the data does not follow the typical theoretical assumptions. K-NN is also a lazy algorithm, which means there is no specific training phase or it is very insignificant. Also, lack of generalization means that k-NN keeps all the training data, i.e., the most the training data is required during the testing phase.

The algorithm is based on feature similarity. How closely out-of-sample features match the training set determines how k-NN classify a given data point. The Eu-

clidean Distance, which is defined by the formula below, is the most used method for continuous variables in k-NN.

$$EuclideanDistance = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

The drawback of the k-NN algorithm is the lousy performance on the unevenly distributed datasets. Hence, if one class hugely overshadows the other ones, it is more likely to have more neighbors of that class due to their large number, and therefore, make incorrect predictions [30].

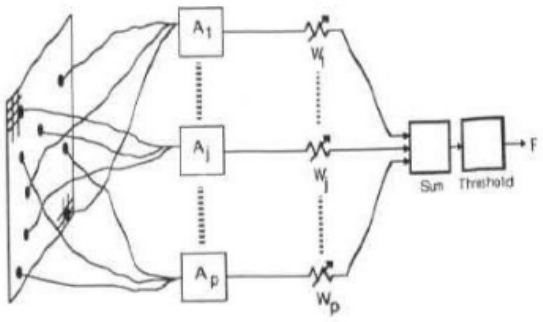
3.4.6 Neural Networks

Overview

The idea behind neural networks, i.e., having computational units that produce "intelligent" results only through interactions with each other, was inspired by the brain. For instance, the Neocognitron system, proposed by Kunihiko Fukushima in 1980, took inspiration from the mammalian visual system and laid the foundation for modern convolutional networks [21]. Hence, the artificial neurons in these networks mimic the structure of biological ones.

The very first artificial model of the biological neuron is in fact perceptron, introduced by Frank Rosenblatt in 1958 [44]. Perceptron is an algorithm for supervised learning in machine learning. Perceptron, in essence, is a simple function that turns inputs (usually a real-valued vector) into one binary output.

Following Figure 3.4, we can see the perceptron receives p inputs, $A_1, A_2, ..., A_p$ (or $x_1, x_2, ..., x_p$, depending on the source. The inputs are then respectively weighted by $w_1, w_2, ..., w_p$, which are real numbers indicating the importance of each of the input values. The output F will then be calculated using the sum of those weighted inputs. Additionally, because the output is a binary value, a threshold is used to achieve the desired result. To be more specific, a perceptron is written as follows:



Hình 3.4 Basic structure of a perceptron [35]

$$F = \begin{cases} 1 & if \sum_i^p A_i w_i \geq Threshold \\ 0 & if \sum_i^p A_i w_i < Threshold \end{cases}$$

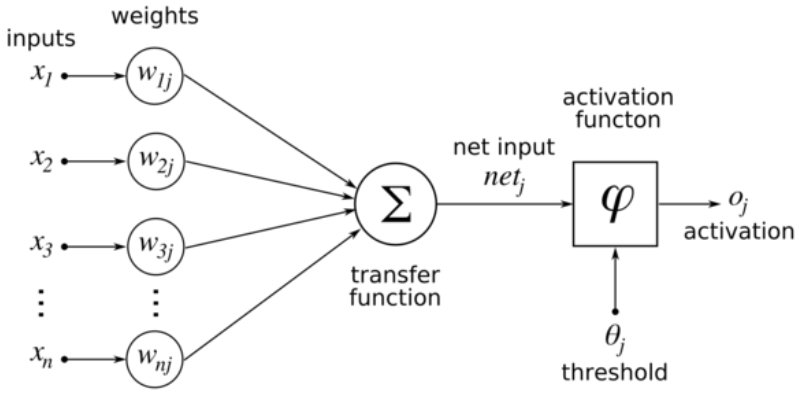
Consequently, the output of a perceptron is controlled by two things: the weights w_1, w_2, ..., w_p and the Threshold. In modern neuron networks, however, the equation changed a bit by bringing the Threshold to the other side of the inequalities. The additive inverse of Threshold is then known as Bias, and the perception will be rewritten as:

$$F = \begin{cases} 1 & if \sum_i^p A_i w_i + Bias \geq 0 \\ 0 & if \sum_i^p A_i w_i + Bias < 0 \end{cases}$$

Activation function

Perceptron is the precursor to modern artificial neurons. Instead of returning only a binary output, an artificial neuron now produces values that are anywhere in the range [0,1]. The reason is to help in the learning process of a network. Neural networks can learn, i.e., finding the appropriate weights and biases given sufficient input data. The learning process, however, needs to be progressive, which means weights and biases get increasingly close to the "good"values. Having a binary output for each of the neurons makes it hard for this process to be done. For training a neural network, we use an error function to see how far or close the network is to the optimal results. Because of

the binary output of perception, a small change in the network's parameters can lead to a stark difference for the output, making it hard to tune the parameters to achieve a good result. Therefore, modifications must be made to the original perceptron model. Instead of using 0 as the threshold at which signals are allowed to be fired, we can use an activation function to map the output to the range we need appropriately.



Hình 3.5 Structure of an artificial neuron [57]

The activation function takes as input the weighted sum of the input values fed to the perceptron and returns a value in the range [0,1]. Historically, the Sigmoid function was used for this purpose as it can squash the sum to the desired range [31]. Modern networks, however, use different functions as well, such as hyperbolic tan function (tanh) or rectified linear unit (ReLU), to achieve better performance [31].

Feedforward and Backpropagation

Two operations are particularly important in neural networks: Feedforward and Backpropagation.

Feedforward is used in both the training and testing stage of a network. The task we need to do in feedforward is straightforward: Passing output of a layer as the input of the next layer. Since all we are doing is unidirectionally putting values through the network from the input layer to the output layer, the operation is called feedforward.

Backpropagation is generally used in the training stage to help neural networks learn their parameters, and it is considered an optimization. Different from feedfor-

ward, the job of this operation is to propagate error of the output values back to the network to update the network parameters. Backpropagation works by first do the normal feedforward operation on the network with the given input. After the output is obtained, we compare the output to the desired output, using a loss function to generate an error term for each of the neuron in the output layer. The error values are then propagated backward from the output layer until every neuron receive their respective error term. The error terms will be used to calculate the gradient, with which we can update the weights of the network to minimize the loss function as the process repeats. To find the most fitting parameters, gradient descend algorithm is usually applied.

3.5 Classification Metrics

In this section, we review how to use several common metrics that are used to evaluate predictions for classification problems.

3.5.1 Logarithmic Loss

Logarithmic loss, or log-loss for short, is a performance metric for evaluating the predictions of probabilities of membership to a given class.

Log-loss takes into account the uncertainty of your prediction based on how much it varies from the actual label, which gives us a more nuanced view of the performance of our model. In binary classification, with y is a binary indicator (0 or 1) of whether class label $c$is the correct classification and p is the model predicted probability, the formula equals:

$$LogLoss = -(y\log(p) + (1-y)\log(1-p))$$

The scalar probability between 0 and 1 can be seen as a measure of confidence for a prediction by an algorithm. Smaller log-loss is better with 0 representing a perfect log-loss.

3.5.2 Confusion Matrix

		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	True positive	False positive
	Predicted condition negative	False negative	True negative

Hình 3.6 Confusion matrix [58]

A clean and unambiguous way to show the prediction results of a classifier is to use a confusion matrix (also called a contingency table). For a binary classification problem, the table has two rows and two columns (which is shown in Figure 3.6). Across the top is the actual class labels and down the side are the predicted class labels. Each cell carries the number of predictions made by the classifier that fall into that cell.

		Actual class	
		Malware	Non-Malware
Predicted class	Malware	97 True Positives	5 False Positives
	Non-Malware	3 False Negatives	15 True Negatives

Hình 3.7 An example of confusion matrix

Figure 3.7 is an example of binary classification in malware detection. Some of the input files are malware, and our test correctly says they are positive. They are called true positives (TP). In contrast, some are malware, but the test incorrectly claims they are not. They are called false negatives (FN). Some are clean files, and the test says they are not malware – true negatives (TN). Finally, there might be clean files have a positive test result – false positives (FP).

There are many derived ratios from confusion matrix, and the most common ones are listed below:

- True Positive Rate (TPR), eqv. with hit rate, recall: $TPR = TP/P = TP/(TP + FN)$
- True Negative Rate (TNR): $SPC = TN/N = TN/(TP + FN)$
- Precision or Positive Predictive Value (PPV): $PPV = TP/(TP + FP)$
- Negative Predictive Value (NPV): $NPV = TN/(TN + FN)$
- Fall-out or False Positive Rate (FPR): $FPR = FP/N = FP/(TP + FN) = 1 - TNR$
- False Discovery Rate (FDR): $FDR = FN/(FN + TP) = 1 - PPV$
- Miss Rate or False Negative Rate (FNR): $FNR = FN/(FN + TP) = 1 - TPR$

3.5.3 Overall Accuracy

Overall accuracy is the number of correct predictions made as a ratio of all predictions made.

$$Accuracy = \frac{True\ positive + True\ negative}{Condition\ positive + Condition\ negative}$$

Overall Accuracy essentially tells us out of all of the reference sites what proportion were mapped correctly. The overall accuracy is usually expressed as a percent, with 100% accuracy being a perfect classification where all reference site were classified correctly. Overall accuracy is the easiest to calculate and understand but ultimately only provides the map user and producer with necessary accuracy information.

This is the most common evaluation metric for classification problems, it is also the most misused. It is only suitable when there is an equal number of observations in each class (which is rarely the case) and that all predictions and prediction errors are equally important (which is often not the case).

3.5.4 Precision and Recall

Precision can be thought of as a measure of a classifiers exactness. Precision attempts to answer the question "What proportion of positive identifications was actually correct?". A low precision can also indicate a large number of False Positives.

$$Precision = \frac{True\ positive}{True\ positive + False\ positive}$$

Recall is the number of True Positives divided by the number of True Positives and the number of False Negatives. Computing in another way is the number of positive predictions divided by the number of positive class values in the test data. Recall attempts to answer "What proportion of actual positives was identified correctly?". It is also called Sensitivity or the True Positive Rate.

$$Recall = \frac{\sum True\ positive}{\sum Condition\ positive}$$

3.5.5 Area Under ROC curve

Area Under the Receiver Operating Characteristic curve, AUROC or AUC for short, is a performance metric for binary classification problems. The AUROC has several equivalent interpretations:

- The expectation that a uniformly drawn random positive is ranked before a uniformly drawn random negative.
- The expected proportion of positives ranked before a uniformly drawn random negative.
- The expected true positive rate if the ranking is split just before a uniformly drawn random negative.
- The expected proportion of negatives ranked after a uniformly drawn random positive.

- The expected false positive rate if the ranking is split just after a uniformly drawn random positive.

An area of 1.0 represents a model that made all predictions perfectly. A rough guide for classifying the accuracy of a classification test is the typical academic point system:

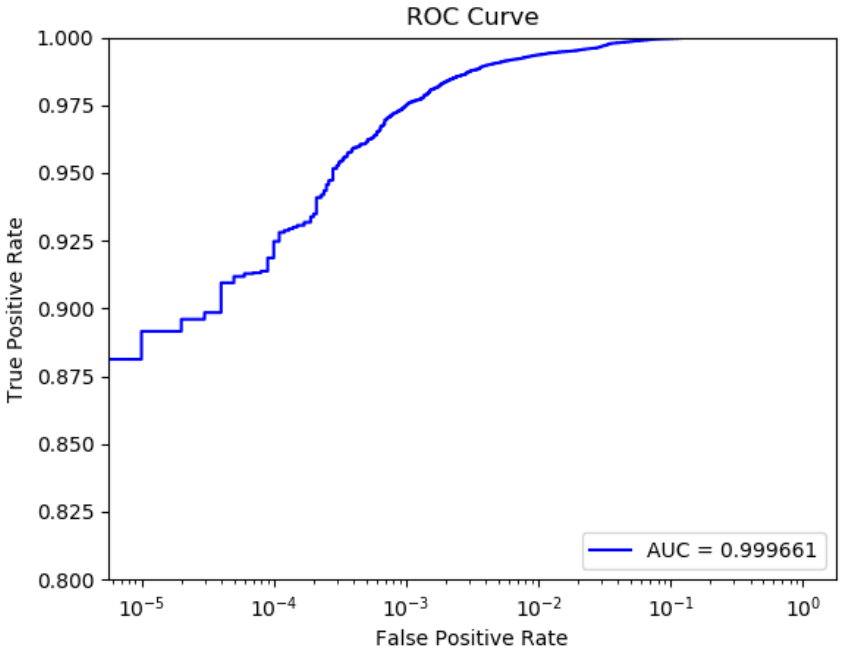
- 0.9 - 1.0 = Excellent
- 0.8 - 0.9 = Good
- 0.7 - 0.8 = Fair
- 0.6 - 0.7 = Poor
- 0.5 - 0.6 = Fail

Compute the AUROC

Assume we have a binary classifier such as logistic regression. First, we compute two metrics from the confusion matrix (their formula are mentioned in section 3.5.2), which will be later combined into one:

1. **True positive rate (TPR).** Intuitively this metric agrees to the proportion of positive data points that are correctly considered as positive, with respect to all positive data points. In other words, the higher TPR, the fewer positive data points we will miss.

2. **False positive rate (FPR).** This metric corresponds to the proportion of negative data points that are mistakenly considered as positive, concerning all negative data points. In other words, the higher FPR, the more negative data points will be misclassified.



Hình 3.8 An example of Receiver Operating Characteristic curve

Then, we combine the FPR and the TPR into one single metric by computing the two former metrics with many different threshold (for example, 0.00, $10^{-}5$, $10^{-}4$, $10^{-}3$, ..., 1.00, as shown as in Figure 3.8) for the logistic regression, then plot them on a single graph, with the FPR values on the x-axis and the TPR values on the y-axis. The resulting curve is called Receiver Operating Characteristic curve, and the metric we consider is the area under this curve.

3.6 LightGBM - A Gradient Boosting Framework

LightGBM, which means Light Gradient Boosting Machine, is a gradient boosting framework that uses tree-based learning algorithm [25]. This framework is obtaining an extreme reputation due to its following advantages:

- Faster training speed and higher efficiency
- Lower-memory usage

- Better accuracy
- Parallel and GPU learning supported
- Capable of handling the large-scale data

The framework uses two following techniques to solve problems when the feature dimension is high, and data size is considerable: Gradient-based One-Side Sampling and Exclusive Feature Bundling.

3.6.1 Gradient-based One-Side Sampling

Base on the notice that, while there is no weight for data instance in the gradient-boosting decision tree, data instances with different gradients play different roles in the computation of information gain. In particular, according to the definition of information gain, instances with larger gradients (i.e., under-trained instances) will contribute more to the information gain. Since, when subsampling the data instances, to retain the accuracy of information gain estimation, LightGBM tends to keep instances with large gradients (e.g., larger than a pre-defined threshold, or among the top percentiles), and only randomly drops instances with small gradients. They proved that such a treatment could lead to a more accurate gain estimation than uniformly random sampling, with the same target sampling rate, especially when the value of information gain has a vast range.

3.6.2 Exclusive Feature Bundling

Regularly in real applications, although there are a large number of features, the feature space is quite sparse, which provides the LightGBM a possibility of using a nearly lossless approach to reduce the number of active features. In fact, in a sparse feature space, many features are almost exclusive, i.e., they rarely take nonzero values together, e.g., the one-hot encoding features, therefore, the framework could safely

bundle such unique features. LightGBM uses an efficient algorithm named Exclusive Feature Bundling, which is a greedy algorithm with a constant approximation ratio. Specifically, they reduce the optimal bundling problem to a graph coloring problem by taking features as vertices and add edges for every two features if they are not together exclusive.

Chương 4

Phương Pháp Đề Xuất

Tóm tắt chương

Chương này mô tả các vấn đề về việc sử dụng các tập dữ liệu mất cân bằng, các phương pháp trích xuất đặc trưng, mô hình Gradient Boosting Decision Trees được đề xuất và lý do đằng sau.

4.1 Các vấn đề về sử dụng tập dữ liệu không cân bằng

Hầu hết các nghiên cứu liên quan sử dụng bộ dữ liệu mất cân bằng [45, 54]. Ví dụ, Saxe và Berlin đã sử dụng tập dữ liệu của 431.926 tệp nhị phân bao gồm 350.016 tệp độc hại [45], Vũ Thanh Nguyên và các cộng sự đã sử dụng tập dữ liệu của 9690 tệp nhưng chỉ có 300 tệp lành tính [54]. Trên thực tế, số lượng tệp độc hại thường lớn hơn nhiều so với số lượng tệp lành tính vì hầu hết các tệp nhị phân lành tính thường được bảo vệ bởi luật bản quyền không cho phép chia sẻ. Điều này làm cho vấn đề nhận diện phần mềm độc hại trở nên khác với các vấn đề phân loại học máy khác vì thường có ít mẫu hơn trong các lớp quan trọng. Hơn nữa, kích thước của tập dữ liệu thường không đủ lớn vì việc phân tích phần mềm độc hại và ghi nhãn dữ liệu là các quy trình tốn

thời gian và yêu cầu các kỹ sư bảo mật được đào tạo tốt. Bên cạnh đó, cũng có nhiều rủi ro trong việc xuất bản tập dữ liệu lớn bao gồm các tệp nhị phân độc hại.

Sử dụng bộ dữ liệu không cân bằng có thể làm cho các số liệu xác thực gây hiểu lầm. Ví dụ, với 96.9% dữ liệu là các tệp độc hại, một mô hình đánh dấu tất cả các mẫu là phần mềm độc hại đạt được độ chính xác 96.9% accuracy, 96.9% precision (P), 100% recall (R) và 0.9843 F-score ($F = 2PR/(P+R)$ [12]). Nó cũng nhường chỗ cho những dự báo sai, gây ra trải nghiệm người dùng tiêu cực. Theo một cuộc khảo sát của các quản trị viên CNTT trong năm 2017, 42% các công ty cho rằng người dùng của họ bị mất năng suất là một vấn đề của kết quả dương tính giả, tạo ra một điểm nghẹt thở cho các quản trị viên CNTT trong môi trường doanh nghiệp [15].

4.2 Feature Extraction

Bằng cách sử dụng các phương pháp trích xuất đặc trưng đơn giản chứ không phải tệp nhị phân thô, việc thu thập dữ liệu không bị ảnh hưởng bởi chính sách bảo mật và dễ dàng hơn để có được tập dữ liệu cân bằng. Bằng cách thực hiện nhiều thử nghiệm, chúng tôi giảm kích thước đặc trưng xuống 30% (1711 thay vì 2351) để giảm thời gian đào tạo nhưng vẫn đạt được kết quả đánh giá tốt hơn. Cụ thể, chúng tôi trích xuất từng tệp PE thành tám nhóm đặc trưng có thể được phân loại thành hai loại: đặc trưng định dạng bất khả tri (format-agnostic features) và đặc trưng PE được phân tích cú pháp (parsed PE features). Các nhóm Format-agnostic Feature làm giảm mối quan tâm về quyền riêng tư trong khi các nhóm parsed PE feature đóng gói thông tin liên quan đến mã thực thi.

4.2.1 Format-agnostic Features

Chúng tôi sử dụng ba nhóm đặc trưng để mô hình hóa nội dung của tệp đầu vào độc lập theo định dạng tệp, có nghĩa là nó không phụ thuộc vào định dạng của tệp.

Byte-Entropy Histogram

Dựa trên những nghiên cứu của Joshua Saxe và Konstantin Berlin [45], họ chỉ ra rằng, trong thực tế, hiệu quả của việc tái thể hiện byte value trong ngữ cảnh entropy mà nó xảy ra sẽ phân tách các byte value ra khỏi bối cảnh chung, ví dụ, dữ liệu x86 instruction data được phân tách từ dữ liệu nén.

Để tính byte-entropy histogram, chúng tôi trượt một cửa sổ 2048-length trên toàn bộ byte đầu vào với bước nhảy 1024 byte. Sử dụng một mẹo đơn giản để tính toán entropy H nhanh hơn, cụ thể, giảm đi một nữa lượng thông tin, và bắt cặp nó với từng byte trong window. Sau đó, chúng tôi tính một histogram 2 chiều với 16×16 bins chứa entropy và byte value. Cuối cùng, chúng tôi cộng các vector hàng trong ma trận và chuẩn hóa để có một 256-value vector.

Byte Histogram

Byte histogram là một 256-value vector đại diện cho phân phối của từng giá trị byte trong tệp.

String Information

Nhóm các đặc trưng định dạng bất khả tri cuối cùng là thông tin chuỗi. Các đặc trưng này được trích xuất từ những chuỗi các kí tự in được trong khoảng `0x20` đến `0x7f`, và phải có ít nhất 5 kí tự. Chúng tôi sử dụng số lượng chuỗi, độ dài trung bình của các chuỗi, số lượng các chuỗi có khả năng là đường dẫn (bắt đầu với `C:\`), địa chỉ web (bắt đầu với `http://` hoặc `https://`), a registry key (sự xuất hiện của `HKEY_`) và một file thực thi được nén (chuỗi `MZ`). Ngoài ra, chúng tôi sử dụng histogram của các ký tự có thể in trong các chuỗi này.

4.2.2 Parsed Features

Ngoài việc sử dụng ba nhóm đặc trưng định dạng bất khả tri, chúng tôi trích xuất năm nhóm khác từ phân tích tệp PE bằng cách sử dụng LIEF - Library to Instrument Executable Formats [40].

General Information

Đây là tập hợp các đặc trưng bao gồm file size và những thông tin cần thiết khác từ PE header: virtual size, số lượng imported và exported functions, số lượng symbols, dữ liệu có hay không debug section, thread local storage, resources, relocations, hoặc signature.

Header Information

Chúng tôi sử dụng thông tinh từ Common Object File Format (COFF) header bao gồm timestamp, target machine và danh sách image characteristics. Từ optional header, chúng tôi dùng target subsystem, DLL characteristics, file magic, major và minor image versions, linker versions, system versions và subsystem versions, code size, header size và commit size. Chúng tôi sử dụng hashing trick với 10 bins có các đặc trưng dạng chuỗi [56].

Imported Functions

Phân tích import address table cho chúng ta một báo cáo về các hàm được import bởi các thư viện. Chúng tôi sử dụng tập hợp các thư viện với hashing trick 128-bin và áp dụng hashing 512-bin để ghi lại các function riêng lẻ, bằng cách biểu diễn dưới định dạng chuỗi `library:function`, ví dụ, `kernel32.dll:CreateFileMappingA`.

Exported Functions

Tương tự như cách trích xuất imported function, chúng tôi thống kê một danh sách các exported function vào một 128-value vector bằng cách hashing.

Section Information

Thuộc tính của từng section được sử dụng: name, size, entropy, virtual size, và một danh sách các chuỗi thể hiện characteristics của section.Chúng tôi vẫn sử dụng hashing trick trên các cặp `(section name, value)` để tạo ra những 50-value vector chứa section size, section entropy, virtual size, và thông tin về entry point characteristics.

4.3 Classification

Trong nghiên cứu này, we đề xuất sử dụng thuật toán Gradient Boosting Decision Trees với 400 iterations và 64 lá mỗi cây. Chúng tôi cấu hình rằng phải có ít nhất 200 mẫu ở một nút và đặt tỷ lệ học tập là 5 phần trăm. Chúng tôi giải thích về lý do đằng sau những lựa chọn dưới đây.

Thứ nhất, số lượng lớn các đặc trưng gây ra các vấn đề về khả năng mở rộng cho nhiều thuật toán học máy. Ví dụ, non-linear SVM kernels yêu cầu $O(N^2)$ phép nhân trong mỗi lần lặp, và k-Nearest Neighbors (k-NN) yêu cầu một lượng tính toán đáng kể và lưu trữ tất cả các mẫu nhãn trong khi dự đoán. Do đó, chúng tôi tập trung vào việc sử dụng neural networks và ensemble decision trees, đó là lựa chọn thay thế có thể mở rộng.

Thứ hai, nguồn lực của chúng tôi, chủ yếu là hỗ trợ tài chính, rất ít. But chi phí cho việc đào tạo neural networks cực kỳ tốn kém tính toán. Các mô hình phức tạp mất nhiều giờ và đòi hỏi nhiều GPU hơn để tăng tốc. Ngoài ra, neural networks là hộp đen và yêu cầu nhiều kinh nghiệm để tối ưu hóa.

Bên cạnh đó, các thuật toán tree ensemble xử lý các không gian đặc trưng lớn rất tốt, cũng như xử lí tốt một số lượng lớn các mẫu đào tạo. Hai thuật toán phổ biến

là Random Forests và Gradient Boosting Decision Trees (GDBT). Đào tạo GBDT thường mất nhiều thời gian hơn vì cây được xây dựng theo tuần tự. Tuy nhiên, kết quả cho thấy GBDT tốt hơn so với Random Forests.

Chương 5

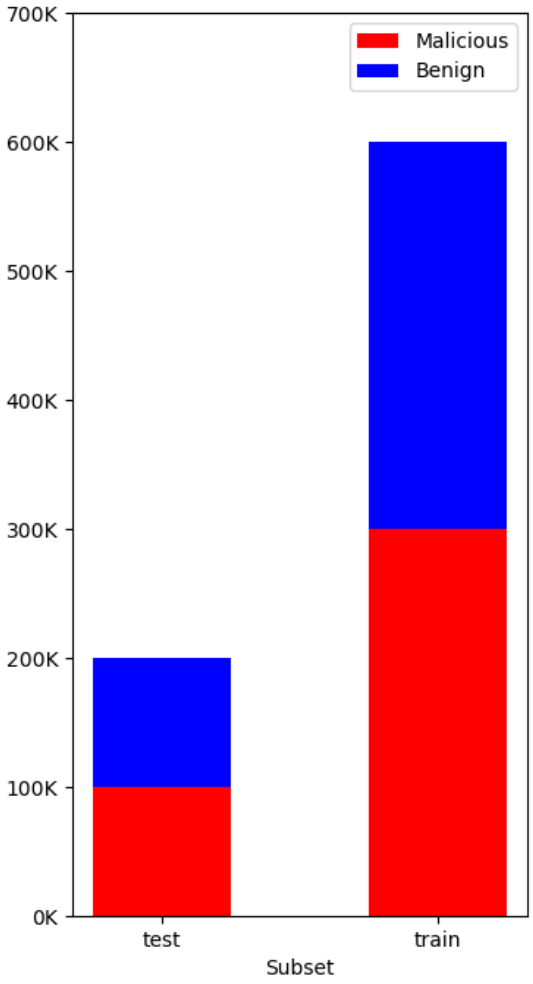
Thực Nghiệm và Đánh Giá

Tóm tắt chương

Firstly, we introduce the dataset used for training and evaluating in our experiments. We also present the evaluation metrics and explain the reason for choosing those criteria. Then, the experiment results are shown in comparing to other proposed models. Finally, we present how to set up the environments for experiments.

5.1 Tập dữ liệu

Trong thử nghiệm của chúng tôi, chúng tôi sử dụng 600.000 mẫu đào tạo được dán nhãn và 200.000 mẫu thử nghiệm từ tập dữ liệu Endgame Malware BEnchmark for Research (EMBER) [3].



Hình 5.1 Phân phối mẫu trong tập dữ liệu.

Tập dữ liệu EMBER là tập dữ liệu công khai lớn để phát hiện phần mềm độc hại, nó bao gồm các tệp lành tính và có tỷ lệ lý tưởng các tệp độc hại và lành tính cho các tác vụ học máy. Điều này sẽ giải quyết vấn đề chung về độ chính xác dự đoán, cụ thể là vấn đề gây hiểu lầm khi dữ liệu bị mất cân bằng.

5.2 Tiêu chí Đánh giá

Như đã thảo luận trong phần 2.3, mục tiêu của nghiên cứu là tìm ra phương pháp phát hiện phần mềm độc hại dựa trên học máy hoạt động ở tỷ lệ dương giả thấp trong khi cố gắng đạt được tỷ lệ phát hiện cao.

5.2.1 Tỷ lệ Báo động sai

Sai tích cự (False positive), hoặc báo động sai, xảy ra khi một mô hình sai lầm gán một nhãn độc hại cho một tập tin lành tính. Chúng tôi có tập trung làm cho tỷ lệ dương tính giả càng thấp càng tốt, đó là việc không điển hình trong ứng dụng học máy. Điều quan trọng là bởi vì ngay cả một báo động giả trong một nghìn tập tin lành tính có thể tạo ra hậu quả nghiêm trọng cho người dùng. Vấn đề này là phức tạp bởi thực tế là có rất nhiều tập tin sạch trên thế giới, chúng tiếp tục xuất hiện, và rất khó khăn để thu thập các tập tin này. Chúng tôi đánh giá phương pháp của chúng tôi với hai giá trị báo động giả, cụ thể: ở mức dưới 0,1% và ở mức dưới 1%.

$$False\ alarm\ rate = \frac{\sum False\ positive}{\sum Condition\ negative}$$

5.2.2 Tỷ lệ Phát hiện

Tỷ lệ phát hiện, (tương đương với recall hoặc true positive rate), đo tỷ lệ các chương trình độc hại được phát hiện trong các tệp phần mềm độc hại được sử dụng để thử nghiệm. Với tỉ lệ cao hơn, ít trường hợp phần mềm độc hại trong thực tế không bị phát hiện. Nói cách khác, tỷ lệ phát hiện cho thấy tiềm năng của các tệp nhị phân độc hại mới sẽ được phát hiện.

$$Detection\ rate = \frac{\sum True\ positive}{\sum Condition\ positive}$$

5.2.3 Diện tích dưới đường cong ROC

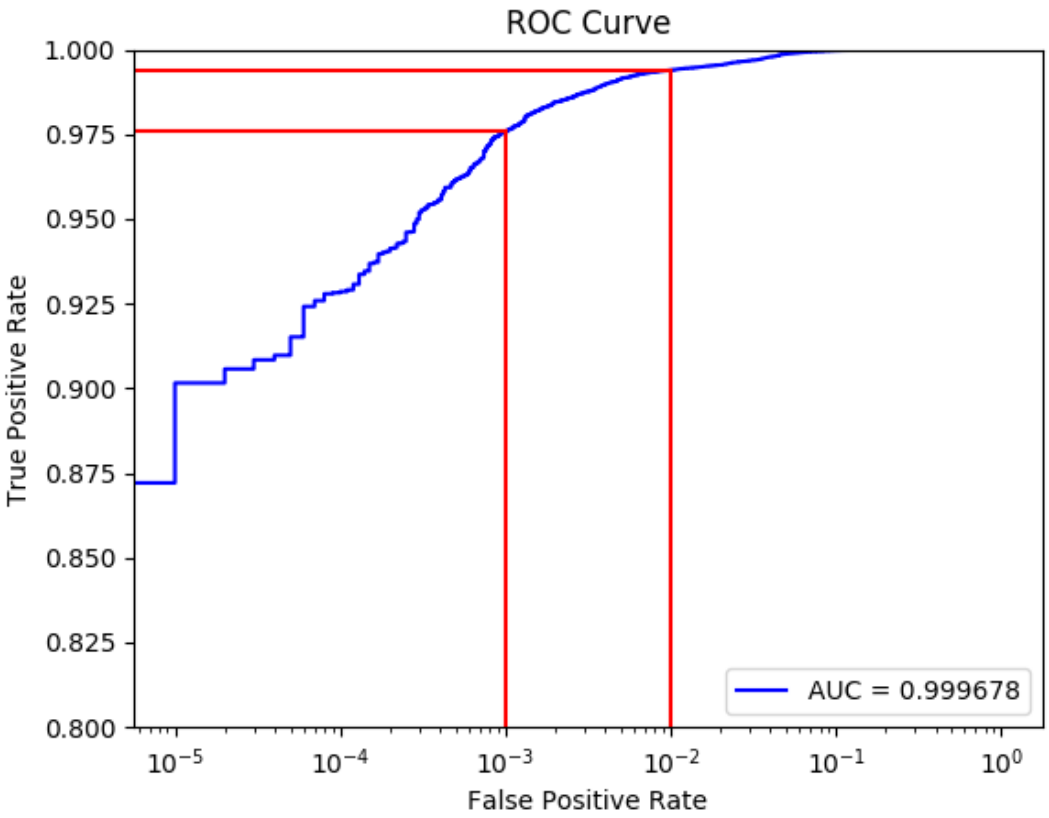
Như đã giới thiệu trong phần 3.5.5, phần Diện tích dưới đường cong ROC (Area Under the ROC curve, viết tắt là AUROC hoặc AUC) cung cấp thước đo tổng thể về hiệu suất trên tất cả các ngưỡng phân loại có thể có. AUC là quy mô bất biến và đo lường dự đoán được xếp hạng tốt như thế nào, chứ không phải là giá trị tuyệt đối của chúng. Bên cạnh đó, AUC là bất biến với ngưỡng phân loại, để nó có thể đo lường chất lượng của các dự đoán không phụ thuộc vào ngưỡng nào được chọn. Một mô hình có dự đoán là 100% sai có AUC là 0,0, và có một dự đoán là 100% đúng có AUC là 1,0.

Hệ thống điểm học thuật điển hình để phân loại độ chính xác của bài kiểm tra phân loại như sau:

- 0.9 - 1.0 = Excellent
- 0.8 - 0.9 = Good
- 0.7 - 0.8 = Fair
- 0.6 - 0.7 = Poor
- 0.5 - 0.6 = Fail

5.3 Kết quả Thực nghiệm

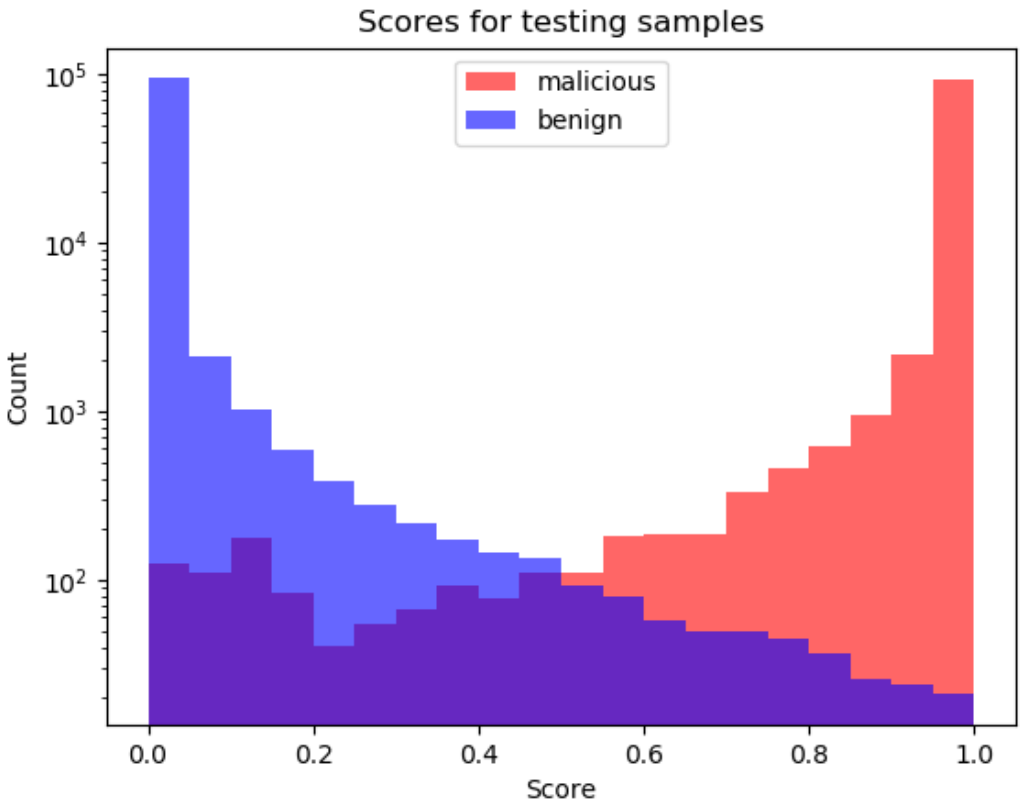
Phương pháp phát hiện phần mềm độc hại dựa trên GBDT được đề xuất được triển khai với LightGBM framework [25], và các vectơ đặc trưng đầu vào có kích thước 1711. Tất cả các thử nghiệm của chúng tôi đều chạy trên một máy tính ảo hóa có 24 vCPUs và bộ nhớ 32 GB. Sử dụng lập trình song song, mất khoảng 10 phút để vector hóa các tính năng thô và khoảng 5 phút để đào tạo mô hình. Đường cong ROC của mô hình cuối cùng được thể hiện trong hình 5.2 và phân phối điểm số cho các mẫu thử được thể hiện trong hình 5.3.



Hình 5.2 The ROC curve of proposed model

Diện tích dưới đường cong ROC đạt mức 0.999678. Với threshold 0.828987, điểm kết quả của mô hình khi ít hơn 0.1% tỉ lệ báo động sai có tỉ lệ phát hiện 97.5720%. Và với mức tỉ lệ báo động sai 1%, mô hình đạt tỉ lệ phát hiện 99.3940% với threshold là 0.307897.

Mô hình cơ sở của EMBER chỉ có diện tích dưới đường cong ROC là 0.99911, kết quả với mức 0.1% FPR là 92.99% TPR, và mức 1% FPR, là 98.2% TPR. Mô hình của chúng tôi có hiệu suất tốt hơn do điều chỉnh siêu tham số và cũng mất ít thời gian hơn cho việc đào tạo do giảm số lượng các đặc trưng.Rõ ràng, mô hình có hiệu suất tốt hơn so với mô hình MalConv được đào tạo trên các tệp nhị phân thô [3], nó có ROC AUC là 0.99821, tương ứng với 92.2% TPR ở mức báo động sai dưới 0.1%, và 97.3% TPR ở mức dưới 1% FPR. Bảng 5.1 và bảng 5.2hiển thị thời gian đào tạo và kết quả



Hình 5.3 The distribution of scores for testing samples

đánh giá của mô hình đề xuất của chúng tôi so với mô hình MalConv và mô hình cơ sở của EMBER.

Bảng 5.1 Thời gian đào tạo của mô hình được đề xuất của chúng tôi so với mô hình MalConv và mô hình cơ sở của EMBER

Model	Input	Specifications	Training time
MalConv	Raw binaries	2 NVDIA TITAN X (Pascal) GPUs	10 days (25 hours/epoch)
EMBER	2351-value vectors	8 vCPUs (2015 MacBook Pro i7)	20 hours
Our model	**1711**-value vectors	24 vCPUs (Google Compute Engine)	**5 minutes**

Bảng 5.2 Kết quả đánh giá của mô hình được đề xuất của chúng tôi so với mô hình MalConv và mô hình cơ sở của EMBER

Model	False Alarm Rate (FPR)	Detection Rate (TPR)	Area Under the ROC curve (AUC)
MalConv	0.1 %	92.200 %	0.998210
	1.0 %	97.300 %	
EMBER	0.1 %	92.990 %	0.999110
	1.0 %	98.200 %	
Our model	0.1 %	97.572 %	0.999678
	1.0 %	99.394 %	

5.4 Hướng dẫn cài đặt môi trường

We mainly use the cross-platform tools in research and development for easily switching between operating systems. We use a Windows 10 Pro virtual machine for static malware analysis, an Ubuntu 16.04 LTS cloud instance for training and testing machine learning models, and use PyCharm Professional as mainly Integrated development environment (IDE).

5.4.1 Windows environment for static analysis

We use a virtual machine to build a background about malware:

- OS: Microsoft Windows 10 Pro
- Version: 10.0.17134
- Architecture: 64-bit

With following tools, we can easily gather malware basic information:

- **CFF Explorer**: PE header parser.

- **PE Explorer** (from Heaventools Software): PE inspection tool.
- **BinText** (from McAfee): extract string from a binary.
- **HxD Hex Editor**: support for viewing file in binary format.

5.4.2 Ubuntu environment for machine learning tasks

Google Cloud Platform

We use an virtual machine for research, that can be deployed with the image from *Cloud Launcher - Canonical - Ubuntu Xenial*. The cloud instance has 24 virtual CPUs, 32 GB for memory, and is located in `asia-southeast1-b` zone, i.e., Jurong West, Singapore.

After deployment, we add two optional firewall rules (*VPC network - Firewall rules - Create a firewall rule*), which allows all in and out connections for the virtual machine, to use Python Interactive Console features in PyCharm IDE.

Anaconda

We choose Anaconda, a free and open source distribution of the Python, to manage package and deploy. The content of `environment.yml` used to deploy is shown below.

```
name: lab
channels:
  - conda-forge
dependencies:
  - python==3.6
  - matplotlib
  - numpy
  - scikit-learn
```

```
- pip:
  - lief
  - git+https://github.com/onnx/onnxmltools
  - lightgbm
```

The environment is created with **Python 3.6** and packages for machine learning:

- **NumPy**: the fundamental package for scientific computing with Python.
- **Matplotlib**: a Python 2D plotting library.
- **Scikit-learn**: a machine learning library.
- **Lief**: library to instrument executable formats.
- **LightGBM**: a gradient boosting framework based on decision tree algorithms.
- **ONNXMLTools**: a tool to convert models to ONNX format.

5.4.3 PyCharm Professional IDE

JetBrains provides *free individual licenses for students* to use PyCharm Professional IDE. This is the powerful Python IDE, which gives us remote development capabilities and supports many scientific tools (e.g., Anaconda, Matplotlib and NumPy).

Following the guide *Configuring Remote Interpreters via SSH* published by JetBrains, we can run, debug remotely from a cloud instance, which gives a great performance and is easy to scale.

Chương 6

Conclusion

Tóm tắt chương

Chapter 6 presents the results of this thesis, including what we have learned and achieved through the experiments. The chapter closes with our proposal for future work.

6.1 Results

Over the course of doing this thesis, we have spent a good amount of time studying about Malware Detection and Machine Learning, including Neural Networks and Gradient Boosting Decision Trees, to acquire essential malware knowledge.

We have learned and distinguished static and dynamic malware detetion. We have also analyzed the technical detail and meaning of features used in proposed static malware detectors. Understanding them is important both for understanding the state-of-the-art methods and for building and optimizing classifiers.

We present and optimize a static malware detection method using hand-crafted features derived from parsing the PE files and Gradient Boosting Decision Trees (GBDT), a widely-used powerful machine learning algorithm. We manage to reduce

the training time by appropriately reducing the feature dimension. In detail, rather than using raw binary files, our proposed method uses the statistical summaries to decrease the privacy concerns of various benign files and makes it easy to request the balanced dataset.

The experiment results show that our proposed method can achieve up to 99.394% detection rate at 1% false alarm rate, and score results in less than 0.1% false alarm rate at a detection rate 97.572%, based on more than 600,000 training and 200,000 testing samples from EMBER dataset [3].

6.2 Future Works

The study conducted in this project was a proof-of-concept, and we can identify some future developments related to the practical implementation:

1. **Reduce the feature space.** There is possible to reduce the dimension of feature vectors. Input vectors with smaller size boost the model and take less training time.

2. **Use other datasets.** Although the EMBER dataset is broad, covering most of the malware species, it does not include all possible kinds. Collecting a dataset is a task that requires a lot of time and efforts, especially in malware detection domain. With using format-agnostic features, we can receive more samples from security organizations in future.

3. **Implement the approach in local computer.** We tried to implement the model with ONNX format and Windows ML format but it was not success because the preview version of Windows ML is changed rapidly and has many limits. We plan to build a demonstration application to propose that machine learning-based malware detectors can run smoothly in personal computer.

Tài liệu tham khảo

[1] Abou-Assaleh, T., Cercone, N., Keselj, V., and Sweidan, R. (2004). N-gram-based detection of new malicious code. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, volume 2, pages 41–42. IEEE.

[2] Almási, A.-D., Woźniak, S., Cristea, V., Leblebici, Y., and Engbersen, T. (2016). Review of advances in neural networks: Neural design technology stack. *Neurocomputing*, 174:31–41.

[3] Anderson, H. S. and Roth, P. (2018). Ember: An open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*.

[4] Athiwaratkun, B. and Stokes, J. W. (2017). Malware classification with lstm and gru language models and a character-level cnn. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2482–2486.

[5] Banko, M. and Brill, E. (2001). Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th annual meeting on association for computational linguistics*, pages 26–33. Association for Computational Linguistics.

[6] Bayer, U., Comparetti, P. M., Hlauschek, C., Kruegel, C., and Kirda, E. (2009). Scalable, behavior-based malware clustering. In *NDSS*, volume 9, pages 8–11. Citeseer.

[7] Benchea, R. and Gavriluţ, D. T. (2014). Combining restricted boltzmann machine and one side perceptron for malware detection. In *International Conference on Conceptual Structures*, pages 93–103. Springer.

[8] Biau, G. (2012). Analysis of a random forests model. *Journal of Machine Learning Research*, 13(Apr):1063–1095.

[9] Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

[10] Chawla, N. V. (2009). Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*, pages 875–886. Springer.

[11] Chen, Q. and Bridges, R. A. (2017). Automated behavioral analysis of malware A case study of wannacry ransomware. *CoRR*.

[12] Chinchor, N. (1992). Muc-4 evaluation metrics, in proceedings of the fourth message understanding conference (muc-4). *Morgan Kaufman Publishers*, page 22.

[13] Cohen, F. (1987). Computer viruses: theory and experiments. *Computers & security*, 6(1):22–35.

[14] Cohen, W. W. (1995). Fast effective rule induction. In *Machine Learning Proceedings 1995*, pages 115–123. Elsevier.

[15] Crowe, J. (2017). Security false positives cost companies $1.37 million a year on average.

[16] Dahl, G. E., Stokes, J. W., Deng, L., and Yu, D. (2013). Large-scale malware classification using random projections and neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3422–3426.

[17] DuPaul, N. (2012). Common malware types: Cybersecurity 101.

[18] Egele, M., Scholte, T., Kirda, E., and Kruegel, C. (2012). A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR)*, 44(2):6.

[19] Fradkin, D. and Madigan, D. (2003). Experiments with random projections for machine learning. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 517–522. ACM.

[20] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

[21] Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

[22] Jang, J., Brumley, D., and Venkataraman, S. (2011). Bitshred: feature hashing malware for scalable triage and semantic analysis. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 309–320. ACM.

[23] Jin, R. and Agrawal, G. (2003). Communication and memory efficient parallel decision tree construction. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 119–129. SIAM.

[24] Jing, R. and Zhang, Y. (2010). A view of support vector machines algorithm on classification problems. In *Multimedia Communications (Mediacom), 2010 International Conference on*, pages 13–16. IEEE.

[25] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3149–3157.

[26] Kearn, M. (2016). Machine learning is for muggles too!

[27] Kephart, J. O., Sorkin, G. B., Arnold, W. C., Chess, D. M., Tesauro, G. J., White, S. R., and Watson, T. (1995). Biologically inspired defenses against computer viruses. In *IJCAI (1)*, pages 985–996.

[28] Kolter, J. Z. and Maloof, M. A. (2006). Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 7(Dec):2721–2744.

[29] Kovacs, E. (2015). False positive alerts cost organizations $1.3 million per year: Report.

[30] Laaksonen, J. and Oja, E. (1996). Classification with learning k-nearest neighbors. In *Neural Networks, 1996., IEEE International Conference on*, volume 3, pages 1480–1483. IEEE.

[31] Li, F.-F., Karpathy, A., and Johnson, J. (2015). Cs231n: Convolutional neural networks for visual recognition. *University Lecture*.

[32] Li, P., Wu, Q., and Burges, C. J. (2008). Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in neural information processing systems*, pages 897–904.

[33] Louppe, G. (2014). Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502*.

[34] Mehta, M., Agrawal, R., and Rissanen, J. (1996). Sliq: A fast scalable classifier for data mining. In *International Conference on Extending Database Technology*, pages 18–32. Springer.

[35] Minsky, M. and Papert, S. (1969). Perceptron (expanded edition).

[36] Mitchell, T. M. et al. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877.

[37] Moir, R. (2003). Defining malware: Faq. *Microsoft Windows Server*.

[38] Moser, A., Kruegel, C., and Kirda, E. (2007). Limits of static analysis for malware detection. In *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*, pages 421–430. IEEE.

[39] Nguyen, V. T., Nguyen, T. T., Mai, K. T., and Le, T. D. (2014). A combination of negative selection algorithm and artificial immune network for virus detection. In *Future data and security engineering*, pages 97–106. Springer.

[40] Quarkslab, R. T. (2017). Library to instrument executable formats.

[41] Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., and Nicholas, C. (2017). Malware detection by eating a whole exe. *arXiv preprint arXiv:1710.09435*.

[42] Ranka, S. and Singh, V. (1998). Clouds: A decision tree classifier for large datasets. In *Proceedings of the 4th Knowledge Discovery and Data Mining Conference*.

[43] Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., and Ahmadi, M. (2018). Microsoft malware classification challenge. *arXiv preprint arXiv:1802.10135*.

[44] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[45] Saxe, J. and Berlin, K. (2015). Deep neural network based malware detection using two dimensional binary program features. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 11–20.

[46] Schneier, B. (2005). Sony's drm rootkit: The real story.

[47] Schultz, M. G., Eskin, E., Zadok, E., and Stolfo, S. J. (2001). Data mining methods for detection of new malicious executables. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 38–.

[48] Shafer, J., Agrawal, R., and Mehta, M. (1996). Sprint: A scalable parallel classi er for data mining. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 544–555. Citeseer.

[49] Shafiq, M. Z., Tabish, S. M., Mirza, F., and Farooq, M. (2009a). A framework for efficient mining of structural information to detect zero-day malicious portable executables. Technical report, Technical Report, TR-nexGINRC-2009-21.

[50] Shafiq, M. Z., Tabish, S. M., Mirza, F., and Farooq, M. (2009b). Pe-miner: Mining structural information to detect malicious executables in realtime. In *International Workshop on Recent Advances in Intrusion Detection*, pages 121–141.

[51] Shahi, G., Pang, E., and Fong, P. (2009). *Technology in a Changing World*. lulu.com.

[52] Van Nhuong, N., Nhi, V. T. Y., Cam, N. T., Phu, M. X., and Tan, C. D. (2014). Semantic set analysis for malware detection. In *IFIP International Conference on Computer Information Systems and Industrial Management*, pages 688–700. Springer.

[53] Vidas, T. and Christin, N. (2014). Evading android runtime analysis via sandbox detection. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 447–458. ACM.

[54] Vu, T. N., Nguyen, T. T., Trung, H. P., Do Duy, T., Van, K. H., and Le, T. D. (2017). Metamorphic malware detection by pe analysis with the longest common sequence. In *International Conference on Future Data and Security Engineering*, pages 262–272. Springer.

[55] Weber, M., Schmid, M., Schatz, M., and Geyer, D. (2002). A toolkit for detecting and analyzing malicious software. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 423–431. IEEE.

[56] Weinberger, K., Dasgupta, A., Langford, J., Smola, A., and Attenberg, J. (2009). Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120. ACM.

[57] Wikipedia (2018a). Artificial neural networks.

[58] Wikipedia (2018b). Confusion matrix.

[59] Wikipedia (2018c). Decision tree learning.

[60] Wikipedia (2018d). Portable executable.