Universidade Estadual de Campinas
Instituto de Computação

Luiz Claudio Navarro

A supervised method for finding discriminant variables in complex problem analysis: case studies on Android security and source printer attribution.

Um método supervisionado para encontrar variáveis discriminantes na análise de problemas complexos: estudos de caso em segurança do Android e em atribuição de impressora fonte.

CAMPINAS
2018

Luiz Claudio Navarro

A supervised method for finding discriminant variables in complex problem analysis: case studies on Android security and source printer attribution.

Um método supervisionado para encontrar variáveis discriminantes na análise de problemas complexos: estudos de caso em segurança do Android e em atribuição de impressora fonte.

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Thesis presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/Orientador: Prof. Dr. Ricardo Dahab
Co-supervisor/Coorientador: Prof. Dr. Anderson de Rezende Rocha

Este exemplar corresponde à versão final da Dissertação defendida por Luiz Claudio Navarro e orientada pelo Prof. Dr. Ricardo Dahab.

CAMPINAS
2018

Agência(s) de fomento e nº(s) de processo(s): Não se aplica.
ORCID: https://orcid.org/0000-0002-8041-8743

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

N228s

Navarro, Luiz Claudio, 1956-
A supervised method for finding discriminant variables in complex problem analysis : case studies on Android security and source printer attribution / Luiz Claudio Navarro. – Campinas, SP : [s.n.], 2018.

Orientador: Ricardo Dahab.
Coorientador: Anderson de Rezende Rocha.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Dispositivos móveis - Medidas de segurança. 2. Tecnologia da informação - Sistemas de segurança. 3. Android (Recurso eletrônico). 4. Aprendizado de máquina. 5. Análise forense de imagens digitais. 6. Impressoras (Computadores). I. Dahab, Ricardo, 1957-. II. Rocha, Anderson de Rezende, 1980-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Um método supervisionado para encontrar variáveis discriminantes na análise de problemas complexos : estudos de caso em segurança do Android e em atribuição de impressora fonte
Palavras-chave em inglês:
Mobile devices - Security measures
Information technology - Security systems
Android (Electronic resource)
Machine learning
Digital image forensics
Computer printers
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Ricardo, Dahab
Marjory Cristiany Da Costa Abreu
Hélio Pedrini
Data de defesa: 20-04-2018
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação

Luiz Claudio Navarro

A supervised method for finding discriminant variables in complex problem analysis: case studies on Android security and source printer attribution.

Um método supervisionado para encontrar variáveis discriminantes na análise de problemas complexos: estudos de caso em segurança do Android e em atribuição de impressora fonte.

Banca Examinadora:

- Prof. Dr. Ricardo Dahab
  IC-UNICAMP - Instituto de Computação - Universidade Estadual de Campinas
- Profa. Dra. Marjory Cristiany Da Costa Abreu
  DIMap-UFRN - Departamento de Informática e Matemática Aplicada - Universidade Federal do Rio Grande do Norte - UFRN
- Prof. Dr. Hélio Pedrini
  IC-UNICAMP - Instituto de Computação - Universidade Estadual de Campinas

A ata da defesa com as respectivas assinaturas dos membros da banca encontra-se no processo de vida acadêmica do aluno.

Campinas, 20 de abril de 2018

Dedicatória

Dedico esta dissertação a minha família, que é o maior bem que sempre pude ter.

À minha esposa Wu Chiang Kuo "Graciela" Navarro que me acompanha em todos os momentos dessa jornada com sua força e determinação, cujo incentivo foi fundamental para o desenvolvimento e conclusão desse trabalho, nunca me deixando abater com as dificuldades.

À minha mãe Ruth Avelino Navarro que me ensinou a ler muito antes de frequentar a escola recortando letras de jornais espalhados no chão e que me educou no gosto de aprender com os livros. Ao meu pai Antonio Navarro, um exemplo de luta e autodidatismo, que me ensinou a observar tudo em detalhes, a aprender com tudo que se vê e se faz, a medir com precisão e fazer com capricho. Sempre me desafiando a dar um passo a frente, com aquela frase que se tornou a sua marca em nossa família "Será que você é capaz de ...", a qual sempre exerceu um poder mágico instigando-nos a buscar soluções.

Aos meus filhos Alexandre Khae Wu Navarro e William Tyi Wu Navarro que sempre me deram alegria e que me inspiram com seus exemplos de vida, e pelo carinho que nos dedicam.

Uma família que com o esforço de todos e uma batalha constante de três gerações pode superar as dificuldades de uma origem humilde e difícil, e alcançar a educação almejada por muitos.

Never regard study as a duty, but as the enviable opportunity to learn to know the liberating influence of beauty in the realm of the spirit for your own personal joy and to the profit of the community to which your later work belongs. (Albert Einstein)

Agradecimentos

Agradeço ao meu orientador Prof. Dr. Ricardo Dahab que me acolheu no programa de Mestrado, já com 57 anos de idade, acreditando na minha capacidade de voltar a frequentar os bancos da escola e aprender depois de uma longa carreira fora da academia, e deu-me liberdade de explorar e navegar pelos temas desse trabalho, mantendo o objetivo e o rumo. Sua orientação me proporcionou o caminho para a readaptação e o progresso na área acadêmica.

Ao Prof. Anderson Rocha, com quem aprendi tudo o que sei sobre aprendizagem de máquina, que mesmo com todo a sua carga de trabalho, sempre me esteve disponível para dúvidas, discussões, "insigths" e o trabalho de co-orientar e revisar em detalhes os artigos de aprendizagem de máquina.

Ao Prof. Dr. Roberto Gallo pelos conhecimentos transmitidos e com quem pude realizar os primeiros trabalhos na área de segurança do sistema Android.

Ao Prof. Dr. Anselmo Ferreira pelo aprendizado e discussão dos métodos de atribuição de impressoras, e pela oportunidade de trabalharmos juntos de forma tão colaborativa.

Aos professores Prof. Dr. Paulo Lício de Geus e Prof. Dr. André Gregio pelas aulas de segurança de sistemas e pela colaboração nos artigos sobre malware Android sem os quais esse trabalho não seria possível.

A todos os colegas do LASCA (Laboratório de Segurança e Criptografia Aplicada) e do laboratório RECOD ("Reasoning of Complex Data") ambos do Instituto de Computação (IC) da UNICAMP pela união e colaboração de todos em um ambiente de suporte mútuo.

Ao Prof. Alberto Romano Schiesari, que nos idos de 1976, antes mesmo que eu cursasse uma faculdade, me ensinou lógica e programação na linguagem COBOL, conceitos que carrego em minha bagagem e sempre me ajudaram na vida profissional e acadêmica.

A todos os mestres e profissionais com quem estudei e trabalhei, que sempre muito me ensinaram e me incentivaram a trabalhar com conhecimento e esmero.

Agradeço a Deus por todas as oportunidades que me foram colocadas na vida, e que sem elas o caminho para realizar este sonho, por tanto tempo esperado, não seria possível. Um antigo sonho aqui se realiza.

Resumo

A solução de problemas onde muitos componentes atuam e interagem simultaneamente requer modelos de representação nem sempre tratáveis pelos métodos analíticos tradicionais. Embora em muitos caso se possa prever o resultado com excelente precisão através de algoritmos de aprendizagem de máquina, a interpretação do fenómeno requer o entendimento de quais são e em que proporção atuam as variáveis mais importantes do processo. Esta dissertação apresenta a aplicação de um método onde as variáveis discriminantes são identificadas através de um processo iterativo de ranqueamento (*"ranking"*) por eliminação das que menos contribuem para o resultado, avaliando-se em cada etapa o impacto da redução de características nas métricas de acerto. O algoritmo de florestas de decisão (*Random Forest*) é utilizado para a classificação e sua propriedade de importância das características (*Feature Importance*) para o ranqueamento. Para a validação do método, dois trabalhos abordando sistemas complexos de natureza diferente foram realizados dando origem aos artigos aqui apresentados. O primeiro versa sobre a análise das relações entre programas maliciosos (*malware*) e os recursos requisitados pelos mesmos dentro de um ecossistema de aplicações no sistema operacional Android. Para realizar esse estudo, foram capturados dados, estruturados segundo uma ontologia definida no próprio artigo (*OntoPermEco*), de 4.570 aplicações (2.150 malware, 2.420 benignas). O modelo complexo produziu um grafo com cerca de 55.000 nós e 120.000 arestas, o qual foi transformado usando-se a técnica de bolsa de grafos (*Bag Of Graphs*) em vetores de características de cada aplicação com 8.950 elementos. Utilizando-se apenas os dados do manifesto atingiu-se com esse modelo 88% de acurácia e 91% de precisão na previsão do comportamento malicioso ou não de uma aplicação, e o método proposto foi capaz de identificar 24 características relevantes na classificação e identificação de famílias de malwares, correspondendo a 70 nós no grafo do ecosistema. O segundo artigo versa sobre a identificação de regiões em um documento impresso que contém informações relevantes na atribuição da impressora laser que o imprimiu. O método de identificação de variáveis discriminantes foi aplicado sobre vetores obtidos a partir do uso do descritor de texturas (*CTGF-Convolutional Texture Gradient Filter*) sobre a imagem scaneada em 600 DPI de 1.200 documentos impressos em 10 impressoras. A acurácia e precisão médias obtidas no processo de atribuição foram de 95,6% e 93,9% respectivamente. Após a atribuição da impressora origem a cada documento, 8 das 10 impressoras permitiram a identificação de variáveis discriminantes associadas univocamente a cada uma delas, podendo-se então visualizar na imagem do documento as regiões de interesse para uma análise pericial. Os objetivos propostos foram atingidos mostrando-se a eficácia do método proposto na análise de dois problemas em áreas diferentes (segurança de aplicações e forense digital) com modelos complexos e estruturas de representação bastante diferentes, obtendo-se um modelo reduzido interpretável para ambas as situações.

Abstract

Solving a problem where many components interact and affect results simultaneously requires models which sometimes are not treatable by traditional analytic methods. Although in many cases the result is predicted with excellent accuracy through machine learning algorithms, the interpretation of the phenomenon requires the understanding of how the most relevant variables contribute to the results. This dissertation presents an applied method where the discriminant variables are identified through an iterative ranking process. In each iteration, a classifier is trained and validated discarding variables that least contribute to the result and evaluating in each stage the impact of this reduction in the classification metrics. Classification uses the Random Forest algorithm, and the discarding decision applies using its feature importance property. The method handled two works approaching complex systems of different nature giving rise to the articles presented here. The first article deals with the analysis of the relations between *malware* and the operating system resources requested by them within an ecosystem of Android applications. Data structured according to an ontology defined in the article (*OntoPermEco*) were captured to carry out this study from 4,570 applications (2,150 malware, 2,420 benign). The complex model produced a graph of about 55,000 nodes and 120,000 edges, which was transformed using the *Bag of Graphs* technique into feature vectors of each application with 8,950 elements. The work accomplished 88% of accuracy and 91% of precision in predicting malicious behavior (or not) for an application using only the data available in the application's manifest, and the proposed method was able to identify 24 relevant features corresponding to only 70 nodes of the entire ecosystem graph. The second article is about to identify regions in a printed document that contains information relevant to the attribution of the laser printer that printed it. The discriminant variable determination method achieved average accuracy and precision of 95.6% and 93.9% respectively in the source printer attribution using a dataset of 1,200 documents printed on ten printers. Feature vectors were obtained from the scanned image at 600 DPI applying the texture descriptor *Convolutional Texture Gradient Filter (CTGF)*. After the assignment of the source printer to each document, eight of the ten printers allowed the identification of discriminant variables univocally associated to each one of them, and it was possible to visualize in document's image the regions of interest for expert analysis. The work in both articles accomplished the objective of reducing a complex system into an interpretable streamlined model demonstrating the effectiveness of the proposed method in the analysis of two problems in different areas (application security and digital forensics) with complex models and entirely different representation structures.

Contents

1 Introduction **11**

2 Articles Published and Submitted **15**

2.1 Leveraging ontologies and machine-learning techniques for malware detection in the Android ecosystem. 15

2.2 Connecting the Dots: Toward Accountable Machine-Learning Printer Attribution Methods 46

3 Discussion **63**

4 Conclusion **67**

Bibliography **69**

Chapter 1

Introduction

This introduction explains the objective of the method applied in both articles "Leveraging ontologies and machine-learning techniques for malware detection in the Android ecosystem" and "Connecting the Dots: Toward Accountable Machine-Learning Printer Attribution Methods" produced by the author during the master's program work. These articles are transcribed entirely in Chapter 2. Chapter 3 discusses the methodology, and the results obtained, proposing future lines of research. Finally, Chapter 4 concludes the work.

The method outlined herein is centered on the problem of streamlining a complex model to discover the elements which most influence the system behavior, favoring in this way the interpretability on the study of the observed data.

As depicted in the summary view provided in Figure 1.1 the method consists of three phases, where the core stage (learning phase) is based in a machine-learning algorithm to discover the most discriminant features. The start and end phases are two transformations: first is a vectorization to convert the complex model into the feature vectors to input in the core phase, and last an inverse transformation to reconstruct a streamlined model from vectors using only the most discriminant features.

The process starts with a pre-processing stage responsible for converting the symbolic model ($\mathcal{M}$) describing the phenomena and data observations ($\mathcal{O}$) into a representative mathematical notation in vector space. Feature vectors in this space represent relationships within the adopted model and measure the relevance of each feature for the class represented by the vector. The conversion is done by a transformation (τ) (Eq. 1.1) responsible for converting the model structure into the designed feature space.

$$\boldsymbol{F} = \tau(\mathcal{M}, \mathcal{O}) \tag{1.1}$$

where τ is the model-to-feature transformation, $\mathcal{M}$ is the original model, $\mathcal{O}$ represents the observations (data) structured according to the original model, and $\boldsymbol{F}$ denotes the created feature vectors in $\mathbb{R}^n$.

After the feature generation step, the objective of the learning phase is to isolate the most discriminant features to the classes of interest recursively identifying discriminative features while maximizing an adopted classifier performance criterion (Eq. 1.2).

$$\boldsymbol{H} = \boldsymbol{F}(\mathcal{I}^\star) \quad | \quad \mathcal{I}^\star = \arg\max_{\mathcal{I}} \quad f(\mathcal{C}(\boldsymbol{F}, \mathcal{I})) \tag{1.2}$$

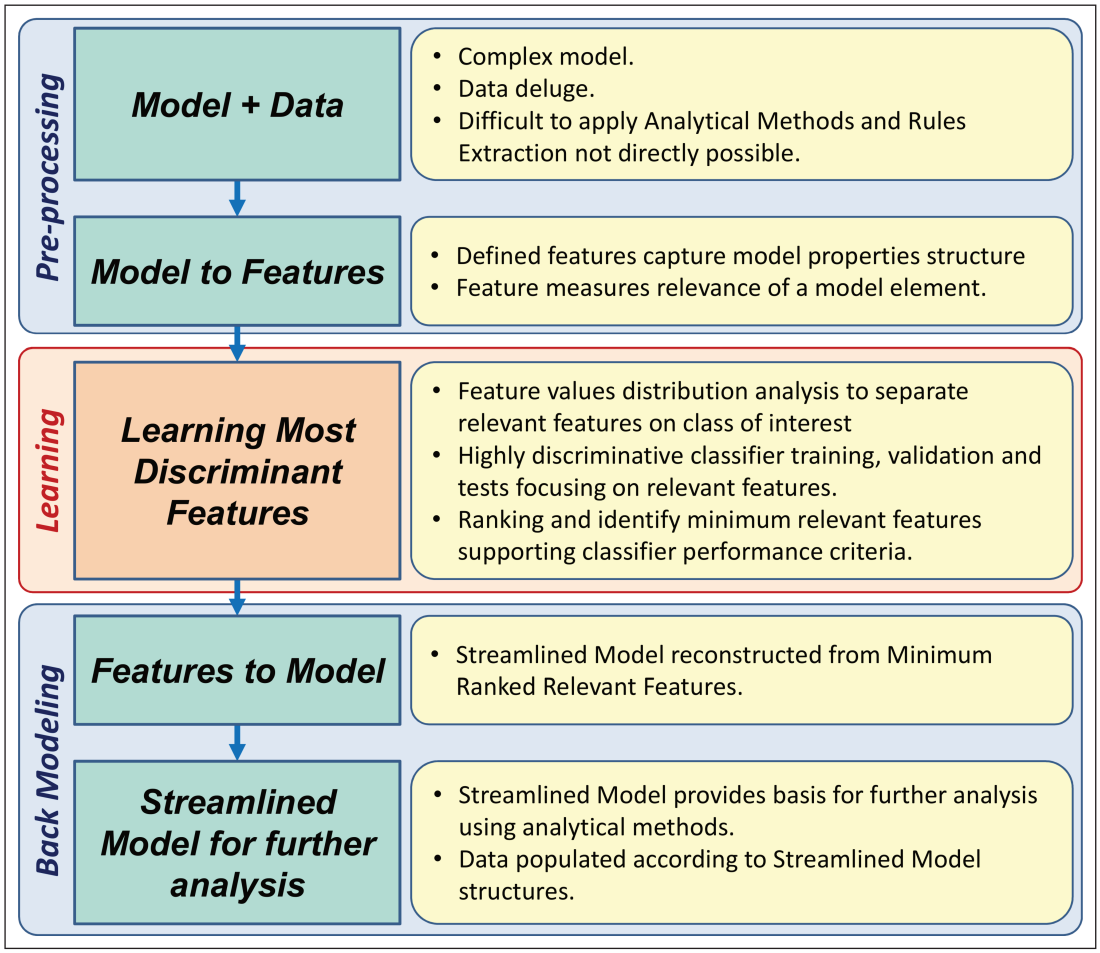

Figure 1.1: Bird's-eye view of the applied method on both articles presented in Chapter 2.

where $\mathcal{C}$ is a classifier, $\boldsymbol{F}$ is the vector of all features in $\mathbb{R}^n$, $\mathcal{I}$ is an index set with at most $m < n$ elements, $\boldsymbol{F}(\mathcal{I}^\star)$ is the optimal subset of $\boldsymbol{F}$ obtained by keeping only the indices $\mathcal{I}$, f is a function that scores the performance of a classifier using the features $\boldsymbol{F}(\mathcal{I})$, $\boldsymbol{H}$ is a vector with the m most discriminative features in $\mathbb{R}^m$.

Given the most discriminative features ($\boldsymbol{H}$) selected by the learning phase, a back-transformation Eq. 1.3 is applied to the original model and original data using the selected features. The back transformation (τ') produces a simplified model and isolates data that represent an empirical approximation of the original phenomenon.

$$\mathcal{S} = \tau'(\mathcal{M}, \mathcal{O}, \boldsymbol{H}) \tag{1.3}$$

where τ' is the back transformation operation, $\mathcal{M}$ is the original model, $\mathcal{O}$ represents the observations (data) structured according to the original model, $\mathcal{S}$ is the streamlined or refined model, $\boldsymbol{H}$ indicates the most discriminative features vector in $\mathbb{R}^m$.

As the phenomena could have different forms of representation, the pre-processing and the back modeling phases are specifically related to the model of each problem. Conversely, the learning phase is a common approach for all problems reusing the same computational programs and interfaces. This is one of the advantages of the applied method in both

problems addressed in the two articles compiled in Chapter 2.

In Android malware relationships analysis Section 2.1, the features extracted from the ontology graph uses the Bag of Words technique [30, 29] with a feature definition that captures the three node-path relationships focusing on the intermediate node and classes of extreme nodes. The original problem model is an ontology representing an ecosystem of applications in an Android device. It generates a huge graph with approximately 50,000 nodes and 120,000 edges which is difficult to address to find elements related to the malware behavior of one application. The method applied to the complex ecosystem model could identify 24 discriminant features that reversed back into 70 nodes of the original ecosystem graph. These nodes were associated with most of the malware families resource requests explaining the results achieved.

In Section 2.2 the problem is to identify what regions on a questioned document contain image signatures generally produced by mechanisms behavior and electromechanical defects, which can link the document with the laser printer device that printed it. The initial model is a scanned image, and the final objective is to pinpoint in that image pixels which represents the most discriminant textures attached to the specific printer attributed to the questioned document. The CTGF texture descriptor previously defined in the article [15], with this dissertation's author collaboration, was used to capture printing imperfections locally represented providing the ground information for modeling and back projecting the discriminant regions of each document attributed to a specific laser printer.

Using 1,200 documents printed on ten printers the method described above achieved average accuracy of 95.6% and precision of 93.9% in the source printer attribution. Most discriminant features identified for each of the ten printers allowed in eight of them to produce maps over the original scanned images pinpointing the regions used by the algorithm to discriminate the source printer, which is a remarkable achievement for the use of the classification techniques in digital forensics and reports of technical expertise for legal purposes.

As previously mentioned, the focus here on problem's resolution is not only to develop a machine learning classifier which can successfully predict the system behavior based on the input data but, more than that, which data structures are involved in that determination.

Looking at this problem, we realized that it is part of a more generic research topic in machine learning, which is currently called accountability for the classification results.

As far as we know, the CTGFmap article (Section 2.2) is the first work in forensics to deal with accountable printer attribution.

In the last few years, there has been an increasing concern and interest in accountable machine learning. New dedicated conferences and workshops such as the ones promoted by FAT/ML organization [13], ICML – Workshop on Human Interpretability in Machine Learning (WHI) [33], AAAI-2017 – W11 Workshop Human-Aware Artificial Intelligence [31], and IJCAI2017 – Workshop on eXplainable Artificial Intelligence (XAI) [1] focus on fairness, accountability, and transparency concepts for machine learning algorithms and applications.

Governments and non-governmental organizations, just to cite some, European Parliament (GDPR - General Data Protection Regulation) [17, 27] and Center for Democracy & Technology (CDT) [8], are issuing policies, directives and best practices concerning the use of technology, and recently focusing on consequences and human rights related to

decisions made by algorithms.

Most of the misgivings are due to the misuse and ethics of machine learning applications and the human rights of data privacy, but also on how to demonstrate decision results in a way that humans can understand. Most publications in this area address one or more principles exposed by the authors of the article [10] in "MIT Technology Review in November/2016: Responsibility, Explainability, Accuracy, Auditability, Fairness".

Explainability is defined in FAT/ML organization Principles for Accountable Algorithms [11] as: "Ensure that algorithmic decisions as well as any data driving those decisions can be explained to end-users and other stakeholders in non-technical terms.". W11 workshop at AAAI-17 (Workshop Human-Aware Artificial Intelligence of Thirty-First AAAI Conference on Artificial Intelligence) explains the objectives of the workshop [31] as: "In order to address this issue and produce truly human-aware artificial intelligence, systems must try to solve the interaction issues that accompany each unique application domain. These interaction issues may broadly be divided into extraction (or interpretation) challenges, and presentation (or steering) challenges".

Closing the loop a new frontier on machine learning studies is how algorithms and humans can cooperate systematically complementing each other and work together in a process to achieve better results. This is one of the topics of the workshop W11 workshop at AAAI-17 [31]: "The key premise of this workshop is based on the idea that augmented intelligence – that is, teams and systems that combine the skills of humans and AI techniques – can achieve better performance than either alone. However, in order to create such systems with augmented intelligence, humans must be accommodated as first-class citizens in the decision-making loop of existing AI systems. Far too often, traditional AI systems have tended to exclude humans (and the problems that accompany interaction with them) and have instead focused on producing optimal artifacts that stand no significant chance to working in the real world.".

The concepts and related work sections in both articles compiled in the next chapter explain in more detail concepts, models, and transformations brought to bear.

Chapter 2

Articles Published and Submitted

This chapter presents a summary and compilation of the articles produced during the master's program with the contribution of the author of this dissertation. Articles Section 2.1 and Section 2.2 are presented in reverse order of writing, i.e., most recent article first.

2.1 Leveraging ontologies and machine-learning techniques for malware detection in the Android ecosystem.

Title: Leveraging Ontologies and Machine-learning Techniques for Malware Analysis into Android Permissions Ecosystems.
Authors: Luiz C. Navarro, Alexandre K. W. Navarro, André Grégio, Anderson Rocha, Ricardo Dahab.
Status: Submitted to a journal.

Leveraging Ontologies and Machine-learning Techniques for Malware Analysis into Android Permissions Ecosystems

Luiz C. Navarro[a], Alexandre K. W. Navarro[b], André Grégio[c], Anderson Rocha[a], Ricardo Dahab[a]

[a] *Institute of Computing - University of Campinas (Unicamp), Campinas, SP, Brazil*
[b] *Engineering Department - University of Cambridge - Cambridge, UK*
[c] *Department of Informatics - Federal University of Paraná (UFPR), Curitiba, PR, Brazil*

Abstract

Smartphones form a complex application ecosystem with a myriad of components, properties, and interfaces that produce an intricate relationship network. Given the intrinsic complexity of this system, we hereby propose two main contributions. First, we devise a methodology to systematically determine and analyze the complex relationship network among components, properties, and interfaces associated with the permission mechanism in Android ecosystems. Second, we investigate whether it is possible to identify characteristics shared by malware samples at this high level of abstraction that could be leveraged to unveil their presence. We propose an ontology-based framework to model the relationships between application and system elements, together with a machine-learning approach to analyze the complex network that arises therefrom. We represent the ontological model for the considered Android ecosystem with 4,570 apps through a graph with some 55,000 nodes and 120,000 edges. Experiments have shown that a classifier operating on top of this complex representation can achieve an accuracy of 88% and precision of 91% and is capable of identifying and determining 24 features that correspond to 70 important graph nodes related to malware activity, which is a remarkable feat for security.

Keywords: Malware, Android Permissions, Ontology, Bags of Graphs, Machine Learning, Discriminant Features.

1. Introduction

Smartphones have become ubiquitous computing devices worldwide. A recent Ericsson Mobility Report [1] indicated that smartphones currently represent 55% of all mobile subscriptions globally. The report further projects the number of unique mobile subscribers to reach 6.1 billion by 2022, covering roughly 75% of the world's population. Despite the multitude of different device models and the availability of several different operating systems for smartphones, the Android operating system currently holds 88% of market share [2].

Mobile devices are increasingly being used for activities that directly impact social, work, and financial environments; as such, they have become a primary target for cyber-criminals. A study published by Deloitte [3] concluded that, in the United Kingdom, the top ten usages for smartphones include social networking, emailing, banking, and shopping with similar patterns across other developed countries. To the eyes of a cyber-criminal, social

Email addresses: luiz.navarro@students.ic.unicamp.br (Luiz C. Navarro), akwn2@cam.ac.uk (Alexandre K. W. Navarro), gregio@inf.ufpr.br (André Grégio), anderson.rocha@ic.unicamp.br (Anderson Rocha), rdahab@ic.unicamp.br (Ricardo Dahab)

networks can be viewed as a repository of the smartphone user's personal information; work-related emails are a potential source of sensitive information, and banking apps are the gateway for accessing the user's finances [4, 5, 6, 7].

As a prophylactic security measure against unauthorized use or access, the Android ecosystem possesses a permission system for its applications (apps) [8]. The permission system informs the user of which system resources and information an app uses prior to installation so that the user can make an informed choice on whether or not to install that app based on the resources used. However, Kelley et al. [9] and Felt et al. [10] have shown flaws in the use of the permission system as a preventive security measure. In particular, users tend not to pay attention to permissions, and more worryingly, permission systems sometimes fail to aid users with the task of properly taking security-related decisions. Furthermore, developers tend to overprivilege applications requesting more permissions than necessary, anticipating future releases [11, 12]. Moreover, Android documentation also has flaws in mapping permissions related to system calls, as described in the study from Pscout developers [13], a software that intercepts system calls and keeps track of which permissions are tested by the operating system, producing actual documentation about which permissions are verified in each system-call access.

As a matter of fact, malicious apps can control seemingly harmless system resources to exploit a vulnerability in another app [9] indirectly. Given that the Android ecosystem has over 1.7 million apps and 235 different permissions [14], the task of mapping and analyzing relationships among permissions, malware, and benign apps is daunting and, undoubtedly, cannot be manually performed by a human curator. Likewise, any developed methodology must be extensible, automatic, and dynamic to allow for new characteristics to be taken into consideration on the fly as apps, malware, and permissions are continuously added or removed from the ecosystem.

Given the above, application testing in Android devices faces important challenges [15] that must be addressed. Within this context, the present contribution proposes two methods (described in Section Section 4): the first for mapping relationships in the Android ecosystem using ontologies and the second, a machine-learning-based solution to analyze malware features from the obtained network of relations and dependencies. We validate the effectiveness of these methods in Section 4 and show that the proposed methods are able to determine the most important nodes related to malware activity, representing an important contribution to smartphone security.

2. Concepts and Related Work

Before we move on to the new methods we propose in this paper, we present a brief introduction to Android security, ontologies, and feature engineering using bags of graphs as well as the random forests classifier, which are necessary concepts to understand the paper. The expert reader can go directly to Section 3, where the new methods are introduced.

2.1. Android Security System and Malware Identification

Android elevates the definition of operating systems to a whole new level. Operating systems constitute the software responsible for managing computer resources (hardware and software) and provide users' computer programs with common APIs to facilitate access to available resources and services while hiding technical details involved in basic operations from end-users. Android, in turn, implements a Java Framework over a

Linux Operating System. This architecture allows the development of smartphone apps, providing users with a high-level interface for accessing organized data, ranging from user contacts stored in a simple database to GUI components and phone resources. It also acts as a security measure to isolate apps and facilitate environment administration with effortless installation and application management.

In the ubiquitous context of smartphones in which Android is inserted, the concept of security based on the user's login is not the sole solution. Linux security options based on the user's privileges for program spaces, data files, and application-context controls are too complex for an information and technology layman. Instead, Android relies on an application-based security system, in which each application runs a single and different user. This design guarantees a completely separate sandbox environment for each application within Linux at the expense of a duplicate Java Framework being created for each application upon installation.

The access to resources is controlled by wrapped Linux interfaces while system calls are managed by a separate Android system application. This application receives requests from each app's Java machine[1] through an internal remote procedure call (RPC) communications mechanism. This mechanism guarantees a security environment based on permissions [Permissions, 16]. The very same mechanism also implements object-based inter-app communication referred to as the intents [Intent and Intent Filters, 16].

Apps declare permission requests and can also define permissions to protect interfaces. This declaration is located in a file named `Application Manifest`, which contains a high-level definition of the structure of the application components, interfaces, and permissions. This file is used by the system when installing an app to grant the app access to permissions according to rules determined in the system.

On Android version 5.0 (Lollipop) and previous ones, to install applications, the smartphone user needs to accept all permissions requested by the app. Denying any of the requested permissions prevents the application from being installed. This mechanism transfers the responsibility of authorizing interactions to users who often do not understand the implications of such actions. Therefore, it is not surprising that permissions represent one of the most important backdoors exploited by malicious software in those older versions [10].

Android version 6.0 introduced a new permissions control that allows one to enable or disable permissions individually even after installation. The experiments conducted in this article used an applications ecosystem of a smartphone with Android version 4.4 (KitKat). After KitKat, several changes were made for version 5.0 (Lollipop) aiming at improving performance and mainly security. The replacement of Dalvik Java runtime machine by ART and the deployment of SELinux in enforcement mode for all domains are two clear examples. More details on Android security mechanisms can be found in [17].

Malware in the Android ecosystem typically targets personal data as smartphones have become a preferred personal interface for banking, electronic payments, online retail stores, and other critical usages. They can exploit vulnerabilities within the internal Android system code at either the Java Framework or Linux levels. Alternatively, they can simply mislead users to provide access to resources about which they are not even aware [18],

[1] Dalvik for versions 4.4 KitKat and earlier or Android Runtime ART for versions 5.0 Lollipop and later.

including repackaging actual benign applications to include malicious code [19]. Most tools for malware detection directly analyze any piece of available code to determine malicious activities [20], thus requiring known pieces of code to be used as signatures. Steering away from direct code analyses, machine-learning methods form powerful tools to identify malicious behavior over logs and information collected from system interface calls. Sanz et al. [21] pioneered the analysis of app permission requests through a set of machine-learning solutions to discriminate malware from benign apps. Likewise, Das et al. [22] also adopted a machine-learning approach to analyze and identify malware behavior in real time with the aid of additional hardware.

Most of the current state of the art malware detection approaches relies on machine learning analysis with different algorithms as we can see in the survey article [23], but all of them capture information using static code analysis tools or execution monitoring applications, or both techniques, to devise features for the machine learning processing. Normally, the deeper the level of analysis, the better the results in the malware detection rate (Tpr), as we can see in Table 1. To complement existing techniques with a different vantage point, our work looks at relations at the highest level of information available, and surprisingly yields a classification rate of 86% in the malware identification.

As we can see in the survey article [23], most current state-of-the-art malware detection approaches rely on machine learning analysis with different algorithms, but all of them capture information using static code analysis tools or execution monitoring applications (or both techniques) to devise the features for the machine-learning processing. The more information and the deeper the level of analysis, the better are the results of the malware detection rate (Tpr) as we can see in Table 1. Our work looks at relations at the highest level of information available and, surprisingly, achieves 86% malware identification, using only 36 most discriminant permissions related features detected in the ranking process.

Table 1: Malware detection state of the art comparison.

Malware detection work	**Tpr (%)**	**Captured and analyzed data**
MADAM[24]	96.9	Multi-level static code analysis and extensive monitoring of kernel and user activity.
DroidSIFT[25]	93	Static code analysis - API calls dependencies graph analysis.
APK Auditor[26]	88.3	Static code analysis: permissions, services and receivers.
Our work - most discriminant features	86	Manifest only data: declared permissions and interfaces.

2.2. Ontologies

Ontologies provide a framework for modeling concepts and their relationships using formal logic. Based on the treatments provided by Gruber [27], Guarino et al. [28], and Studer et al. [29], an ontology can be succinctly defined as the tuple $\mathcal{O}=(\mathcal{T},\mathcal{A},\mathcal{L})$, which consists of a set of classes and properties referred to as a terminology $\mathcal{T}$, a set of axioms $\mathcal{A}$ that relates two instances of the terminology, and a formal logic language $\mathcal{L}$ that allows the description of the axioms and logical inference.

One of the strengths of this general framework resides in expressing models in an extensible way: new terminology elements can be linked back to existing terminology elements through axioms. The embedded formal logic enables the discovery and inference of implicitly defined relationship values for relationships not previously assigned.

As a consequence of their high interpretative power, ontologies have been successfully used in many different setups. Applications include automatic curation of protein-protein databases [30], management of genomic sequence databases [31], integration of different biomedical databases [32], document mapping of virtual organizations [33], and description of sustainable business models [34].

Ontologies and semantic web technologies are growing fast with the standardization of languages, tools, and applications that can provide integration, interoperability, and reuse of data in multidisciplinary studies [35].

In the forefront of the systems security arena, most current ontologies focus on the taxonomy of threats and countermeasures [36], as well as security risk assessment [37, 38]]. Nguyen et al. [39] provide an overview of ontology concepts, tools, and applications in information and security systems. In turn, the authors in [40] explore the use of ontologies to model software architectural styles using UML-based ontology.

For the purposes intended in this contribution, ontologies will be described as a graph in which axioms are expressed by an edge between two nodes representing the instances of terminology linked by the property within each axiom.

2.3. Bags of Graphs

A Bag of Graphs (BoG) is a technique used to vectorize a graph representation creating a set of symbols (items of the BoG), which represent elements or subgraphs included in the graph. This concept was first introduced by Silva et al. [41, 42].

Given a graph $G=(V,E)$ and a function $w_i=f(G,i)$, which extracts a symbol defined as an ordered sequence $w_i=(p_1,...,p_m)$ of elements $p_j \in G$, a BoG (Bag of Graphs) can be defined as the set $B=\{w_1,...,w_n\}$ of all symbols extracted from G by f. Subgraphs $S \subset G$ can be described as a set of items $W \subset B$.

In this article, we use an ontology representation for the application ecosystem, which produces a large graph encompassing the relationships among applications. As we are interested in finding common sets of properties, which appear on malware' subgraphs, a BoG representation is suitable for our problem. There are two advantages to adopting BoG modeling in this work. First, the use of logic in the ontology to find malware properties alleviates the fact that we do not know which properties are common to malware beforehand. Second, as the ecosystem's graph grows, so does its complexity, but exponentially. Converting the subgraph of each application in the ecosystem into BoG vectors enables us to use them with a machine-learning process to identify the most discriminant features that characterize malware apps in the studied ecosystem.

2.4. Random Forest Feature Importance

In this section, we provide a succinct description of the Random Forest feature importance analysis, a key concept we rely on in this paper. Originally introduced by Breiman [43], random forests represent a practical framework for performing classification and regression tasks, having attracted attention both in the academia [44]

and the industry [45, 46, 47]. In short, random forests combine classification and regression trees (CART) [48] with a bootstrap aggregation technique, also referred to as bagging [49].

Classification and regression tree models represent mathematical functions by partitioning the space of the inputs and forming a local function approximation for each partition. Each of the decisions to split the input space into two can be represented as introducing a node in a binary tree representing the different regions in the input space. This tessellation of the input space is performed to minimize a classification or regression loss function that assesses the goodness-of-fit. To avoid over-fitting, a pruning step can be used during the process of growing the tree to reduce its complexity [48, 50].

CART models are highly dependent on the data used in their training as they often rely on greedy methodologies to evaluate the feature space. One approach to alleviating such high dependence is to use the bagging technique [49]. This technique subsamples the training set to produce K different sets, with which classification (or regression) tree models can be trained. Then the predictions from each of the K learned models are averaged to produce a single final estimate. This averaging of different trees reduces the impact of each single data point in the overall estimation process.

Bagging can also be leveraged to provide an estimate for the generalization error and the probability of a given node in a tree. Breiman [51] shows that each point (x_i,y_i) of the original set is only used to train 63% of the K trees when bagging is applied to CART models. Hence, (x_i,y_i) can be used in the remaining trees as a cross-validation measure, allowing us to calculate what is known as out-of-bag (OOB) error. This error can be used to assess the importance of a given feature in the predictions of a forest by randomly permuting the values of a given feature [43]. When a feature is correlated with others, additional care should be taken to avoid overestimating the importance of correlated variables, as described by Strobl et al. [52] and Altmann et al.[53].

3. Proposed Method

In this work, our primary goal is to analyze which permissions and resources are related to malicious apps in the Android ecosystem as represented in the Android manifests. We rely solely upon application manifest XML files as our source of information. The reasoning for this choice is that such files are publicly available and do not require any reverse engineering, code execution monitoring, or complicated code-level analysis to detect the presence of malware in a system, as described in [54]. We have adopted the complex ecosystem of a commercial Android smartphone loaded with its default operating system and apps provided in the factory software image as well as benign applications downloaded from official app stores and applications known to be malware from security research repositories.

We are not focused on finding resources and relationships directly involved in an attack itself. Rather, we aim to understand which structures and relationships are important for malware activities. As an example, malware samples that steal users' contact information require network access to send this information to an agent external to the smartphone. Although network resources may not be directly involved in the attack, it is clear that they are an important part of the malware structure and how it interacts with the operating system.

To gather information available in the application manifests, we first need to define how to build a model capable of establishing the involved entities and their relationships, while still retaining a desirable analysis foundation to be used later on. While a traditional database appears to be a good choice to store all captured data while supplying advanced querying capabilities, ontologies are more advantageous in this case, as they are focused on semantics and standardization of concepts and are easy to evolve with by definition [55, 56, 57].

On the one hand, ontologies provide a good foundation for model building. On the other, the application of ontology logic through queries over the knowledge base to represent complex phenomena is difficult to determine experimentally, and even if we can find this relationship, the complex logic queries over large datasets normally require too much processing. Then, following a different path, our method relies on machine learning concepts to discover important relationships which explain the phenomena and use that result to streamline the ontology knowledge base itself.

To process the original model by machine-learning algorithms, the properties and relationships associated with an entity to be analyzed (class of interest) must be represented by individual measurable elements in vector form, referred to as features. Then all entities in the ecosystem are represented by features, which feed ML-based classifiers to determine their predictability performance, and which are the most discriminative ones for the decision-making process.

As an ontology model can be built and populated with data from application manifests, this model is so complex that trivial statistical analysis and queries may fail to identify important relationships related to the behavior of malware apps. To deal with this problem, we devised a way of transforming the relationships for each application inside the ecosystem into meaningful representative features, aiming at singling out the most important ones, reducing the model complexity of the malware structure.

The method outlined in Figure 2 is centered on the problem of simplifying an ontology model to map the most important malware relationships in the Android smartphone ecosystem. This is a particular case of a generalized approach we call conceptual framework, as illustrated in Figure 1. The purpose of this framework is to simplify models by transforming a symbolic representation into features, then determining the most discriminative features through the use of a machine-learning process. Finally, it transforms the selected features into a refined/simplified description model.

In our proposed conceptual framework, we start with a pre-processing stage responsible for converting the symbolic model $\mathcal{M}$ describing the phenomena and data observations (O) into a representative mathematical notation in vector space. Feature vectors in this space represent relationships within the adopted model and measure the relevance of each feature for the class represented by the vector. This is done by a transformation τ (Eq. 1) responsible for converting the model structure into the designed feature space.

$$\boldsymbol{F}=\tau(\mathcal{M},\mathcal{O}) \tag{1}$$

where τ is the model-to-feature transformation, $\mathcal{M}$ is the original model, $\mathcal{O}$ represents the observations (data) structured according to the original model, and $\boldsymbol{F}$ denotes the created feature vectors in $\mathbb{R}^n$.

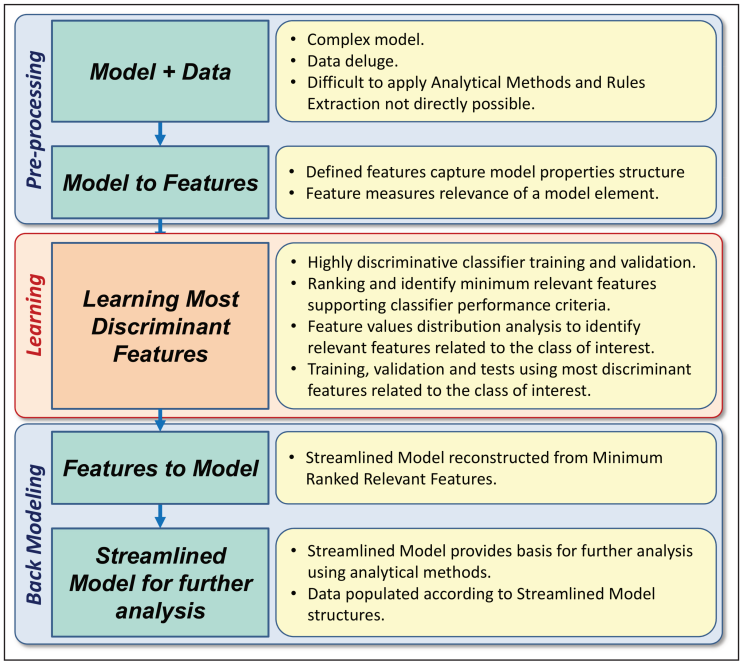

Figure 1: Bird's-eye view of the proposed method for discovering which permissions and resources are likely related to malware apps in the Android ecosystem.

After the feature generation step, the objective of the learning phase is to isolate the most discriminant features to the classes of interest recursively identifying discriminative features while maximizing an adopted classifier performance criterion (Eq. 2).

$$\boldsymbol{H} = \boldsymbol{F}(\mathcal{I}^{\star}) \quad | \quad \mathcal{I}^{\star} = \underset{\mathcal{I}}{\operatorname{argmax}} \; f(\mathcal{C}(\boldsymbol{F}, \mathcal{I})) \tag{2}$$

where $\mathcal{C}$ is a classifier, $\boldsymbol{F}$ is the vector of all features in $\mathbb{R}^n$, $\mathcal{I}$ is an index set with at most $m < n$ elements, $\boldsymbol{F}(\mathcal{I}^{\star})$ is the optimal subset of $\boldsymbol{F}$ obtained by keeping only the indices $\mathcal{I}$, f is a function that scores the performance of a classifier using the features $\boldsymbol{F}(\mathcal{I})$, $\boldsymbol{H}$ is a vector with the m most discriminative features in $\mathbb{R}^m$.

Given the most discriminative features $\boldsymbol{H}$ selected by the learning phase, a back-transformation Eq. 3 is applied to the original model and original data using the selected features. The back transformation τ' produces a simplified model and isolates data that represent an empirical approximation of the original phenomenon.

$$\mathcal{S} = \tau'(\mathcal{M}, \mathcal{O}, \boldsymbol{H}) \tag{3}$$

where τ' is the back transformation operation, $\mathcal{M}$ is the original model, $\mathcal{O}$ represents the observations (data) structured according to the original model, $\mathcal{S}$ is the streamlined or refined model, $\boldsymbol{H}$ indicates the most discriminative features vector in $\mathbb{R}^m$.

Given the problem of understanding the most important structures associated with malware on Android resources and permissions, we applied the conceptual framework using as the original model an ontology (AndroPermEco ontology terminology) created from data observations (ontology axioms) captured from the application's manifests of the Android ecosystem and a BoG transformation to map the ontology graph onto feature vectors representing Android apps.

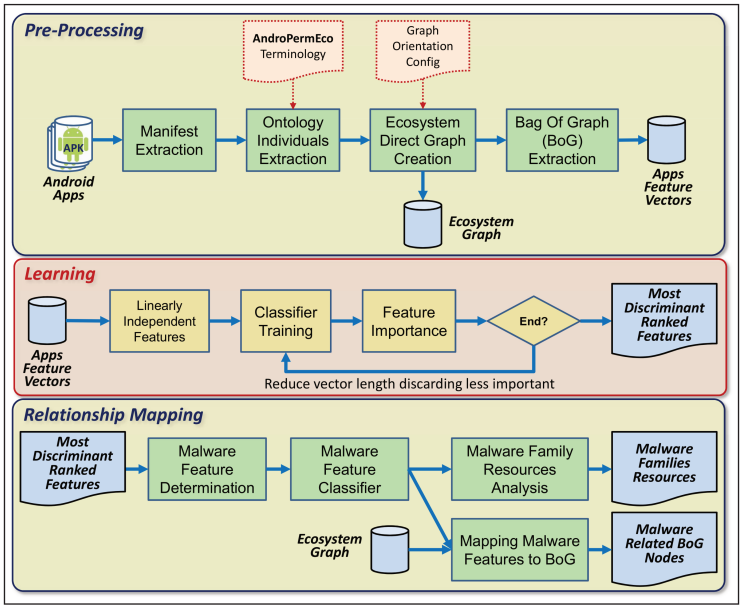

Figure 2: Refined view of the proposed method for discovering which permissions and resources are likely related to malware apps in the Android ecosystem.

Finally, the back-modeling process creates a refined model out of the most discriminant features, thus reducing the complex original ontology graph pointing out nodes related to the malware identification.

Figure 2 illustrates the process, while the following sections explain each individual step thereof.

3.1. Building Ecosystem Ontology Graph from Manifests

Android apps are deployed for installation as a package file using the Android Package Kit (APK) format (in fact a zip file). The APK file contains code and all necessary information for installation and configuration, including an Android XML file, the manifest. There, the developer describes resources used and defined in the app, as well as permissions (defined by the app and with requested access).

All analyses and investigations described herein relied upon the Android ecosystem version 4.4 KitKat. We defined an ontology for describing the permissions ecosystem of Android apps that we called AndroPermEco. This ontology describes relationships among (i) permissions defined by apps; (ii) permissions used by apps; (iii) interfaces protected by permissions; and (iv) resources accessed through the interfaces. Consequently, it identifies data, resources, and components shared by apps that provide services and others that consume those services. The complete terminology for AndroPermEco is presented in Appendix A. In this paper, we rely on the Web Ontology Language (WOL) with RDF/Turtle syntax to express ontologies, as defined in W3C Recommendation [58].

Ontologies, as previously discussed, can be represented by a graph, whereby each relation between individuals is expressed by a node for the subject and another node for the object, in addition to the edge of the property connecting them. Edges representing properties in ontology graphs are directed from subject node to object node. As we intend to traverse the graph in a direction which represents the relationship direction from origin to target, or consumer to provider, we introduce a direction configuration for each property in the ontology graph, representing it as shown in Figure 3. This directed graph of ontology for the entire ecosystem reflects

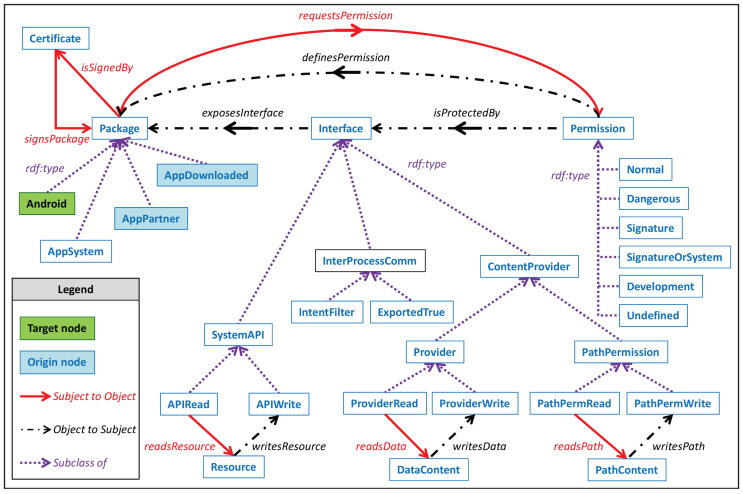

Figure 3: Android Permissions Ecosystem Ontology — Terminology Directed Graph.

relationships created by the coexistence of those apps in the same device. We prefer to keep the AndroPermEco definition and configure the direction of the graph to transverse each property rather than changing the property to passive voice (thus transforming objects into subjects and vice-versa). **AndroPermEco Graph:**

$$G_{onto} = (V_{onto}, E_{onto}) \tag{4}$$

where V_{onto} is the set of nodes of object instances in ecosystem ontology, E_{onto} is the set of edges of property instances in ecosystem ontology.

All the information needed to build the AndroPermEco ontology graph for the apps ecosystem can be obtained directly from manifest files of apps. We build a Python program that automatically extracts the manifest (XML) and certificate (text) files inside a given APK, parsing them to gather axioms expressed in the AndroPermEco terminology. The terminology guides the extraction program to generate the N3 lines according to each tag captured in the XML file.

3.2. Extracting Features from Ontology Graph

Our framework relies upon a machine-learning methodology to identify the most important features; thus, we need to define a vectorization process that creates a representation for the malware and benign app subgraphs within the entire ecosystem graph. This is also useful to identify nodes on malware graphs, which frequently appear connecting classes of objects. At this point, we are looking for specific ontology individuals, i.e., bridges connecting two structural elements not directly connected by their relationships.

Figure 4 provides an example of a simple ontology statement expressed in the RDF/Turtle language converted to the item of the Bag of Graphs (BoG). Although this is a simple example, it contains all concepts applied to the complex ecosystem of Android permissions transformation into apps feature vectors as we describe herein.

Each edge on the ontology graph represents an ontology property, which is defined as a relationship between classes. Subclasses inherit the properties defined for the superclasses; then, when we express an edge by the property name, we are already capturing the classes defined for the property. Nevertheless, it may not represent

effectively the class of the nodes connected, as they can be subclasses of the defined relationship. As seen in Figure 4, ***email*** and ***browser*** subclasses inherited the property ***requests***, which link both to the ***permission*** class, and consequently to the ***dangerous*** subclass. The same occurs with ***dangerous*** and ***protocol***, which inherited properties from their superclasses ***permissions*** and ***network***, respectively. This is the reason for including the class of origin and destinations on the items of BoG representation.

Representing all intermediate nodes (nodes with at least one in-going edge and one out-going edge) with all combinations of their incoming and outgoing edges, which leads to a finite set of features. Then, each subgraph of malware or benign application can be described by the presence (or absence) of those elements on the features set. This technique is known as a bag of graphs and was described in Section 2.3.

We define a bag of graphs (Ω) as a set of tuples (ω) constructed with the information of three consecutive nodes in the ontology DAG $G_{onto} = (V_{onto}, E_{onto})$ where each node $v \in V_{onto}$ is a subject or object of the property (relationship) $e \in E_{onto}$, connecting v to another node. In this way, each node $v \in V_{onto}$ has a unique class determined in the terminology of AndroPermEco, which we represent as a class $C \in \Psi$, where Psi is the space of classes in the AndroPermEco terminology.

Using the example in Figure 4 nodes: *OpenMsger*, *SEND_SMTP* and *SMTP* belongs to the nodes set V_{onto} of the graph with respective classes C in Ψ: *email*, *dangerous* and *protocol* .

```
############################################################
#                                                          #
# Subject       Predicate           Object                 #
#                                                          #
########################## Classes ##########################
:application    rdf:type            owl:Class .
:email          rdf:type            owl:Class ;
                rdfs:subClassOf     :application .
:browser        rdf:type            owl:Class ;
                rdfs:subClassOf     :application .
:permission     rdf:type            owl:Class .
:dangerous      rdf:type            owl:Class ;
                rdfs:subClassOf     :permission .
:network        rdf:type            owl:Class .
:protocol       rdf:type            owl:Class ;
                rdfs:subClassOf     :network .
################### Object Properties ###################
:requests       rdf:type            owl:ObjectProperty ;
                rdfs:domain         :application ;
                rdfs:range          :permission .
:protects       rdf:type            owl:ObjectProperty ;
                rdfs:domain         :permission ;
                rdfs:range          :network .
###################### Individuals ######################
:SMTP           rdf:type            owl:NamedIndividual ,
                                    :protocol .
:SEND_SMTP      rdf:type            owl:NamedIndividual ,
                                    :dangerous ;
                :protects           :SMTP .
:OpenMsger      rdf:type            owl:NamedIndividual ,
                                    :email ;
                :requests           :SEND_SMTP .
############################################################
```

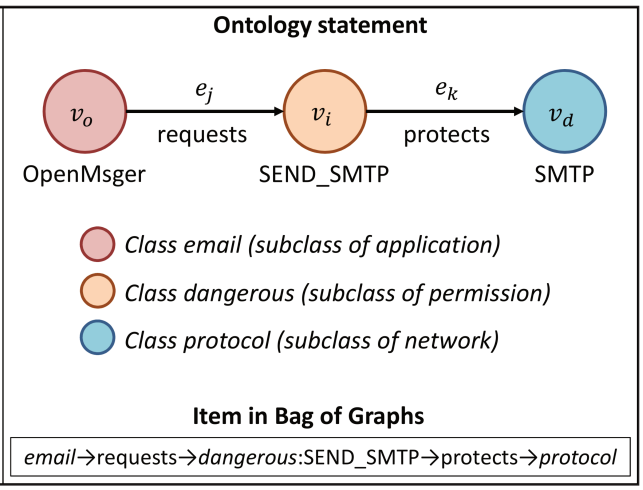

Figure 4: Toy-example of a simple ontology statement conversion to a bag of graph items following the rules established in this work.

As we are interested in representing nodes which act as bridges between classes (we can consider that a new property that links two properties), we represent in the tuple only the class of the origin and destination nodes, with the intermediate node represented by the instance and class of the node. One intermediate node can participate in many Bag of Graphs item, as it can be linked with many incoming and outcome nodes. Then the symbol ω_t in the Bag of Graphs Ω is defined as the tuple:

$$\omega_t = (C_o, e_j, C_i, v_i, e_k, C_d)$$

where: $v_o, v_i, v_d \in V$; $e_j, e_k \in E$; $e_j = (v_o, v_i)$; $e_k = (v_i, v_d)$; $v_o \in C_o, v_i \in C_i, v_d \in C_d$; $C_o \in \Psi, C_i \in \Psi, C_d \in \Psi$;

The BoG Ω is the set of all items ω_t that can be extracted from the graph G.

$$\Omega = \quad \{\omega_1 ... \omega_t ... \omega_n\} \quad \forall t \subset G$$
$$n = \text{ number of elements in } \Omega$$

The BoG Ω, extracted from the entire ecosystem graph G, is the feature set to be used to describe the investigated subgraphs — the apps downloaded into the Android device ecosystem. As the feature set is defined by the ordered elements in Ω, we need to establish a rule to compute each feature value and how to extract the features corresponding to each application (malware or benign) to be analyzed by the machine learning steps.

Each application is represented by a starting node of the class package, and our apps of interest are particularly the ones of subclasses AppPartner or AppDownloaded (as seen in Figure 3 which are subclasses of the class Package. Then we represent each application of interest by its initial node, γ.

Given that the ontology graph is unweighted, i.e., all edges bear the same importance, we can traverse the graph using a Breadth-First Search (BFS) algorithm from its initial node γ to extract the minimum spanning tree (MST) to obtain the feature representation of the application. The BFS algorithm stops if any loop is found, a target node or a node with no outgoing connection is achieved, or if it exceeds a maximum user-defined depth for searching. During the traversal process, all BoG items (ω_t) found for each intermediate node (v_i) of the application's subgraph is represented by the corresponding feature value (δ_i), which is defined as the inverse of its distance (in a number of nodes) to the root of the MST:

$$D_\gamma = \quad \{\delta_1, ..., \delta_t, ..., \delta_n\},$$
$$\forall t \quad | \quad 1 \leq t \leq n = |\Omega|$$

$$\delta_t = \quad \frac{1}{distance(\gamma, v_i)}$$

where: δ_t is the value of the feature which represents the BoG item $\omega_t \in \Omega$ containing the intermediate node $v_i \in V_{onto}$ as present in the application subgraph with root node $\gamma \in V_{onto}$. Therefore, the vector elements representing BoG items that are not present in the subgraph are set to 0.

The ***inverse of the distance from the root node*** was chosen as the metric of feature relevance for the application. The closer to the root a node is, the higher the value of the feature, which means that fewer intermediates and more directly connected structures are present in the model. On the other hand, exploring nodes far from the root implies relationships that depend on many more elements inside the ecosystem.

The technique described above and exemplified by Figure 4 can be applied to convert any ontology graph into a set of feature vectors to describe subgraph relationships from nodes that belong to a set of classes. Particularly on this work, the ecosystem is the environment created with apps downloaded by users and manufacturer partner's apps inside an Android smartphone, representing the relationships described by AndroPermEco ontology, which maps only permissions, resources, and interface types.

3.3. Learning the Most Discriminant Features

To accelerate the following steps in the process, we eliminate the linearly dependent of the set of D_γ vectors, leaving only one feature to represent each set of linearly dependent BoGs. In the back modeling, after identifying the most discriminant features related to malware apps, we expand the BoGs correlated to the selected features. The resulting vectors after the elimination of dependent columns are feature vectors to input in the machine learning process.

$$F_\gamma = \{...,\delta_i,...\} \quad | \quad \delta_i \in D_\gamma \quad \text{and}$$
$$F_i \quad \text{is linearly independent of} \quad F_j \quad | \quad \forall F_i \in F, \quad \forall F_j \in F$$

For ranking features according to their relevance in the classification, an iterative process is performed to decrease the length of the vectors. This process also discards less important features and refines the rank for the remaining ones in the next iteration. On each iteration, a Random Forest classifier is trained, and features are weighted by their Feature Importance, as described in Section 2.4. Upon sorting the different importance values in descending order, we end up with a ranking of features.

The performance of the classifier is also measured, identifying how discarded features impacted the classification effectiveness. Looking at the metrics, it is possible to determine the length and, consequently, the set of features that produced the best performance. The metrics also provide the threshold point where performance starts to be more impacted by feature reduction, and a minimum length that supports the acceptance criteria where the iterations should be stopped. Obviously, if the minimum performance criteria are not violated by any length, the process will stop when the length of 1 is reached.

Typically, the minimum performance criteria are set to a minimum of 60% for recall (the prediction correctness for the real positive samples) and a minimum of 60% for precision (the prediction correctness for the predicted positive samples). This implies a 60% minimum of f1-score metric, which is used to determine the best performance length, and its estimated derivative is used to determine the threshold point.

At the end of the ranking process, the most discriminant features are determined by the top-ranked ones for the threshold length.

3.4. Back Modeling — Identifying Malware Features

After determining the most discriminant features, it is now necessary to identify the ones with the highest probability of having higher values in the malware apps compared to the benign ones. This means that the BoG represented by a given feature occupies a position nearer to the root in the malware MST graphs, as opposed to graphs for benign apps. This implies that this particular feature is more representative for the malware graph nodes. In particular, our method is based upon Rubin's propensity score[59] for the observational causal inference setup. This propensity score framework can be informally summarized as computing a score τ based on the outcomes of applying a treatment to a population ($r_{i,t=1}$), against the results of not applying the treatment

to a population ($r_{i,t=0}$), i.e.,

$$\tau = \sum_{i=1}^{N} r_{i,t=1} - r_{i,t=0}, \tag{5}$$

where r_i denotes the outcomes for individual i of the N individuals in the entire population studied. More generally, the framework can be used to analyze the counter-factuals between two given classes $\mathcal{C}_0$ and $\mathcal{C}1$ for a feature x_i, i.e.,

$$\tau i = \int \Big(p(x_i|\mathcal{C}_0) - p(x_i|\mathcal{C}_1) \Big)^2 \mathrm{d}x_i. \tag{6}$$

In the proposed method, we consider the classification output of a classification algorithm as a treatment to the population of Android apps considered, in terms of their features. By construction, features that are important for distinguishing malware from benign apps will exhibit a greater difference between their distributions $p(x_i|\mathcal{C}_0)$ (benign) and $p(x_i|\mathcal{C}_1)$ (malware). To ascertain whether such a difference is significant, we can employ simple statistical tests such as the two-sample Kolmogorov-Smirnov test (see, e.g., [60], [61]).

Given that the counterfactual distributions are unimodal and share the same support, we can find the transition point $\beta^\star$ for each feature x_i through optimization such that a given app is more likely to be benign than malware.

$$\beta^\star = \underset{\beta}{\operatorname{argmax}} \int_{-\infty}^{\beta} \Big(p(x_i|\mathcal{C}_0) - p(x_i|\mathcal{C}_1) \Big)^2 \mathrm{d}x_i. \tag{7}$$

In this way, we call ***malware features*** $H_m \in F$ the ones that present the highest probability of having higher values for malware apps over benign apps by meeting the following criteria:

$$H_m \in F \tag{8}$$

$$\beta^\star > 0 \quad \text{and} \quad \mu\big(p(x_m|\mathcal{C}_1)\big) > \mu\big(p(x_m|\mathcal{C}_0)\big) \tag{9}$$

Then, once a final classifier is trained and tested using only malware features to guarantee that such features retain enough information about the malware apps, we can proceed with the analysis. If the metrics are above 80%, we can conclude that malware features are representative of the BoG relationships of malware in the ecosystem graph.

3.5. Back Modeling — Mapping Malware Features to Ontology Graph Nodes

The next step is to identify BoGs that correspond to malware features. It is important to remember here that we eliminated the linearly dependent columns of vectors in D_γ. Then, it is time to recover the information about which BoGs are represented by each malware feature by looking for the elements of D_γ, which are highly correlated to those. The correlation matrix is calculated as follows:

$$\rho(D_i, D_j) = \frac{1}{f-1} \sum_{k=1}^{f} \frac{D_{k,i} - \mu_i}{\sigma_i} \frac{D_{k,j} - \mu_j}{\sigma_j}, \tag{10}$$

$$
\begin{aligned}
where: \quad & \mu_i = \text{mean of column i over all } f \text{ vectors in } D \\
& \sigma_i = \text{standard deviation of column i over all } f \text{ vectors in } D \\
& \mu_j = \text{mean of column j over all } f \text{ vectors in } D \\
& \sigma_j = \text{standard deviation of column j over all } f \text{ vectors in } D \\
& i = \{1,...,n\}, j = \{1,...,n\}
\end{aligned}
$$

$$
R = \begin{bmatrix} 1 & ... & \rho(D_1,D_j) & ... & \rho(D_1,D_n) \\ ... & ... & ... & ... & ... \\ \rho(D_n,D_1) & ... & \rho(D_n,D_j) & ... & 1 \end{bmatrix} \tag{11}
$$

Given that $H_m = D_k$, we consider that the malware feature H_m represents the BoGs D_j if the correlation index exceeds a threshold κ, i.e. $R_{k,j} \geq \kappa$. Particularly, we used $\kappa = 1$ in our experimental tests.

As BoGs are identified, it is possible to understand which permissions, interfaces, and resources appear more frequently on the relationships of malware apps inside the ecosystem. By the analysis of the prevalence of those features in each malware family, we can correlate the malware behavior and malicious activities with the resources used, showing that information extracted and filtered from the ecosystem can explain most of the malware activities.

4. Experiments and Results

In the following sections, we report on the experiments conducted to verify the method proposed in Section 3 with real-world data. In Section 4.1, we describe the metrics used to evaluate the performance of classifiers; in Section 4.2, we explain the Android ecosystem used on the experiments, which was transformed by the pre-processing method described in Sections 3.1 and 3.2 onto the features dataset. The full dataset was broken down into two partitions, one for the fitting process and another to the final test procedure. The fit partition was randomly divided into 10 subpartitions, which were combined in training and validation on the ratio of 80% to 20% for 10-round experiments. These led to the identification of the most discriminant malware features, which provided the base for the identification of malware groups, as well as the correlation among them.

All the steps of the method are automatically executed in a batch process. A Python program extracts manifests and certificates of all apps from a folder that contains the APK files. Then a parsing program, guided by the terminology description of the ontology and rules expressed in a config file, extracts from the XML manifests and certificates the ontology axioms, generating the N3 file that contains all ontology ecosystem sentences. Continuing the process using Python programming (it is easy to manipulate strings than in Matlab), we build the graph using Python dictionaries to resolve the symbolic (string names), building an indexed adjacency list of the nodes. The next step extracts the BoG tuples and the extraction of feature vectors for the applications using the MST traversal algorithm. The remaining processing of machine learning steps, to rank features and determine BoGs related to malware activity, are made using Matlab scripts.

4.1. Classifier Performance Metrics

The performance of binary classifiers is evaluated by the statistical metrics derived from a confusion matrix. The positive class in this article represents malware, while the negative class refers to benign apps. True Positives (TP) are correctly classified samples of malware, True Negatives (TN) are correctly classified samples of benign apps, False Positives (FP) are misclassified samples of the positive class, and False Negatives (FN) are misclassified examples of the negative class.

Table 2: Metrics used to calculate the performance of the classifiers. TPR is the acronym for True Positive Rate.

Metric	Recall (RCL) or TPR	Specificity (SPC)	Precision (PRC)	F1-score (F1S)	Accuracy (ACC)
Formula	$\frac{TP}{TP+FN}$	$\frac{TN}{TN+FP}$	$\frac{TP}{TP+FP)}$	$2 \cdot \frac{PRC \cdot RCL}{PRC+RCL}$	$\frac{(RCL+SPC)}{2}$

We use the normalized form to compute accuracy (ACC), averaging recall (RCL), and specificity (SPC) to avoid miscalculation if the datasets are unbalanced. We tried to keep the datasets as balanced as possible during the capture of samples, but it is not a requirement for the method.

4.2. Android Ecosystem Used On Experiments

An experimental ecosystem was built with a total of 4,570 apps, corresponding to 2,417 benign and 2,153 malware APKs. A Samsung Galaxy S5 smartphone with default factory installation was used, containing 181 APKs from Android v. 4.4 KitKat system (including the Android kernel app), and 117 partners' applications, which are APKs that come with the factory installation but are not signed by the same certificate as the operating system applications. To complete the benign set, 269 APKs with more than a million downloads were obtained directly from the [62], and 1,850 were randomly selected from AndroZoo repository [63].

For the malware set, we collected 193 APKs from the [64], from a torrent acquired from VirusShare [65] and AndroMalShare [66]; and 1,960 were gathered randomly selected from AndroZoo repository [63].

SHA256 hashes of all APK files were submitted to VirusTotal [67] to guarantee that benign apps are found in the repository with no infection reported; for malware, at least 10 anti-virus-reported detections were required.

The report produced by VirusTotal for the malware apps was submitted to the AVclass labeler program, publicly available at `https://github.com/malicialab/avclass`, and described by authors in the article [68]. The program classifies ***malware*** in families, according to the anti-virus proprietary codes.

Only malware from these families was included in the ecosystem, ensuring that we did not include apps classified as ***PUP*** (Potentially Unwanted Program). Although PUPs consume device resources by performing unethical tasks or annoy the user by installing toolbars, exhibiting continuous advertisements, and other unsolicited activities, they do not cause direct damage or incur costs to users, nor do they steal personal information.

The total of 4,570 APK files generated a list of 264,653 lines in N-Triples file format (.n3) [69]; these lines are divided into 102 for terminology and 264,551 for axioms, which were then transformed into a directed graph with

54,620 nodes and 119,624 edges (mean degree: 2.19, maximum degree: 2,751). For consistency and correctness, the N-Triples file was validated by importing it into Protege software [70].

A BoG with 8,950 items was extracted from the ontology graph, creating a feature matrix of 4,389 rows (2,236 benign, no system apps, and 2,153 malware) by 8,950 columns (the BoG items).

4.3. Datasets on Experiments

The complete dataset of 4,389 apps (vectors) was then randomly divided row-wise into a fit partition P_{fit} with 80% of samples (3,506) to be used in the training and validation of the learning process, and a test partition P_{test} with 20% of vectors (883), that was put apart for final testing.

For statistical verification of stability and robustness of classifiers, we randomly split the fit partition P_{fit} into 10 sub-partitions, which are combined into 10 datasets for each fitting round, that we called experiments. Each experiment uses 80% of P_{fit} for training (8 sub-partitions corresponding to 64% of all samples), and 20% of P_{fit} for validation test (2 sub-partitions corresponding to 16% of all vectors).

In the conducted experiments, it is important to observe that in the process of building partitions, we guarantee that benign and malware families are proportionally represented in the randomly selected samples.

4.4. Features Ranking and Most Discriminant Features Determination

First of all, we determine the set of linearly independent features using the vectors in the fit partition P_{fit}, discarding the dependent ones for further analysis. From the initial 8,950 vector columns (directly BoG-related), only 673 features remain as linearly independent and representatives for all BoG features.

Before proceeding with the ranking process, the P_{fit} partition is used to determine the number of trees for the Random Forest classifiers as a function of the vector length. In this case, 64 trees were determined for all lengths, by the grid-search results presented in Figure 5.

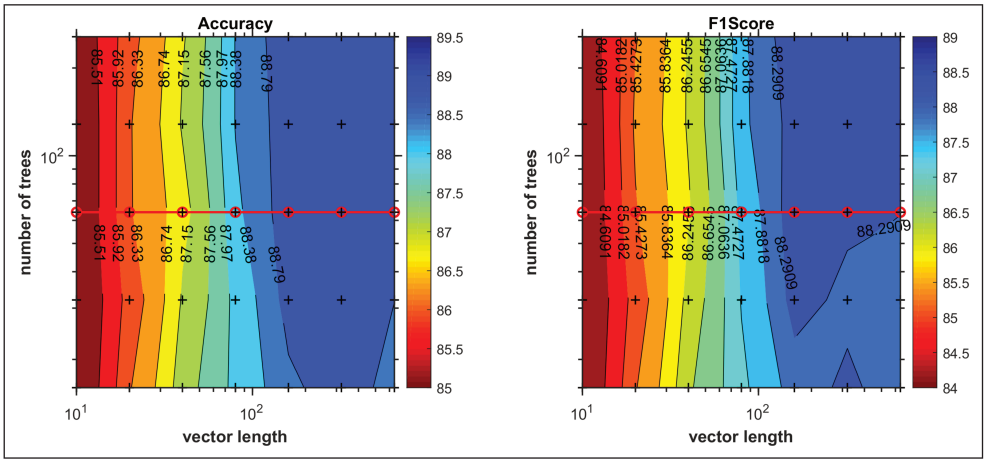

Figure 5: Grid search results for the number of trees determination. The red line indicates that, for the following experiments, 64 trees were chosen regardless of the vector length.

Then, Random Forest classifiers are trained and validated using the subpartitions derived from P_{fit} in 10 rounds for each step of decreasing vector length. In each step, the performance metrics of the 10-round classifiers

are summarized by the mean and standard deviation (80% training and 20% validation in each round). The ranking process consists of computing the variable importance of each trained classifier of the 10 rounds and averaging them into a feature importance score. The descendant sorts the score in each step and updates the rank of remaining features. After the process of 10 rounds for training, validation, and ranking of each step, we discard the ranked feature list tail (we used 10% reduction factor), repeating the process until a breaking criterion is achieved (or length comes to 0). We used a breaking criterion for the iterations if accuracy, recall or precision was less than 60%.

Performing the ranking process using the 3,506 vectors of P_{fit} with an initial length of 673, we found the maximum F1s (F1-score) at the length of 152 ranked features. However, we considered the length of 36 ranked features as the threshold point for most discriminant features. Figure 6 shows, on the left, the classification metrics during the reduction process and, on the right, the estimated derivative of F1s to determine the threshold point when the metric starts to decrease faster. Results decreased consistently until length 1, without violating the minimum criteria of 60% on recall and precision.

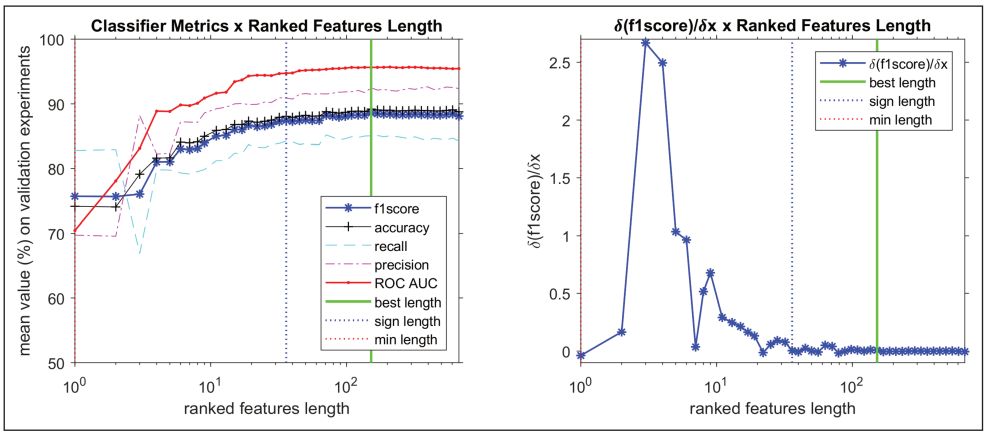

Figure 6: (left) Features ranking and minimum length determination. Best length (152) means length with maximum F1s, sign length (36) means length for a threshold on metrics and min length (1) means minimum length above minimum criteria. (right) An estimated derivative of F1s length search curve.

A descendant curve with the reduction of features indicates that the eliminated features from the ranking tail positively to the performance of classification. Table 3 displays the performance of classification metrics on the validation experiments, as well as the final test for the trained classifiers, all linearly independent, maximum F1s, and the most discriminant, determined in the ranking process. Table 3 also presents the performance metrics for the malware features, which will be explained in the next section.

The accuracy is above 88%, with a standard deviation below 1%. Precision and specificity are above 90%, with a standard deviation below 2%. Only recall presents lower values, but all above 84%, with a standard deviation below 2%. For those three vector lengths, results are statistically equivalent, as all of them match the overlapped standard deviation intervals. Those metrics are deemed high, considering the level of information captured on the application manifests, and given their statistical equivalence. We may conclude that the selected length for the most discriminant features has a representative set of variables that interfere with the classification process.

The low standard deviation on the metrics in the 10 rounds of validation experiments (with sub-partitions

Table 3: Random Forest classifiers performance metrics according to the features selected.

Classifier Metric	Features										
	All Linear Independent			Maximum F1-score		Most Discriminant			Malware		
	Validation (P_{fit})		Final Test (P_{test})	Validation (P_{fit})		Validation (P_{fit})		Final Test (P_{test})	Validation (P_{fit})		Final Test (P_{test})
	μ	σ	μ	μ	σ	μ	σ	μ	μ	σ	μ
Vector Length	673			152		36			24		
Number of Trees	64			64		64			64		
Accuracy (%)	88.8	0.7	90.8	89.2	0.7	88.0	0.9	89.9	85.0	1.5	86.4
Recall (%)	84.3	1.4	86.7	85.1	1.8	84.1	1.3	86.0	79.7	2.0	81.2
Specificity (%)	93.3	1.4	94.9	93.3	1.6	91.8	1.7	93.7	90.3	1.5	91.5
Precision (%)	92.3	1.4	94.3	92.5	1.6	90.9	1.7	93.1	88.7	1.7	90.3
F1-Score (%)	88.1	0.7	90.3	88.6	0.8	87.4	1.0	89.4	84.0	1.7	85.5

of P_{fit}) and the compatible results of the final tests with the separated partition P_{test} indicate that the training process produced robust classifiers with good generalization (reduced overfitting) of the ecosystem, capturing benign and malware families.

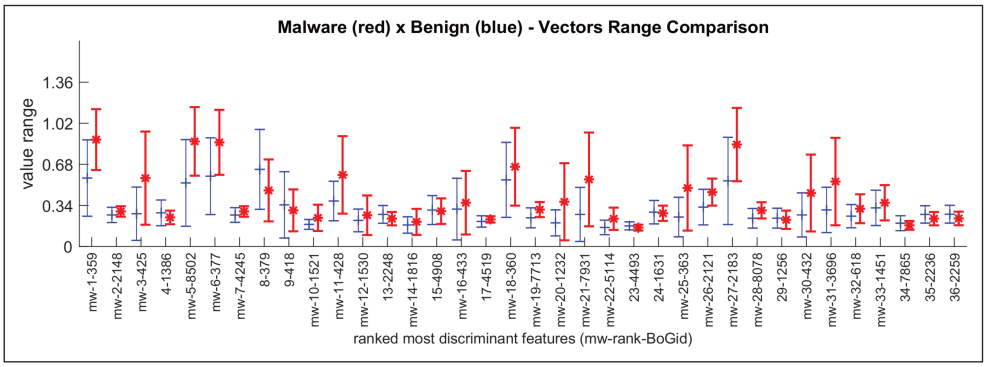

Figure 7: Feature value range comparison between malware and benign vectors for the most discriminant features. Malware features are labeled in the x-axis with the prefix ***mw-***.

4.5. Malware Features and Malware Families Relationship

After we had determined the most discriminant features, we performed the distribution analysis as explained in Section 3.4 to determine the ones that were malware class-related and called them ***malware feature***. A malware feature has a higher probability of presenting a higher value than a benign class vector, indicating that the BoG node represented by the variable is more likely to be nearer the application starting node in the ecosystem.

Using the vectors in the fit partition P_{fit} and performing the distribution analysis, we found 24 malware features; they are labeled in Figure 7 with the prefix ***mw-***. Then, we used only those malware features to train and validate the 10-round Random Forest classifiers with the P_{fit}. The final testing training was performed with the entire P_{fit} and the P_{test} as separated partitions. Although results shown in Table 3 for malware features are lower than other test results with same datasets, they can also be considered good, as accuracy is above 85%, specificity and precision is higher than 88%. The recall is approximately 80%, driving the f1-score to 84%. All measures have standard deviation below 2%, indicating good stability in overall round datasets. These results allow us to go forward in the analysis of the relationships between those features with malware families and behaviors.

Table 4: Malware features and correlated BoG Items.

Feature Rank	Feature Index	Correlated BoG-Id	BoG item				
			Node	Permission	Interface	Operation	Resource
1	16	359	android_permission_READ_PHONE_STATE	READ_PHONE_STATE@Signature	SysAPI	read	phone
2	1805	2148	android_permission_SET_WALLPAPER	SET_WALLPAPER@Normal	SysAPI	write	wallpaper
3	82	425	android_permission_GET_TASKS	GET_TASKS@Dangerous	IntentFilter	action	tasks
	82	426	android_permission_GET_TASKS	GET_TASKS@Dangerous	SysAPI	read	tasks
5	8159	8502	read_phone_state_sysapi167_read		SysAPI	read	phone
6	34	377	android_permission_ACCESS_WIFI_STATE	ACCESS_WIFI_STATE@Normal	SysAPI	read	wifi
	34	378	android_permission_ACCESS_WIFI_STATE	ACCESS_WIFI_STATE@Normal	SysAPI	write	wifi
	3902	4229	android_permission_SET_WALLPAPER_HINTS	SET_WALLPAPER_HINTS@Normal	SysAPI	write	wallpaper
7	3902	4245	android_permission_BIND_APPWIDGET	BIND_APPWIDGET@SignatureOrSystem	ExportedTrue	action	applications
	3902	4246	android_permission_BIND_APPWIDGET	BIND_APPWIDGET@SignatureOrSystem	SysAPI	write	applications
10	1178	1521	data_com_android_launcher2_LauncherProvider		Provider	read	laucher
	1178	1522	data_com_android_launcher2_LauncherProvider		Provider	write	laucher
	1178	2160	data_com_android_launcher2_LauncherProviderID		Provider	read	laucher
	1178	2161	data_com_android_launcher2_LauncherProviderID		Provider	write	laucher
	1178	7836	data_com_android_launcher2_LauncherProviderID		Provider	read	laucher
	1178	7837	data_com_android_launcher2_LauncherProviderID		Provider	read	laucher
	1178	8444	data_com_android_launcher2_LauncherProvider		Provider	read	laucher
	1178	8445	data_com_android_launcher2_LauncherProvider		Provider	read	laucher
11	85	428	android_permission_SYSTEM_ALERT_WINDOW	SYSTEM_ALERT_WINDOW@Dangerous	SysAPI	read	gui
	85	429	android_permission_SYSTEM_ALERT_WINDOW	SYSTEM_ALERT_WINDOW@Dangerous	SysAPI	write	gui
12	1187	1530	set_wallpaper_hints_sysapi223_write		SysAPI	write	wallpaper
14	1473	1816	set_debug_app_sysapi20_write		SysAPI	write	applications
16	90	433	android_permission_RECORD_AUDIO	RECORD_AUDIO@Normal	SysAPI	write	audio
18	17	360	android_permission_ACCESS_COARSE_LOCATION	ACCESS_COARSE_LOCATION@Dangerous	IntentFilter	action	location
	17	361	android_permission_ACCESS_COARSE_LOCATION	ACCESS_COARSE_LOCATION@Dangerous	SysAPI	read	location
	17	362	android_permission_ACCESS_COARSE_LOCATION	ACCESS_COARSE_LOCATION@Dangerous	SysAPI	write	location
19	7370	7713	bind_call_service_sysapi159_write		SysAPI	write	phone
	7370	7714	call_phone_sysapi160_read		SysAPI	read	phone
	7370	7715	call_phone_sysapi160_write		SysAPI	write	phone
	7370	7716	call_privileged_sysapi161_read		SysAPI	read	phone
	7370	7717	call_privileged_sysapi161_write		SysAPI	write	phone
	7370	7718	invoke_carrier_setup_sysapi162_write		SysAPI	write	phone
	7370	7719	modify_phone_state_sysapi163_write		SysAPI	write	phone
	7370	7720	process_outgoing_calls_sysapi164_read		SysAPI	read	phone
	7370	7721	process_outgoing_calls_sysapi164_write		SysAPI	write	phone
	7370	7722	read_call_log_sysapi165_read		SysAPI	read	phone
	7370	7723	read_logs_sysapi166_read		SysAPI	read	phone
	7370	7724	read_phone_state_sysapi167_read		SysAPI	read	phone
	7370	7725	read_privileged_phone_state_sysapi168_read		SysAPI	read	phone
	7370	7726	write_apn_settings_sysapi169_write		SysAPI	write	settings
	7370	7727	write_call_log_sysapi170_write		SysAPI	write	phone
20	889	1232	mount_unmount_filesystems_sysapi89_read		SysAPI	read	filesystems
	889	1233	mount_unmount_filesystems_sysapi89_write		SysAPI	write	filesystems
	7588	7930	android_intent_action_MAIN_ifilt1918		IntentFilter	action	applications
21	7588	7931	get_tasks_sysapi209_read		SysAPI	read	tasks
22	4771	5114	get_detailed_tasks_sysapi208_read		SysAPI	read	tasks
	4771	5115	get_tasks_sysapi209_read		SysAPI	read	tasks
	4771	5116	remove_tasks_sysapi210_read		SysAPI	read	tasks
	4771	5117	remove_tasks_sysapi210_write		SysAPI	write	tasks
	4771	5118	reorder_tasks_sysapi211_read		SysAPI	read	tasks
	4771	5119	reorder_tasks_sysapi211_write		SysAPI	write	tasks
25	20	363	android_permission_SEND_SMS	SEND_SMS@Dangerous	IntentFilter	action	sms_mms
26	1778	2121	resource_phone		SysAPI	read	phone
	1778	2122	resource_phone		SysAPI	read	phone
	1778	2123	resource_phone		SysAPI	read	phone
27	1840	2183	access_wifi_state_sysapi224_read		SysAPI	read	wifi
	1840	2184	access_wifi_state_sysapi224_write		SysAPI	write	wifi
28	7735	8078	access_wifi_state_sysapi224_read		SysAPI	read	wifi
30	89	432	android_permission_WRITE_SETTINGS	WRITE_SETTINGS@Normal	SysAPI	write	settings
31	3353	3696	system_alert_window_sysapi111_read		SysAPI	read	gui
	3353	3697	system_alert_window_sysapi111_write		SysAPI	write	gui
32	275	618	resource_phone		SysAPI	write	phone
	275	619	resource_phone		SysAPI	write	phone
	275	620	resource_phone		SysAPI	read	phone
33	1108	1451	resource_location		SysAPI	read	location
	1108	1452	resource_location		SysAPI	read	location
	1108	1453	resource_location		SysAPI	read	location
	1108	2490	resource_location		SysAPI	write	location
	1108	2491	resource_location		SysAPI	read	location
	1108	2492	resource_location		SysAPI	write	location

It is important to remember that the selected malware features are the most discriminant linearly-independent features that have higher values for the malware class. In this way, to find the nodes represented by these malware features in the ecosystem ontology graph (elements of the BoG vectors), we need to identify the related BoG (Section 3.5), which means to expand the correlated elements of the vector. Then, computing

the correlation matrix of the BoG vectors and selecting BoG elements with correlation index equal to one, we expanded the 24 malware features into 70 BoG elements described in Table 4.

Table 5: Classification results for AVclass malware families. $T*R$ means TNR (true negative rate) for benign and TPR(True Positive Rate) for malware apps. Family names with † indicate that there were less than 10 samples available in the ecosystem.

	Number of vector (Samples)			Features					
				All LI		Most Discriminant		Malware	
	Validation	Final Test	Total	Validation	Final Test	Validation	Final Test	Validation	Final Test
Vector Length				**673**		**36**		**24**	
Family	P_{fit}	P_{test}	P_{all}	$T*R$ (%)	$T*R$ (%)	$T*R$ (%)	$T*R$ (%)	$T*R$ (%)	$T*R$ (%)
benign	1789	447	2236	94.6	94.9	93.3	93.7	91.3	91.5
plankton	184	46	230	76.1	78.3	74.5	76.1	73.4	71.7
mecor	174	44	218	100.0	100.0	100.0	100.0	100.0	100.0
leadbolt	97	24	121	74.2	66.7	76.3	66.7	61.9	54.2
fakeapp	86	21	107	89.5	95.2	90.7	95.2	73.3	76.2
kuguo	82	21	103	97.6	100.0	98.8	95.2	96.3	95.2
umpay	79	20	99	100.0	95.0	98.7	100.0	96.2	100.0
secapk	78	20	98	97.4	100.0	96.2	100.0	94.9	100.0
smsreg	62	15	77	88.7	100.0	88.7	100.0	85.5	93.3
ramnit	58	15	73	50.0	66.7	44.8	66.7	44.8	53.3
smspay	57	14	71	100.0	100.0	100.0	100.0	98.2	100.0
fobus	55	14	69	67.3	57.1	65.5	42.9	40.0	28.6
fakeinst	42	11	53	95.2	90.9	92.9	90.9	92.9	90.9
dowgin	40	10	50	97.5	100.0	97.5	100.0	97.5	100.0
jiagu	39	10	49	92.3	100.0	89.7	100.0	89.7	100.0
tencentprotect	34	9	43	100.0	100.0	100.0	100.0	100.0	100.0
wateh	34	9	43	100.0	100.0	100.0	100.0	100.0	100.0
porno	32	8	40	6.3	37.5	6.3	37.5	6.3	37.5
youmi	31	8	39	93.5	87.5	90.3	100.0	93.5	87.5
viser	27	7	34	63.0	71.4	63.0	71.4	55.6	71.4
smforw	26	6	32	100.0	100.0	96.2	100.0	92.3	100.0
opfake	26	6	32	96.2	100.0	96.2	100.0	96.2	100.0
igexin	26	6	32	96.2	100.0	96.2	100.0	92.3	100.0
torjok	22	6	28	100.0	100.0	100.0	100.0	95.5	100.0
redirector	22	5	27	86.4	100.0	86.4	100.0	86.4	100.0
rootnik	20	5	25	100.0	100.0	100.0	100.0	100.0	100.0
ginmaster	20	5	25	90.0	100.0	90.0	100.0	95.0	80.0
inmobi	16	4	20	81.3	100.0	81.3	100.0	56.3	75.0
gappusin	15	4	19	100.0	100.0	100.0	100.0	100.0	100.0
virut	15	4	19	0.0	0.0	0.0	0.0	0.0	0.0
clickfraud	14	4	18	7.1	25.0	7.1	25.0	7.1	25.0
poogle	14	3	17	100.0	100.0	100.0	100.0	100.0	100.0
drosel	13	3	16	100.0	100.0	100.0	100.0	100.0	100.0
zhui	12	3	15	100.0	100.0	100.0	100.0	100.0	100.0
buzztouch	10	3	13	70.0	100.0	70.0	100.0	40.0	33.3
adcolony	10	3	13	40.0	33.3	40.0	33.3	20.0	33.3
inor	10	3	13	30.0	66.7	30.0	66.7	0.0	0.0
tekwon	10	2	12	100.0	100.0	100.0	100.0	100.0	100.0
wroba	10	2	12	100.0	100.0	100.0	100.0	100.0	100.0
igamo	10	2	12	100.0	100.0	100.0	100.0	90.0	100.0
mulad	9	2	11	88.9	100.0	88.9	50.0	77.8	100.0
finspy	8	2	10	100.0	100.0	100.0	100.0	100.0	100.0
deng	8	2	10	62.5	50.0	75.0	50.0	75.0	50.0
nandrobox†	7	2	9	100.0	100.0	100.0	100.0	100.0	100.0
mobidash†	7	2	9	100.0	50.0	100.0	50.0	57.1	0.0
andup†	6	2	8	100.0	100.0	100.0	100.0	100.0	100.0
dianjin†	6	2	8	100.0	50.0	100.0	50.0	100.0	50.0
anydown†	6	2	8	100.0	0.0	100.0	0.0	83.3	0.0
lotoor†	6	1	7	0.0	0.0	16.7	0.0	16.7	0.0
faketoken†	5	1	6	100.0	100.0	100.0	100.0	100.0	100.0
androrat†	5	1	6	100.0	0.0	100.0	100.0	100.0	100.0
gidix†	4	1	5	100.0	100.0	100.0	100.0	100.0	100.0
smsagent†	4	1	5	100.0	100.0	100.0	100.0	100.0	100.0
marcher†	4	1	5	75.0	100.0	100.0	100.0	100.0	100.0
zitmo†	3	1	4	100.0	100.0	100.0	100.0	100.0	100.0
smsspy†	3	1	4	100.0	100.0	100.0	100.0	66.7	100.0
gepew†	2	1	3	100.0	100.0	100.0	100.0	100.0	100.0
golddream†	2	1	3	100.0	100.0	100.0	100.0	100.0	100.0
samsapo†	2	1	3	100.0	100.0	100.0	100.0	100.0	100.0
recal†	2	1	3	100.0	100.0	100.0	0.0	100.0	100.0
koler†	2	1	3	100.0	100.0	100.0	100.0	0.0	0.0
mobilespy†	2	1	3	50.0	100.0	50.0	100.0	50.0	100.0
basebridge†	2	1	3	50.0	0.0	50.0	0.0	50.0	0.0
Total	**3506**	**883**	**4389**	**90.1**	**90.8**	**89.3**	**89.9**	**86.0**	**86.4**
Total Benign	**1789**	**447**	**2236**	**94.6**	**94.9**	**93.3**	**93.7**	**91.3**	**91.5**
Total Malware	**1717**	**436**	**2153**	**85.4**	**86.7**	**85.0**	**86.0**	**80.5**	**81.2**

The identification of BoG elements corresponding to the ranked malware features allows us to identify which resources, interfaces, and permissions are represented by them, guiding us on the back modeling of those nodes and relationships into the ecosystem graph. It also allows us to evaluate the relationships between each malware family (classification of the APKs related to the vectors), according to the AVclass labeler program [68], which corresponds to the malware behavior and attack goal.

In this way, we first evaluate the TPR(True Positive Rate or Recall) of each family to assess whether the family was correctly identified by the classifiers; this is presented in Table 5. As we can see, results are consistent across all families on the experiments, with all linearly independent (673), most discriminant (36), and malware features (vector length of 24).

A summary of results in tests with the most discriminant features is as follows: 58% of families achieved 100%; 17% less than 100%, but above 80%; 7% less than 80%, but above 60%; and only 18% below 60%. Malware features performed at 49%, 19%, 6%, and 25%, respectively. The high number of families with TPR above 80%, 75% on most discriminant tests, and 68% on malware features tests shows that most families were correctly identified by the selected features.

Families with poor results in the most discriminant features (marked in red in tables) also show poor results in other tests, indicating that classifiers could not capture their features in training. This means that the selected features cannot be used to model their malware behavior. Although most results for families with few samples (less than 10 in total) are excellent (most are 100% of correctness), we cannot consider those families for further resources and relationships analysis, as they do not have enough data. We marked them with † after the family name in Tables 5 and 6.

Benign applications were well recognized by the classifiers in all tests, presenting TNR (True Negative Rate) above 91% in all tests.

It is important to evaluate the resources filtered in the most discriminant features with known malware behavior and attack goals in order to validate that the BoG graph retains this information inside relationships. In this way, Table 6 presents the use of the resources associated with the 24 malware features by the malware families. A checkmark in Table 6 indicates that vectors of the family inside the P_{fit} partition have a mean value above the mean value of benign applications on the resource related most discriminant features, which means that resource is more requested by the malware family apps than in the benign apps.

Table 6 shows that most malware families access resources such as ***phone, settings, wifi, applications, sms_mms, filesystems and tasks***. Almost 90% of all malware samples, and above 80% of malware families use phone and settings. Binding another application is also used by 79% of malware samples and families. ***Wifi, sms_mms, filesystems*** and ***tasks controls*** are used by 71% of families and by the same percentage of samples.

Conversely, eight out of ten malware families marked with poor classification performance have few checkmarks, which indicates that they do not use the selected malware features as much; this is a clear indication of why they were confused with benign applications by the classifiers. The only exceptions to the previous sentence are the families ***mulad*** and ***deng***, which use almost all of the features but were not detected: this means that

they do not fit into the combinations selected by the Random Forest rules for malware detection in the ecosystem.

Table 6: AVclass malware family resource usage compared to that of the benign apps. A checkmark (✓) indicates that the mean values of features related to the resource for the malware family vectors are higher than the mean values of the same features in the benign apps. Family names with † indicate that there were less than 10 samples available in the ecosystem.

Malware	**Number of**	**Resource**											
Family	**Samples**	**phone**	**settings**	**wifi**	**applications**	**sms_mms**	**filesystems**	**tasks**	**launcher**	**location**	**wallpaper**	**gui**	**audio**
plankton	230	✓	✓	✓	✓	✓	✓	✓	✓		✓		
mecor	218	✓	✓	✓	✓	✓				✓	✓		
leadbolt	121	✓	✓	✓	✓		✓	✓					
fakeapp	107	✓	✓	✓						✓			
kuguo	103	✓	✓	✓	✓	✓	✓	✓	✓			✓	
umpay	99	✓	✓	✓	✓	✓	✓	✓	✓				✓
secapk	98	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
smsreg	77	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ramnit	73			✓									
smspay	71	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
fobus	69	✓	✓	✓						✓			✓
fakeinst	53	✓	✓		✓	✓	✓	✓			✓		
dowgin	50	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
jiagu	49	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
tencentprotect	43	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
wateh	43	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
porno	40												
youmi	39	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
viser	34	✓	✓	✓									
smforw	32	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
opfake	32	✓	✓		✓	✓	✓	✓	✓		✓	✓	
igexin	32	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
torjok	28	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
redirector	27	✓	✓		✓								✓
rootnik	25	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
ginmaster	25	✓	✓	✓	✓		✓	✓	✓	✓		✓	
inmobi	20			✓						✓		✓	
gappusin	19	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓
virut	19												
clickfraud	18												
poogle	17	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
drosel	16	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
zhui	15	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
buzztouch	13	✓	✓							✓		✓	
adcolony	13			✓						✓			
inor	13												
tekwon	12	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓
wroba	12	✓	✓		✓	✓	✓	✓	✓		✓		✓
igamo	12	✓	✓	✓	✓	✓	✓	✓				✓	
mulad	11	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
finspy	10	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓
deng	10	✓	✓	✓	✓	✓	✓	✓	✓				
nandrobox†	9	✓	✓	✓	✓	✓	✓	✓					
mobidash†	9	✓	✓		✓					✓			✓
andup†	8	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
dianjin†	8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓
anydown†	8	✓	✓	✓									
lotoor†	7												
faketoken†	6	✓	✓		✓	✓	✓	✓	✓		✓	✓	✓
androrat†	6	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
gidix†	5	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
smsagent†	5	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
marcher†	5	✓	✓		✓	✓	✓	✓	✓		✓		✓
zitmo†	4	✓	✓		✓	✓	✓	✓	✓		✓	✓	
smsspy†	4				✓		✓	✓					
gepew†	3	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
golddream†	3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
samsapo†	3	✓	✓		✓	✓	✓	✓	✓		✓	✓	✓
recal†	3	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓
koler†	3	✓	✓		✓	✓	✓	✓					
mobilespy†	3	✓		✓	✓	✓					✓		✓
basebridge†	3	✓		✓	✓	✓					✓		
Families usage	**62**	**85.5%**	**82.3%**	**71.0%**	**79.0%**	**71.0%**	**71.0%**	**71.0%**	**58.1%**	**45.2%**	**50.0%**	**50.0%**	**51.6%**
Samples Usage	**2153**	**90.4%**	**90.1%**	**87.3%**	**79.8%**	**71.2%**	**67.8%**	**67.8%**	**57.8%**	**51.7%**	**48.9%**	**41.0%**	**37.1%**

The non-detected families ***virut, porno***, and ***clickfraud*** can easily be explained by the fact that they are not related to the infected app's behavior but to the HTML web pages contained or accessed by the application. For example, virut is a botnet spread through HTML files that are infected with virus code [71].

Another good example that explains why a family is not detected by the malware features is ***ramnit***. This is not Android malware, but its purpose is to steal banking login data from Windows operating system users [72]. It is carried inside installed Android app files and started when the Android device connects to a Windows PC and accesses an infected HTML web page. Then, neither malicious behavior nor access to Android resources that are different from benign applications is needed.

As we can see from the results in this article, using just a combination of 12 resources (out of 55 in the ecosystem definition) is enough to detect many malware families in the Android environment. Most of them are related to ***phone*** states and call logs (as we can see from the BoGs list in Table 4), ***settings*** access, ***wifi*** states, and ***SMS*** messages. Those resources are compatible with most malware and malicious activities that depend on external communications to send data and use phone calls as well as SMS messages for charged services.

4.6. Computing Performance Metrics

All experiments described herein and time measured in Table 7 were performed using a Samsung NP500R5H-XD3BR with Intel Core i7-5500CPU, 2.4 GHz, 2 Cores, 4 logical processors, 8 GB of physical memory, and 1 TB HD 5400RPM SATA-III 6GB/s. Programs of steps 1 to 4 were written in Python 2.7.10, which used 97% of the entire elapsed time. Steps 5 to 9 run in MATLAB R2018a 64-bit version 9.4.0.813654 (23-Feb-2018). All process steps from manifests and certificates extraction of APK files to the end results took approximately 22.5 hours. The most time-consuming step was the feature vector generation from the ontology graph that took almost 20 hours, where the BoGs of each application were extracted from the entire ecosystem graph. Machine learning steps consume 39 minutes, mostly in the 10-round iterations of the number of trees grid search and the ranking process. The time for training the model with or checking (predict) each application vector using the most discriminant feature (36 features) is less than a millisecond.

Table 7: Processing time for the experiments described in this article.

Step #	Process step	Total elapsed (hh:mm:ss)	# of Items processed	Item	Time per item	Time unit
1	**Manifest and certificates extration from APK files**	01:11:54	4570	APK file	944	ms
2	**Certificate and XML parsing to ontology N3 file**	00:01:57	4570	APK file	25.7	ms
3	**Build ontology graph from N3 file**	00:40:21	264653	N3 line	9.1	ms
4	**Build feature vector (ecosystem graph MST traversal for each app)**	19:45:36	4389	Vector	469	sec
5	**Data preparation (linear independent features determination and partitioning)**	00:06:31	4389	Vector	89.1	ms
6	**Grid search for number of trees**	00:18:27	35	Point	31.6	sec
7	**Ranking and most discriminant features determination**	00:12:52	46	Iteration	16.8	sec
8	**Final Classifier Training (P_{fit}= 3506 vectors) and Final Test (P_{test} = 883 vectors) - vector length 36 - 64 trees.**	00:00:05	4389	Vector	1.1	ms
	Training time (ensemble RF fitting)	0.975 sec	3506	Vector	278	μs
	Test time (ensemble RF predict)	0.074 sec	883	Vector	83	μs
9	**Feature values distribution analysis**	00:00:27	3506	Vector	7.7	ms
	End to end process elapsed time	**22:18:11**	**4570**	**APK file**	**7.5**	**min**

5. Conclusion and Future Work

In this paper, we have introduced two new methods to address the problem of mapping the relationships and characteristics of malicious software in smartphones. We provided an extensible framework for mapping the analyzed elements in the Android system using ontologies, as well as a random forest-based method for automatically extracting meaningful information from the ontological map obtained from the new mapping algorithm. Experimental results in the considered Android ecosystem showed that the proposed methodology was capable of detecting 36 key features, 24 of which are malware-related, corresponding to 70 graph nodes relative to malware activities, a remarkable result for security. Those nodes represent relationships between malware and 12 systems resources that are related to their behavior and malicious purpose. Equally important, all of this was made possible without the need to directly access malware code samples, nor existing signatures. Everything was analyzed directly by looking at how each app interacts with system permissions.

The proposed method has shown significant potential for modeling relationships between properties of components in an ecosystem, identifying structural elements that can characterize their behavior. The model used in the experiments showcases this power by providing good results in the classification of malware and benign apps. The information used for discrimination was obtained solely from the manifest of apps, which contains high-level information about the use of permissions and are publicly available.

One of the most important characteristics of our model underpinned by ontologies relies on its incremental nature, allowing us to improve the ecosystem's model as more knowledge is acquired and more apps are analyzed, without the need to start from scratch. If data collected from a new application can be described only by existing classes through the existing bag of words, the inclusion into the graph and use of the already trained classifier is trivial. As more samples of malware and benign apps are aggregated to the graph, a new round of training can be done to refine the prediction and identification of the most important features. This methodology provides a way to evolve the model as more knowledge is acquired and also to adapt as malware apps change their attack strategies over time.

This work also opens three novel research avenues. First, it is important to analyze whether other information that can be easily obtained from the Android system significantly improves the mapping and classification algorithms derived herein. Second, an empirical study can be conducted to assess how different pairs of mapping and analysis algorithms perform compared to our study. Finally, the proposed methodology can be applied to different operating systems and ecosystems, and their investigation would undoubtedly be important future work.

Acknowledgment

We thank the financial support of Intel Strategic Research Alliance (Grant #440850/2013-4), the National Council for Scientific and Technological Development – CNPq (Grant #302224/2015-7), the São Paulo Research Foundation – Fapesp (DéjàVu Grant #2017/12646-3), and the Coordination for the Improvement of Higher Education Personnel – Capes (DeepEyes grant), as well as Cambridge Trusts-CAPES grant BEX 9407-11-1.

References

[1] S. Carson, A. Furuskr, P. Jonsson, J. Kronander, P. Lindberg, R. Ludwig, K. hman, J. S. Sehti, Ericson Mobility Report, Ericsson AB (11 2016).
URL https://tinyurl.com/jeet6pg

[2] L. Sui, Global smartphone os market share by region: Q3 2016, Tech. rep., StrategyAnalytics (10 2016).
URL http://tinyurl.com/j6nospu

[3] P. Lee, E. Talbot, There's no place like phone - global mobile consumer survey 2016: Uk cut, Tech. rep., Deloitte LLP (2016).
URL https://tinyurl.com/ydat32u8

[4] H. Lee, Y. Zhang, K. L. Chen, An investigation of features and security in mobile banking strategy, Journal of International Technology and Information Management 22 (4) (2013) 23–46.
URL http://scholarworks.lib.csusb.edu/jitim/vol22/iss4/2/

[5] A. F. A. Kadir, N. Stakhanova, A. A. Ghorbani, An Empirical Analysis of Android Banking Malware, CRC Press, 2016, Ch. 9, pp. 209–232.

[6] S. Bojjagani, V. N. Sastry, Stamba: Security testing for android mobile banking apps, in: S. M. Thampi, S. Bandyopadhyay, S. Krishnan, K.-C. Li, S. Mosin, M. Ma (Eds.), Proceedings of Second International Symposium on Signal Processing and Intelligent Recognition Systems (SIRS-2015), Springer International Publishing, 2016, pp. 671–683. doi:10.1007/978-3-319-28658-7_57.
URL http://dx.doi.org/10.1007/978-3-319-28658-7_57

[7] R. Chanajitt, W. Viriyasitavat, K.-K. R. Choo, Forensic analysis and security assessment of android m-banking apps, Australian Journal of Forensic Sciences 0 (0) (2016) 1–17. doi:10.1080/00450618.2016.1182589.
URL http://dx.doi.org/10.1080/00450618.2016.1182589

[8] W. Enck, M. Ongtang, P. McDaniel, Understanding android security, IEEE Security Privacy 7 (1) (2009) 50–57. doi:10.1109/MSP.2009.26.
URL http://dx.doi.org/10.1109/MSP.2009.26

[9] P. G. Kelley, S. Consolvo, L. F. Cranor, J. Jung, N. Sadeh, D. Wetherall, A conundrum of permissions: Installing applications on an android smartphone, in: J. Blyth, S. Dietrich, L. J. Camp (Eds.), Financial Cryptography and Data Security: FC 2012 Workshops, USEC and WECSR 2012, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 68–79. doi:10.1007/978-3-642-34638-5_6.
URL http://dx.doi.org/10.1007/978-3-642-34638-5_6

[10] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, D. Wagner, Android permissions: User attention, comprehension, and behavior, in: Proceedings of the Eighth Symposium on Usable Privacy and Security, SOUPS '12, ACM, New York, NY, USA, 2012, pp. 3:1–3:14. doi:10.1145/2335356.2335360.
URL http://dx.doi.org/10.1145/2335356.2335360

[11] A. P. Felt, E. Chin, S. Hanna, D. Song, D. Wagner, Android permissions demystified, in: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS '11, ACM, New York, NY, USA, 2011, pp. 627–638. doi:10.1145/2046707.2046779.
URL http://dx.doi.org/10.1145/2046707.2046779

[12] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe, D. Wagner, How to ask for permission, in: Proceedings of the 7th USENIX Conference on Hot Topics in Security, HotSec'12, USENIX Association, Berkeley, CA, USA, 2012, pp. 7–7.
URL https://tinyurl.com/yd5nx3mt

[13] K. W. Y. Au, Y. F. Zhou, Z. Huang, D. Lie, Pscout: Analyzing the android permission specification, in: Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS '12, ACM, New York, NY, USA, 2012, pp. 217–228. doi:10.1145/2382196.2382222.
URL http://dx.doi.org/10.1145/2382196.2382222

[14] K. O. andMichelle Atkinson, Apps permissions in the google play store, Tech. rep., PewResearchCenter (11 2015).
URL https://tinyurl.com/o9nn2nh

[15] Y. Wang, Y. Alshboul, Mobile security testing approaches and challenges, in: 2015 First Conference on Mobile and Secure Services (MOBISECSERV), 2015, pp. 1–5. doi:10.1109/MOBISECSERV.2015.7072880.
URL http://dx.doi.org/10.1109/MOBISECSERV.2015.7072880

[16] G. Inc., the Open Handset Alliance, Android development api guide (2017).
URL https://developer.android.com/guide/index.html

[17] N. Elenkov, Android Security Internals: An In-Depth Guide to Android's Security Architecture, 1st Edition, No Starch Press, San Francisco, CA, USA, 2014.

[18] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, M. Rajarajan, Android security: A survey of issues, malware penetration and defenses, IEEE Communications Surveys Tutorials 17 (2) (2015) 998–1022. doi:10.1109/COMST.2014.2386139.
URL http://dx.doi.org/10.1109/COMST.2014.2386139

[19] X. Zheng, L. Pan, E. Yilmaz, Security analysis of modern mission critical android mobile applications, in: Proceedings of the Australasian Computer Science Week Multiconference, ACSW '17, ACM, New York, NY, USA, 2017, pp. 2:1–2:9. doi:10.1145/3014812.3014814.
URL http://dx.doi.org/10.1145/3014812.3014814

[20] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, A. N. Sheth, Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones, in: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10, USENIX Association, Berkeley, CA, USA, 2010, pp. 393–407.
URL https://tinyurl.com/ycljteez

[21] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, G. Álvarez, Puma: Permission usage to detect malware in android, in: Á. Herrero, V. Snášel, A. Abraham, I. Zelinka, B. Baruque, H. Quintián, J. L. Calvo, J. Sedano, E. Corchado (Eds.), International Joint Conference CISIS'12-ICEUTE12-SOCO12 Special Sessions, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 289–298. doi:10.1007/978-3-642-33018-6_30.
URL http://dx.doi.org/10.1007/978-3-642-33018-6_30"

[22] S. Das, Y. Liu, W. Zhang, M. Chandramohan, Semantics-based online malware detection: Towards efficient real-time protection against malware, IEEE Transactions on Information Forensics and Security 11 (2) (2016) 289–302. doi:10.1109/TIFS.2015.2491300.
URL http://dx.doi.org/10.1109/TIFS.2015.2491300

[23] S. K. Muttoo, S. Badhani, Android malware detection: state of the art, International Journal of Information Technology 9 (1) (2017) 111–117. doi:10.1007/s41870-017-0010-2.
URL https://doi.org/10.1007/s41870-017-0010-2

[24] A. Saracino, D. Sgandurra, G. Dini, F. Martinelli, Madam: Effective and efficient behavior-based android malware detection and prevention, IEEE Transactions on Dependable and Secure Computing 15 (1) (2018) 83–97. doi:10.1109/TDSC.2016.2536605.
URL https://doi.org/10.1109/TDSC.2016.2536605

[25] M. Zhang, Y. Duan, H. Yin, Z. Zhao, Semantics-aware android malware classification using weighted contextual api dependency graphs, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, ACM, New York, NY, USA, 2014, pp. 1105–1116. doi:10.1145/2660267.2660359.
URL https://doi.org/10.1145/2660267.2660359

[26] K. A. Talha, D. I. Alper, C. Aydin, Apk auditor: Permission-based android malware detection system, Digital Investigation 13 (2015) 1–14. doi:10.1016/j.diin.2015.01.001.
URL https://doi.org/10.1016/j.diin.2015.01.001

[27] T. Gruber, Ontology, in: Encyclopedia of Database Systems, Springer-Verlag, US, 2009, p. 0.

[28] N. Guarino, D. Oberle, S. Staab, What is an ontology?, in: Handbook on Ontologies, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 1–17.

[29] R. Studer, V. R. Benjamins, D. Fensel, Knowledge engineering: Principles and methods, Data Knowl. Eng. 25 (1-2) (1998) 161–197. doi:10.1016/S0169-023X(97)00056-6.
URL https://doi.org/10.1016/S0169-023X(97)00056-6

[30] M. Krallinger, F. Leitner, M. Vazquez, D. Salgado, C. Marcelle, M. Tyers, A. Valencia, A. Chatraryamontri, How to link ontologies and proteinprotein interactions to literature: text-mining approaches and the biocreative experience, Database (Oxford) 2012 (2012) bas017. doi:10.1093/database/bas017.
URL http://dx.doi.org/10.1093/database/bas017

[31] K. Eilbeck, S. E. Lewis, C. J. Mungall, M. Yandell, L. Stein, R. Durbin, M. Ashburner, The sequence ontology: a tool for the unification of genome annotations, Genome Biology 6 (5) (2005) R44. doi:10.1186/gb-2005-6-5-r44.
URL http://dx.doi.org/10.1186/gb-2005-6-5-r44

[32] B. Smith, M. Ashburner, C. Rosse, J. Bard, W. Bug, W. Ceusters, L. J. Goldberg, K. Eilbeck, A. Ireland, C. J. Mungall, The OBI Consortium, N. Leontis, P. Rocca-Serra, A. Ruttenberg, S.-A. Sansone, R. H. Scheuermann, N. Shah, P. L. Whetzel, S. Lewis, The obo foundry: coordinated evolution of ontologies to support biomedical data integration, Nature Biotechnology 25 (2007) 1251–1255. doi:10.1038/nbt1346.
URL http://dx.doi.org/10.1038/nbt1346

[33] R. Navigli, P. Velardi, Learning domain ontologies from document warehouses and dedicated web sites, Computational Linguistics 30 (2) (2004) 151–179. doi:10.1162/089120104323093276.
URL http://dx.doi.org/10.1162/089120104323093276

[34] A. Upward, P. Jones, An ontology for strongly sustainable business models, Organization & Environment 29 (1) (2016) 97–123. doi:10.1177/1086026615592933.
URL http://dx.doi.org/10.1177/1086026615592933

[35] I. Horrocks, Ontologies and the semantic web, Commun. ACM 51 (12) (2008) 58–67. doi:10.1145/1409360.1409377.
URL http://dx.doi.org/10.1145/1409360.1409377

[36] V. Singh, S. Pandey, Revisiting security ontologies, International Journal of Computer Science Issues (IJCSI) 11 (6) (2014) 150.

[37] S. Fenz, Ontology-and bayesian-based information security risk management (September 2008).

[38] S. Fenz, A. Ekelhart, Formalizing information security knowledge, in: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security, ASIACCS '09, ACM, New York, NY, USA, 2009, pp. 183–194. doi:10.1145/1533057.1533084.
URL http://dx.doi.org/10.1145/1533057.1533084

[39] V. Nguyen, Ontologies and information systems: a literature survey, Tech. rep., DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION EDINBURGH (AUSTRALIA) (2011).

[40] C. Pahl, S. Giesecke, W. Hasselbring, An ontology-based approach for modelling architectural styles, ECSA: European Conference on Software Architecture 4758 (2007) 60–75. doi:10.1007/978-3-540-75132-8.
URL http://dx.doi.org/10.1007/978-3-540-75132-8

[41] F. B. Silva, S. Tabbone, R. da S. Torres, Bog: A new approach for graph matching, in: 2014 22nd International Conference on Pattern Recognition, ICPR-2014, Stockholm, Sweden, 2014, pp. 82–87. doi:10.1109/ICPR.2014.24.
URL http://dx.doi.org/10.1109/ICPR.2014.24

[42] F. B. Silva, R. de O. Werneck, S. Goldenstein, S. Tabbone, R. da S. Torres, Graph-based bag-of-words for classification, Pattern Recognition 74 (Supplement C) (2018) 266–285. doi:10.1016/j.patcog.2017.09.018.
URL https://doi.org/10.1016/j.patcog.2017.09.018

[43] L. Breiman, Random forests, Machine Learning 45 (1) (2001) 5–32. doi:10.1023/A:1010933404324.
URL http://dx.doi.org/10.1023/A:1010933404324

[44] R. Caruana, A. Niculescu-Mizil, An empirical comparison of supervised learning algorithms, in: Proceedings of the 23rd International Conference on Machine Learning, ICML '06, ACM, New York, NY, USA, 2006, pp. 161–168. doi:10.1145/1143844.1143865.
URL http://dx.doi.org/10.1145/1143844.1143865

[45] A. Narayanan, E. Shi, B. I. P. Rubinstein, Link prediction by de-anonymization: How we won the kaggle social network challenge, in: The 2011 International Joint Conference on Neural Networks, 2011, pp. 1825–1834. doi:10.1109/IJCNN.2011.6033446.
URL http://dx.doi.org/10.1109/IJCNN.2011.6033446

[46] A. Criminisi, J. Shotton, E. Konukoglu, Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning, Foundations and Trends® in Computer Graphics and Vision 7 (2-3) (2012) 81–227. doi:10.1561/0600000035.
URL http://dx.doi.org/10.1561/0600000035

[47] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, R. Moore, Real-time human pose recognition in parts from single depth images, Commun. ACM 56 (1) (2013) 116–124. doi:10.1145/2398356.2398381.
URL http://dx.doi.org/10.1145/2398356.2398381

[48] L. Breiman, J. Friedman, C. J. Stone, R. A. Olshen, Classification and Regression Trees, The Wadsworth and Brooks-Cole statistics-probability series, Taylor & Francis, 1984.
URL https://books.google.com.br/books?id=JwQx-WOmSyQC

[49] L. Breiman, Bagging predictors, Machine Learning 24 (2) (1996) 123–140. doi:10.1023/A:1018054314350.
URL http://dx.doi.org/10.1023/A:1018054314350

[50] G. James, D. Witten, T. Hastier, R. Tibshirani, An Introduction to Statistical Learning: With Applications in R, Springer Publishing Company, Incorporated, 2014. doi:10.1007/978-1-4614-7138-7.
URL http://dx.doi.org/10.1007/978-1-4614-7138-7

[51] L. Breiman, Out-of-bag estimation, Tech. rep., Statistics Department, University of California (1996).
URL https://www.stat.berkeley.edu/~breiman/OOBestimation.pdf

[52] C. Strobl, A.-L. Boulesteix, A. Zeileis, T. Hothorn, Bias in random forest variable importance measures: Illustrations, sources and a solution, BMC Bioinformatics 8 (1) (2007) 1–21. doi:10.1186/1471-2105-8-25.
URL http://dx.doi.org/10.1186/1471-2105-8-25

[53] A. Altmann, s. Laura Tolo O. Sander, T. Lengauer, Permutation importance: a corrected feature importance measure, Bioinformatics 26 (10) (2010) 1340–1347. doi:10.1093/bioinformatics/btq134.
URL http://dx.doi.org/10.1093/bioinformatics/btq134

[54] L. V. M. Medina, S. J. Rueda, Identifying android malware instructions, in: 2014 IEEE Latin-America Conference on Communications (LATINCOM), 2014, pp. 1–7. doi:10.1109/LATINCOM.2014.7041862.
URL http://dx.doi.org/10.1109/LATINCOM.2014.7041862

[55] N. F. Noy, M. Klein, Ontology evolution: Not the same as schema evolution, Knowledge and Information Systems 6 (4) (2004) 428–440. doi:10.1007/s10115-003-0137-2.
URL http://dx.doi.org/10.1007/s10115-003-0137-2

[56] M. Hartung, J. Terwilliger, E. Rahm, Recent advances in schema and ontology evolution, in: Z. Bellahsene, A. Bonifati, E. Rahm (Eds.), Schema Matching and Mapping, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 149–190. doi:10.1007/978-3-642-16518-4_6.
URL http://dx.doi.org/10.1007/978-3-642-16518-4_6

[57] M. Malhotra, T. R. G. Nair, Evolution of knowledge representation and retrieval techniques, International Journal of Intelligent Systems and Applications(IJISA) 7 (7) (2015) 18–28. doi:10.5815/ijisa.2015.07.03.
URL http://dx.doi.org/10.5815/ijisa.2015.07.03

[58] D. Beckett(W3C), T. Berners-Lee(W3C), E. Prud'hommeaux(W3C), I. Gavin Carothers(Lex Machina, Rdf 1.1 turtle - terse rdf triple language, Tech. rep., World Wide Web Consortium (W3C) (2 2014).
URL http://www.w3.org/TR/2014/REC-turtle-20140225/

[59] P. R. Rosenbaum, D. B. Rubin, The central role of the propensity score in observational studies for causal effects, Biometrika 70 (1) (1983) 41. doi:10.1093/biomet/70.1.41.
URL http://dx.doi.org/10.1093/biomet/70.1.41

[60] F. J. M. Jr., The Kolmogorov-Smirnov Test for Goodness of Fit, Journal of The American Statistical Association 46 (1951) 68–78. doi:10.1080/01621459.1951.10500769.
URL http://dx.doi.org/10.1080/01621459.1951.10500769

[61] L. H. Miller, Table of Percentage Points of Kolmogorov Statistics, Journal of the American Statistical Association 51 (273) (1956) 111–121. doi:10.1080/01621459.1956.10501314.
URL http://dx.doi.org/10.1080/01621459.1956.10501314

[62] Google, Google play, accessed: 2017-01-25 (2017).
URL https://play.google.com/store?hl=en

[63] U. du Luxembourg, Androzoo, accessed: 2018-04-29 (2018).
URL https://androzoo.uni.lu/

[64] Y. Zhou, X. Jiang, Dissecting android malware: Characterization and evolution, in: Proceedings of the 2012 IEEE Symposium on Security and Privacy, SP '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 95–109. doi:10.1109/SP.2012.16.
URL http://dx.doi.org/10.1109/SP.2012.16

[65] V. Community, Apk malware samples acquired from a torrent, accessed: 2017-01-25 (2017).
URL http://tracker.virusshare.com:6969/

[66] X. J. U. NSKeyLab, Andromalshare is a project, accessed: 2018-03-01 (2017).
URL http://sanddroid.xjtu.edu.cn:8080/

[67] V. Community, Virustotal public api v2.0, accessed: 2017-08-15 (2017).
URL https://www.virustotal.com/en/documentation/public-api/

[68] M. Sebastián, R. Rivera, P. Kotzias, J. Caballero, Avclass: A tool for massive malware labeling, in: F. Monrose, M. Dacier, G. Blanc, J. Garcia-Alfaro (Eds.), Research in Attacks, Intrusions, and Defenses: 19th International Symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings, Springer International Publishing, Cham, 2016, pp. 230–253. doi:10.1007/978-3-319-45719-2_11.
URL https://doi.org/10.1007/978-3-319-45719-2_11

[69] D. Beckett(W3C), Rdf 1.1 n-triples, Tech. rep., World Wide Web Consortium (W3C) (2 2014).
URL https://www.w3.org/TR/n-triples/

[70] Stanford-BMIR, Protege software, accessed: 2017-06-19 (2017).
URL http://protege.stanford.edu/

[71] Wikipedia, Mobile threat monday: Android apps hide windows malware, accessed: 2018-03-01 (02 2018).
URL https://en.wikipedia.org/wiki/Virut

[72] M. Eddy, Mobile threat monday: Android apps hide windows malware, accessed: 2018-03-01 (12 2014).
URL https://goo.gl/SFJqR9

A. Android Permissions Ecosystem Ontology

In the following, we present the terminology section of AndroPermEco ontology using Turtle/RDF syntax [58].

```
@prefix :<http://www.semanticweb.org/owl/owlapi/turtle#> .
@prefix owl:<http://www.w3.org/2002/07/owl#> .
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml:<http://www.w3.org/XML/1998/namespace> .
@prefix xsd:<http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#> .
@base <http://www.w3.org/2002/07/owl#> .
[ rdf:type owl:Ontology] .

#####################################################
#    Classes
#####################################################
:APIRead       rdf:type       owl:Class .
:APIWrite      rdf:type       owl:Class .
:Android       rdf:type       owl:Class ;
               rdfs:subClassOf owl:Package .
:AppDownloaded  rdf:type       owl:Class ;
               rdfs:subClassOf owl:Package .
AppPartner     rdf:type       owl:Class ;
               rdfs:subClassOf owl:Package .
:AppSystem     rdf:type       owl:Class ;
               rdfs:subClassOf owl:Package .
:Certificate    rdf:type       owl:Class .
:ContentProviderrdf:type       owl:Class ;
               rdfs:subClassOf owl:Interface .
:Dangerous     rdf:type       owl:Class ;
               rdfs:subClassOf owl:Permission .
:DataContent    rdf:type       owl:Class .
:Development    rdf:type       owl:Class ;
               rdfs:subClassOf owl:Permission .
:ExportedTrue   rdf:type       owl:Class ;
               rdfs:subClassOf owl:InterProcessComm .
:IntentFilter   rdf:type       owl:Class ;
               rdfs:subClassOf owl:InterProcessComm .
:InterProcessComm rdf:type     owl:Class ;
               rdfs:subClassOf owl:Interface .
:Interface     rdf:type       owl:Class .
:No_Defined_Class rdf:type     owl:Class .
:Normal        rdf:type       owl:Class ;
               rdfs:subClassOf owl:Permission .
:Package       rdf:type       owl:Class .
:PathContent    rdf:type       owl:Class .
:PathPermRead   rdf:type       owl:Class ;
               rdfs:subClassOf owl:PathPermission .
:PathPermWrite  rdf:type       owl:Class ;
               rdfs:subClassOf owl:PathPermission .
:PathPermission rdf:type       owl:Class ;
               rdfs:subClassOf owl:ContentProvider .
:Permission    rdf:type       owl:Class .
:Provider      rdf:type       owl:Class ;
               rdfs:subClassOf owl:ContentProvider .
:ProviderRead   rdf:type       owl:Class ;
               rdfs:subClassOf owl:Provider .
:ProviderWrite  rdf:type       owl:Class ;
               rdfs:subClassOf owl:Provider .
:Resource      rdf:type       owl:Class .
:Signature     rdf:type       owl:Class ;
               rdfs:subClassOf owl:Permission .
:SignatureOrSystem rdf:type    owl:Class ;
               rdfs:subClassOf owl:Permission .
:SysAPIRead    rdf:type       owl:Class ;
               rdfs:subClassOf owl:SystemAPI .
:SysAPIWrite    rdf:type       owl:Class ;
               rdfs:subClassOf owl:SystemAPI .
:SystemAPI     rdf:type       owl:Class ;
               rdfs:subClassOf owl:Interface .
:Undefined     rdf:type       owl:Class ;
               rdfs:subClassOf owl:Permission .
#####################################################
#    Object Properties
#####################################################
:controls      rdf:type      owl:ObjectProperty ;
               rdfs:domain    :Resource ;
               rdfs:range     owl:Android .
:exposes       rdf:type       owl:ObjectProperty ;
               rdfs:domain    owl:Interface ;
               rdfs:range     owl:Package .
:isProtectedBy  rdf:type       owl:ObjectProperty ;
               rdfs:domain    owl:Interface ;
               rdfs:range     owl:Permission .
:isSignedBy    rdf:type       owl:ObjectProperty ;
               rdfs:domain    owl:Package ;
               rdfs:range     owl:Certificate .
:managesData    rdf:type       owl:ObjectProperty ;
               rdfs:domain    owl:DataContent ;
               rdfs:range     owl:Package .
:managesPath    rdf:type       owl:ObjectProperty ;
               rdfs:domain    owl:PathContent ;
               rdfs:range     owl:Package .
:readsData     rdf:type       owl:ObjectProperty ;
               rdfs:domain    owl:ProviderRead ;
               rdfs:range     owl:DataContent .
:readsPath     rdf:type       owl:ObjectProperty ;
               rdfs:domain    owl:PathPermRead ;
               rdfs:range     owl:PathContent .
:readsResource  rdf:type       owl:ObjectProperty ;
               rdfs:domain    owl:APIRead ;
               rdfs:range     owl:Resource .
:requestsPermission rdf:type    owl:ObjectProperty ;
               rdfs:domain    owl:Package ;
               rdfs:range     owl:Permission .
:signsPackage   rdf:type       owl:ObjectProperty ;
               rdfs:domain    owl:Certificate ;
               rdfs:range     owl:Package .
:writesData    rdf:type       owl:ObjectProperty ;
               rdfs:domain    owl:ProviderWrite ;
               rdfs:range     owl:DataContent .
:writesPath    rdf:type       owl:ObjectProperty ;
               rdfs:domain    owl:PathPermWrite ;
               rdfs:range     owl:PathContent .
:writesResource rdf:type       owl:ObjectProperty ;
               rdfs:domain    owl:APIWrite ;
               rdfs:range     owl:Resource .
```

2.2 Connecting the Dots: Toward Accountable Machine-Learning Printer Attribution Methods

Title: Connecting the Dots: Toward Accountable Machine-Learning Printer Attribution Methods [24].
Authors: Luiz C. Navarro, Alexandre K. W. Navarro, Anderson Rocha, Ricardo Dahab.
Status: Published in Journal of Visual Communication and Image Representation, Volume 53, May 2018, Pages 257—272, ISSN 1047-3203, Elsevier. Received 6 June 2017, Revised 20 March 2018, Accepted 11 April 2018, Available online 13 April 2018.
DOI: 10.1016/j.jvcir.2018.04.002

Journal of Visual Communication and Image Representation 53 (2018) 257–272

Contents lists available at ScienceDirect

Journal of Visual Communication and Image Representation

journal homepage: www.elsevier.com/locate/jvci

Connecting the dots: Toward accountable machine-learning printer attribution methods☆

Luiz C. Navarro[a,*], Alexandre K.W. Navarro[b], Anderson Rocha[a], Ricardo Dahab[a]

[a] Institute of Computing – University of Campinas (Unicamp), Campinas, SP, Brazil
[b] Engineering Department – University of Cambridge Cambridge, UK

ARTICLE INFO

Keywords:
Accountable machine learning
Digital forensics
Source printer attribution
Feature back-projection
Feature mapping
Feature importance

ABSTRACT

Digital forensics is rapidly evolving as a direct consequence of the adoption of machine-learning methods allied with ever-growing amounts of data. Despite the fact that these methods yield more consistent and accurate results, they may face adoption hindrances in practice if their produced results are absent in a human-interpretable form. In this paper, we exemplify how human-interpretable (a.k.a., accountable) extensions can enhance existing algorithms to aid human experts, by introducing a new method for the source printer attribution problem. We leverage the recently proposed Convolutional Texture Gradient Filter (CTGF) algorithm's ability to capture local printing imperfections to introduce a new method that maps and highlights important attribution features directly onto the investigated printed document. Supported by Random Forest classifiers, we isolate and rank features that are pivotal for differentiating a printer from others, and back-project those features onto the investigated document, giving analysts further evidence about the attribution process.

1. Introduction & related work

Over the past decade, machine-learning methods have attained substantial importance in the field of digital forensics. Such techniques have been successfully applied in image and video forgery detection [1], predatory conversation identification in social media [2], face expression recognition [3], attribution of documents to their source printers [4], among others. Of particular interest, attributing printers to documents is a problem of practical relevance to forensic analysts when connecting printers and available evidence (such as forged documents, fraudulent reports, and fake bills) apprehended in search-and-seizure operations. Despite advances in electronic documenting and digital signature algorithms, integrity enforcement, authentication and non-repudiation methods, our society continues to produce printed and physically-signed documents for official purposes, posing a constant need for source attribution techniques of questioned documents.

Traditional methods for printer attribution often use physical properties of the paper and ink to determine the association between printers and printed documents. Techniques may use, for example, an infra-red spectrometer equipped with a microscope [5], or reactive dyes, chemical assays, and microscopy [6]. Other works [7] rely on Fourier transform spectroscopy to perform spectral discrimination and detect counterfeit documents. The costs involved in these methods are substantial as they often require expensive made-to-order equipment and specialized personnel. Moreover, methods such as those involving chemical analyses can lead to unintended consequences such as damaging or destroying apprehended evidence.

An alternative to these approaches is to focus on printer defects of malfunctioning, as captured on the scanned images of a printed document and use image processing techniques to identify the document's source. Such methods are based on intrinsic signatures extracted from the document's image: texture characterization methods, as described by Chiang et al. [8,9], and geometric distortions on printed pages, as investigated by Shang et al. [10].

The banding effect, which encompasses cyclical space variations on the halftones distance and ink density, has also been the subject of investigation. Deviations produce such effects due to mechanical tolerances and defects of printer components such as axis eccentricity and motor drift. As such, they result in unique features for attributing a document to its printer. This technique has also eased cost-related concerns. Whereas previous experiments [11,12] required high-resolution document scanning for precise measurements, ranging from 1200 up to 8000 DPIs, more recent studies [13–15] have used 600-DPI

☆ This paper has been recommended for acceptance by Zicheng Liu.

* Corresponding author.
E-mail addresses: luiz.navarro@students.ic.unicamp.br (L.C. Navarro), akwn2@cam.ac.uk (A.K.W. Navarro), anderson.rocha@ic.unicamp.br (A. Rocha), rdahab@ic.unicamp.br (R. Dahab).

https://doi.org/10.1016/j.jvcir.2018.04.002
Received 6 June 2017; Received in revised form 20 March 2018; Accepted 11 April 2018
Available online 13 April 2018
1047-3203/ © 2018 Elsevier Inc. All rights reserved.

documents, which are less costly and easier to find in typical commercial scanners.

Another line of investigation for printer attribution considers geometric distortion methods to measure and correlate linear geometric distortions between the actual printed image and an expected ideal image, looking at characters extracted by OCR [16] or using estimated centroid variations from halftones [17,18].

Due to its power to represent intrinsic details of a printed document, textures have also been subject to research when attributing documents to their printers. Texture-based methods rely upon patterns across neighboring pixels created by imperfections such as toner ink melting and fixation problems; toner spread around letters and knurled contours. These methods benefit from image processing descriptors and machine-learning algorithms for the identification of discriminant patterns, further correlating them to the source printer of a document. One of the first authors to exploit texture features, Mikkilineni et al. [19] introduced the use of Gray-Level Co-occurrence Matrices (GLCMs) image descriptors over images of letters "e" extracted from 2400-DPI scanned documents allied with a simple KNN classifier for source printer attribution. The authors further improved their results by using Support Vector Machines (SVM) classifiers [20].

Ferreira et al. [4] experimented with 600-DPI images of scanned documents of 10 laser printers and created one of the first public standardized datasets in the area [21], which we also adopt in this work. The authors investigated the use of letters "e" extracted from documents as a whole and also rectangular non-overlapping regions cropped from documents usually containing several characters at once, referred to as frames. Various image descriptors including GLCM, Local Binary Patterns (LBP), and Histogram of Oriented Gradients (HoG) were investigated. The authors also introduced one description method tailored for the attribution problem and referred to as Convolution Textures Gradient Filter (CTGF). Tsai et al. [22] extracted microscopy images using 300× magnification of characters and applied descriptors such as LBP and GLCM (among others) in four different alphabets (English, Arabic, Chinese, and Japanese).

Despite the fact that image-processing and machine-learning techniques outperform traditional chemical-based methods and are more appropriate in some setups, these algorithms often do not provide clear explanations as for why and how each document is attributed to a printer. The lack of human-interpretable evidence is particularly troublesome in forensic science, where decisions made by forensic analysts inform investigations and therefore may lead to legal implications. Moreover, the use of machine-learning methods that do not provide human-interpretable outputs in the forensic analysis may be legally inadmissible shortly. An example of the emergence of such restrictions, the European Union has voted in 2016 a resolution to be implemented by mid-2018 regarding the rights to human-interpretable explanations when decisions that can significantly affect its citizens are founded on machine-learning algorithms [23,24]. Other countries may shortly follow this example, and it hallmarks the need for digital forensics researchers to devise and develop human-interpretable machine-learning methods and hold the algorithms accountable.

Drawing on these insights, in this paper, we highlight how human-interpretable machine-learning methods can be derived from non-interpretable ones. We extend upon the Convolutional Texture Gradient Filter (CTGF) algorithm introduced by Ferreira et al. [4] to analyze, isolate and produce visible and interpretable features, giving rise to the CTGF-Map algorithm. The proposed method investigates the source of a document by finding the most discriminant features for attribution and by back-projecting those features directly onto the document, showing analysts the most relevant regions used in the process. Finally, we also discuss empirical results for this new method, tradeoffs between interpretable and non-interpretable counterparts of CTGF and the use of these techniques alongside other forensic processes.

2. Background concepts

The following sections present key concepts and methods used in this paper and discuss some properties of the techniques.

2.1. Accountable machine learning and explainability

In the last few years, there has been increasing concern and interest in accountable machine learning. There are new dedicated conferences and workshops on the subject such as the ones promoted by FAT/ML organization [25], the International Conference on Machine Learning (ICML) Workshop on Human Interpretability in Machine Learning (WHI) [26] and the AAAI W11 Workshop on Human-Aware Artificial Intelligence [27]. These events have been focusing on *fairness*, *accountability*, and *transparency* concepts for machine learning algorithms and applications. Governments and Non-governmental organizations are issuing policies, directives and best practices concerning the use of technology, and recently focusing on consequences and human rights related to decisions made by algorithms. Some examples include European Parliament recently approved General Data Protection Regulation (GDPR) [24,23] as well as studies from the Center for Democracy & Technology (CDT) [28]. Most concerns are due to the misuse and ethics of machine learning applications and the human rights of data privacy, but also on how to demonstrate decision results in a way that humans can understand. Most publications in this area address one or more principles exposed by a recent MIT Technology Review study [29] in MIT Technology: *Responsibility*, *Explainability*, *Accuracy*, *Auditability*, and *Fairness*. Explainability is defined in FAT/ML organization Principles for Accountable Algorithms [30] as: "[To] ensure that algorithmic decisions, as well as any data driving those decisions, can be explained to end-users and other stakeholders in non-technical terms".

With this backdrop, the primary objective of our work herein is to provide forensic analysts with a printer attribution method that fits in the explainability concept above. Questioned documents source attribution is usually part of proof and evidence presented in court trials by experts to court members, who are not familiar with machine-learning methods, but they can more easily understand physical explanations with visual evidence presentation. Handwriting analysis, for example, is an old and routine technique to visually present in courts for manually-written letters, memos, and most common for signatures. In our case, we aim at providing experts with a machine-learning method, which can classify questioned documents with high accuracy and precision, and also can show, **visually**, which regions on the image were used by the algorithms to identify the source of the document in the decision-making process thereof.

2.2. CTGF image descriptor

Convolutional Texture Gradient Filter (CTGF) [4] is a computational forensics method for describing documents regarding features that can be used by machine-learning algorithms to attribute a document to its source printer. The original presentation for the CTGF method [4] followed an empirical standpoint. In this section, we review this method while providing an alternative explanation based on the underlying physical principles of laser printers as well as a probabilistic interpretation of the descriptor.

The core physical principles used by laser printers to generate documents are electrostatics, photonics, and thermal curing. Electrostatics is used in the first stage of printing a document when electrical charges ink powder is placed on the printer optical charged drum (OPC) that will carry out the printing (see the OPC drum and process steps in Fig. 1). OPC drum is uniformly charged in steps A and B; then for the printer to differentiate from blank and printed areas, the OPC drum needs to be anisotropically charged. This anisotropy is

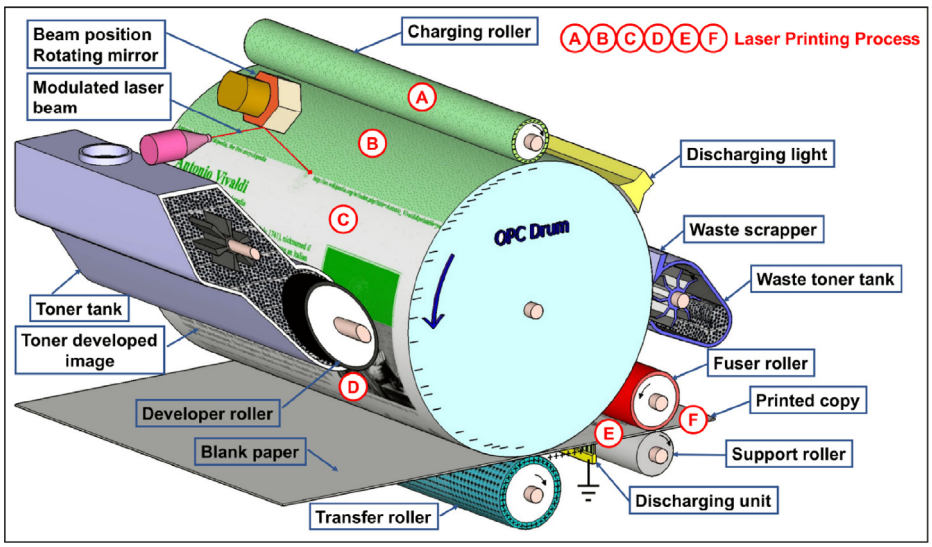

Fig. 1. Laser printer components and process. (a) Charging roller charges drum surface. (b) Drum's surface uniformly charged. (c) Controlled laser beam discharges printing blank areas. (d) Toner powder fills drum charged surface. (e) Toner grains transferred to paper by opposite charge attraction. (f) Toner melted on paper by fuser roller.

obtained through the selective discharging of the OPC's drum by a laser beam (step C). Ink particles are attracted to the drum by charged areas in step D and then transferred to the paper by touching the paper with the drum (step E) and fixated through a thermal unit, which fixes the ink particles to the paper (step F). The discharging unit removes charges from paper, waste scrapper removes residual toner, and discharging light cleans charges from the OPC drum preparing for the next printing cycle.

As a consequence of the printing process characteristics, variation in manufacturing and operation of each of the printer's components, there are differences among laser printers. These differences might be textures from ink powder patterns on white regions of the paper, flaws in letters' body or smudged patterns at the borders of letters or images. Drum defects, ink fixation problems, imprecise laser control, motor position accuracy, and drifts cause these textures. A unique distribution of textures arises in every printer, even if they come from the same manufacturer, have the same model or are printing the same document, as they depend on the individual physical components present in the printer. While traditional forensics practice identifies the unique printer footprints through physical assays, such as microscopy, the CTGF algorithm, in turn, attempts to identify attribution fingerprints through a machine-learning standpoint.

The task of identifying a printer from its unique footprints can be translated into a generative probabilistic model, in which the ultimate goal is to establish the probability of a printer, given the textures it generates on a document, to have created such document, that is $p(\boldsymbol{Y}|\boldsymbol{x})$. Generative probabilistic models are inherently interpretable as they allow to formalize the hypothesis that underpins how data are produced. For the printer attribution problem using printer-associated textures to identify the source of a given document, a reasonable model for how textures are included in a document can be described as.

1. Choose a printer from the set of suspect printers.
2. Choose a document or documents to be analyzed from the pool of possible ones to be used as references for that particular printer.
3. Extract and analyze textures pertinent to the selected printer from printed training documents.
4. Compare the extracted patterns from training data with patterns obtained from the investigated document for which we want to find the source printer.

Steps 1 to 3 above can be translated into a probabilistic generative model described below, while Step 4 is more related to a decision-making/learning mechanism to be applied later on:

1. Sample a printer $\boldsymbol{Y}$ from $p(\boldsymbol{Y})$,
2. Sample a document $\boldsymbol{w}$ from $p(\boldsymbol{w})$,
3. Sample textures $\boldsymbol{x}$ from $p(\boldsymbol{x}|\boldsymbol{Y},\boldsymbol{w})$,

with the classification Step 4 reflecting by computing the posterior $p(\boldsymbol{Y}|\boldsymbol{w})$. Using the rules of probability, the evaluation of the posterior can be defined through Bayes rule and by marginalizing the variable $\boldsymbol{x}$:

$$p(\boldsymbol{Y}|\boldsymbol{x}) = \int \frac{p(\boldsymbol{x}|\boldsymbol{w},\boldsymbol{Y})p(\boldsymbol{w})}{p(\boldsymbol{x})} \mathrm{d}\boldsymbol{w} \times p(\boldsymbol{Y}) \tag{1}$$

$$= \frac{p(\boldsymbol{x}|\boldsymbol{Y})}{p(\boldsymbol{x})} \times p(\boldsymbol{Y}). \tag{2}$$

The CTGF algorithm approximates the integral term by treating the document as an image and using image processing algorithms to generate a non-parametric estimate of the texture density. The adoption of this approach over defining parametric forms for the probability densities sidesteps the difficulty in explicitly determining probability distributions over documents as well as computing the high-dimensional and intractable integral in Eq. (1). The machine-learning method that uses the features produced by CTGF then approximately calculates Eq. (2) and provides either a probability of a printer given the textures identified or a point-estimate representing the most likely printer to have produced such textures.

Next, we examine how the CTGF obtains the printer texture distribution. The first step in CTGF is to represent a document as a scanned gray-scale 600-DPI image, that is, documents are represented as $r \times c$ matrices $\mathbf{S}$ with entries ranging from 0 to 255, indicating each pixel color range from white to black (negative grayscale image). The $\mathbf{S}$ matrix is convolved with $\mathbf{J}$ — a matrix of ones, square convolution window with odd dimension — and then, a filter $\mathbf{F}$ is multiplied element-wise with the image, resulting in a texture matrix. More formally, this procedure can be described as forming

$$\mathbf{C}[i,j] = (\mathbf{S}*\mathbf{J})[i,j] = \sum_{m=-\delta_\mathrm{r}}^{\delta_\mathrm{r}} \sum_{\ell=-\delta_\mathrm{c}}^{\delta_\mathrm{c}} \mathbf{S}[i+m, j+\ell] \tag{3}$$

where $\delta_\mathrm{r} = \frac{W_\mathrm{row}}{2}$ and $\delta_\mathrm{c} = \frac{W_\mathrm{col}}{2}$, then defining

$$\mathbf{T}[i,j] = (\mathbf{C} \circ \mathbf{F})[i,j] \tag{4}$$

where the filter $\mathbf{F}$ is defined image gradient, i.e.,

$$\nabla\mathbf{S}[i,j] = \max_{|a|\leqslant\delta_\mathrm{r},\, |b|\leqslant\delta_\mathrm{c}} |\mathbf{S}[i,j] - \mathbf{S}[i-a, j-b]| \tag{5}$$

and a band filter to generate

$$\mathbf{F}[i,j] = \begin{cases} 1 & \text{if} 0 > i > c, \quad 0 > j > r, \quad g_{lb} \leqslant \nabla\mathbf{S}[i,j] \leqslant g_{ub} \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Each of the entries in the matrix **T** will contain a number corresponding to a given pattern of pixels in a W_{row} by W_{col} window. By design, this procedure to encode pixel patterns has significant consequences for the learning: the convolution and filtering operations embed invariance to rotations and translations of the pixel patterns. This property directly impacts the choice of W_{row} and W_{col}, as larger W_{row} and W_{col} will consider a large number of patterns to be equivalent. On the other hand, small W_{row} and W_{col} values increase the size of **T** and, in the limit, where $W_{\text{col}} = W_{\text{row}} = 0$ the convolution is non-informative. This rationale supports the empirical analysis in [4], which found best results when $W_{\text{col}} = W_{\text{row}} = 3$ with the classification performance decreasing with larger kernel choices for W_{col} and W_{row}.

The last step in the CTGF is to transform the matrix **T** into a $(W_{\text{col}} \times W_{\text{row}} \times 255)$-dimensional count vector $\boldsymbol{b}$. Each entry of the vector $\boldsymbol{b}$ represents how many times a texture value appears in the document. This binning procedure is referred to as creating a histogram of textures in [4].

Features used in the machine-learning algorithm are obtained applying a normalization factor to the count vector, $\vec{\boldsymbol{x}}_i = \frac{1}{r \times c} \times \boldsymbol{b}_i$. Here, the normalizer $\lambda = r \times c$ can be linked to a physical quantity; the number of pixels on the image (number of elements of **S** matrix). Then, each component of the feature vector represents the density of the correspondent texture in an image. Considering that CTGF textures are linked to imperfections (in pure black and white printing) the sum of all elements in the feature vector corresponds to the density of defects per squared pixel in the image, which can be transformed just multiplying by the scanning resolution (DPI) in defects/in^2,

$$\rho = \sum_{i=0}^{W_{\text{col}} \times W_{\text{row}} \times 255} \vec{\boldsymbol{x}}_i \quad (defects/pixel^2). \tag{7}$$

From a probabilistic vantage point, different transformations applied to the count vector $\boldsymbol{b}$ imply in assuming different distributions for the incidence of textures. For example, the binning and normalization procedures used in the CTGF algorithm are analogous to modeling the frequency of features in Latent Dirichlet Allocation [31] as a Dirichlet-Categorical distribution.

The low-gradient filter highlights areas with smooth variations. This mathematical property translates into detecting irregular toner ink coverage in the letter's body and residual ink in non-printed regions.

A closer investigation of Fig. 2 shows six regions of interest (ROIs) captured by the CTGF low-gradient filtering process out of which only areas 1, 2 and 3 are of interest for attribution.

- **Region 1** - *ROI 1*: Imperfections on the toner fixation and melting within the letter's body produces texture gray tones with low-gradient values. In this area, a high CTGF value denotes variations that are almost invisible inside the body's letter, while a low value indicates a cluster with the absence of toner, which is precisely identified by readers as printing defects.
- **Region 2** - *ROI 2*: Lack of precision on the laser bean, motors variance, non-controlled motion inertia, and poor-printing resolution create jagged letter's contour with small portions of toner sliding around characters.
- **Region 3** - *ROI 3*: Such blank areas should not have any toner, but due to charge and discharge defects, particles of toner could adhere to the drum while being transferred to paper. Few toner dots are almost invisible to human eyes and are represented by CTGF values near 0. Conversely, high values indicate clusters of toner, which are seen as a dirty background.
- **Regions 4 and 5**: Correspond to 0-value gradients, indicating a flat single color printing, which is expected for the blank region outside letters (Area 4) and the letters' internal body (Area 5).
- **Region 6**: Letter's borders have high gradient thus printing imperfections cannot be detected by this method, then correspondent textures are eliminated by the filter.

2.3. Random forests classifier

In this section, we provide a succinct introduction to Random Forest classifiers. For a more comprehensive exposition of the underpinnings of Random Forests models, the interested reader is referred to references [32–34], while an expert on these algorithms may skip this section altogether.

Random forests [35] are a machine-learning technique that has enjoyed great success in academic [36] and commercial contexts, with applications spanning from social networks [37] to medical imaging analysis [32] and motion capture [38]. It can be defined as the combination of two pivotal concepts: classification and regression trees (CART) [39] and bootstrap aggregation (Bagging) [40].

CART models act by fitting a piece-wise function to the data by growing a binary decision tree. The piece-wise aspect of this model follows the intuition that the underlying function that generates the data can be approximated by splitting the input space $\boldsymbol{x}$ into Ω different regions $\mathscr{R}_m$, with $m = 1,\ldots,\Omega$ and associating to each region $\mathscr{R}_m$ a weight ω_m to match the expected value of the data in that region,

$$\boldsymbol{Y} \approx f(\boldsymbol{x}) = \sum_{m=1}^{\Omega} \omega_m \cdot \mathbb{1}\{\boldsymbol{x} \in \mathscr{R}_m\}, \tag{8}$$

where $\mathbb{1}$ is the indicator function. The binary tree element in CART models arises from the recursive structure used in generating the regions. Each region is defined by introducing a boundary function to split an existing region into two. Each of these separations forms a node in the binary tree. This boundary is chosen to minimize a classification or regression loss function that assesses the goodness-of-fit. To avoid over-fitting, after a full tree is grown, a pruning step is performed to reduce the tree complexity by selecting the best sub-tree using cross-

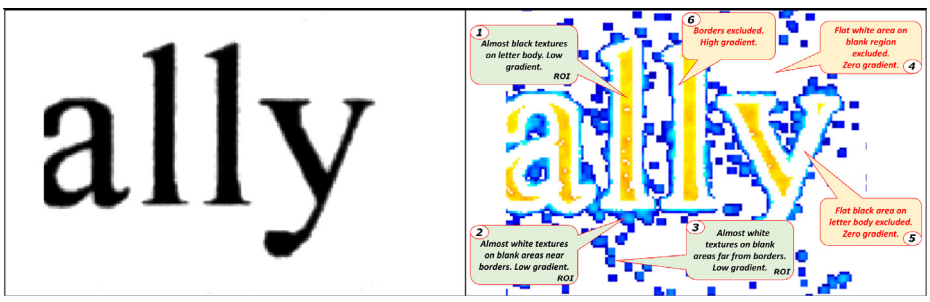

Fig. 2. The original fragment of a scanned document (left) and the corresponding textures for obtained using the CTGF gradient filter (right). The right image is colorized using yellow for near-black textures and blue for near white textures. Callouts indicate areas discarded and retained by the gradient-filtering process explained in this article and related to the gradient filter behavior. The picture on the right is a representation of matrix **T** computed with $g_{lb} = 1$ and $g_{ub} = 32$. Regions 4 and 5 correspond to filtering elements $\mathbf{F}_{i,j} < g_{lb}$; Regions 1, 2 and 3 corresponds to $g_{lb} <= \mathbf{F}_{i,j} <= g_{ub}$; and Region 6 is the result of filtering by $\mathbf{F}_{i,j} > g_{ub}$.

validation. For further details on tree pruning procedures, see [39,33].

As a consequence of the greedy nature of growing a CART model, they exhibit high dependence on the available training data. Random forests use the concept of bootstrap aggregation to overcome this limitation, which trains K models on randomly sub-sampled (with reposition) data from the original dataset. Then, the predictions from each model $f_k(\boldsymbol{x})$ are averaged,

$$f(\boldsymbol{x}) = \frac{1}{K}\sum_{k=1}^{K} f_k(\boldsymbol{x}), \tag{9}$$

which reduces the impact of a single data point and is guaranteed to reduce the estimator variance by the Rao-Blackwell theorem [41].

Besides lowering the variance and improving robustness of random forests, bagging also provides estimates for the generalization error and the probability of nodes, i.e., that a decision to split the feature space occurs at a given location. It is possible to show that each point $(\boldsymbol{x}_i, \boldsymbol{Y}_i)$ of the original set is only used in the training of 63% of the K trees [42]. Therefore predictions of the remaining trees that were not trained with the point $(\boldsymbol{x}_i, \boldsymbol{Y}_i)$ can be aggregated through averaging or majority voting to yield a cross-validation measure, the Out-of-bag (OOB) error.

The OOB error plays a central role in estimating the importance of a given feature in the predictions of random forests. One way to assess the importance of a variable based on OOB error is to evaluate the OOB error by randomly permuting the values of a given feature [35]. The rationale behind this procedure relies on the fact that if the predictor f is independent of a given feature and all remaining ones, then randomly permuting its values should not affect the OOB error [43]. When a feature is correlated to others, additional care should be taken to avoid overestimating their importance [44,45].

3. Proposed method

Our objectives in this work are twofold: we aim at designing and developing an effective printer attribution method while, at the same time, being able to directly point out, on the investigated document, the most important features used in the decision-making process.

The CTGFmap method was designed based on requirements to address the explainability concept mentioned previously Section 2.1. The technique aims at a visual identification of the most important regions (of the document) used by the classifiers in the source attribution process that are more present in the target class than on others. In other words, we are interested not only in finding the source printer of a document but also in the isolating the features that lead to the conclusion that a given printer generated the input document.

To satisfy the explainability requirement, we resorted to employing a visual mask over the target document. This mask would allow using the discriminant features to generate regions of interest in the original image. The visual map created by the mask over the target document needs the feature space representing image descriptors to be locally-based, i.e., we can identify in the document the locations that generate a specific feature component on the vector representation of the document. The CTGFmap descriptor meets this requirement at the pixel level, that is, identifying the locations of pixels that lead to specific features in the CTGF vector space. This process aids the forensic analyst with a physical meaning that can be precisely interpreted, differently from existing methods in prior art.

To achieve accountability, the feature should be quantified to represent the presence of a visual property. This requirement is essential to identify characteristics related to the target printer/class in comparison with documents produced by other printers. In a document's examination process, it is much easier to show elements which are more present as a consequence of some mechanical or electrical effect on the printed materials, than by the absence of artifacts that are existent on other classes, which require an extensive visual comparison. In this direction, we extended CTGF's method to represent the texture's density in the set of image pixels (by dividing the counting of texture activations of each CTGF value by the number of pixels in the document or region of interest). In this way, each feature represents the relative quantity of a specific texture inside the document fragment represented by a vector, which allows the selection of components with a higher probability of denser texture in each class of interest.

By satisfying the previous two requirements, we can identify features which are more present and discriminant in the target class, through statistical testing over each space component determining which classes have a more prominent probability to be denser, for which we referred to as 'positive' features of the class (see Section 3.1).

Given the positive features of the target class, it is important to determine which ones are discriminant and more relevant in the classes separation. Reducing the dimensionality is also a key factor for the visualization as a visual map with many points will not provide a good explanation to forensic analysts, neither help examiners to find the most relevant loci in the investigated documents.

We opted for the Random Forests in the classification step for a series of reasons including:

- **Feature Importance:** decision trees classifiers provide information about the relevance of each feature in the decision trees by evaluating how a change or omission of one variable impacts the classification results. Importance is a decisive property of the classification algorithm to provide the explainability principle of the proposed method.
- **High-classification performance:** high effectiveness on printer attribution using CTGF feature vectors (see Section 4.3), statistically comparable to the correspondent SVM classification used in previous CTGF articles.
- **Runtime:** in our experiments, Random Forests outperformed SVMs in execution time for training. For some printers, the speed-up was 60×.
- **No need of kernel and parametrization adjustments:** although SVMs with linear kernel performed well on CTGF multiclass classification under the OvO multi-class scheme, our initial tests using OvA [46] with SVMs were much slower to converge, and also led to worse performance. More tests could be done with adjustable grid-search parameters for the RBF kernel, for example, but as Random Forests already provide us with the essential property of built-in feature weighing, we opted to prioritize them. OvA scheme was selected as it facilitates the feature importance determination in a pairwise fashion (one vs. rest).

The discussed requirements and decisions led to the development of the CTGFmap algorithm that finally calculates saliency maps of the most discriminant features over the original documents. CTGFmap extends upon the original CTGF method [4] to include the *explainability* requirement of the source printer attribution process. Therefore its main novel aspects rely upon the explainability capacity and supervised dimensionality reduction process.

Based on the objectives explained above, the proposed method encompasses four main steps: Preprocessing, Learning, Back-projection, and Visualization, as Fig. 3 depicts.

The first stage, Preprocessing, consists of characterizing the investigated document (and possibly additional training ones) with a printer attribution description method. We split the input documents (scanned versions) into non-overlapping frames to produce more samples per analyzed document. Regions that are empty or containing only a small printed portion are discarded, as they are more susceptible to misclassification. Each remaining frame is converted into a feature vector using the CTGF descriptor as Section 2.2 describes.

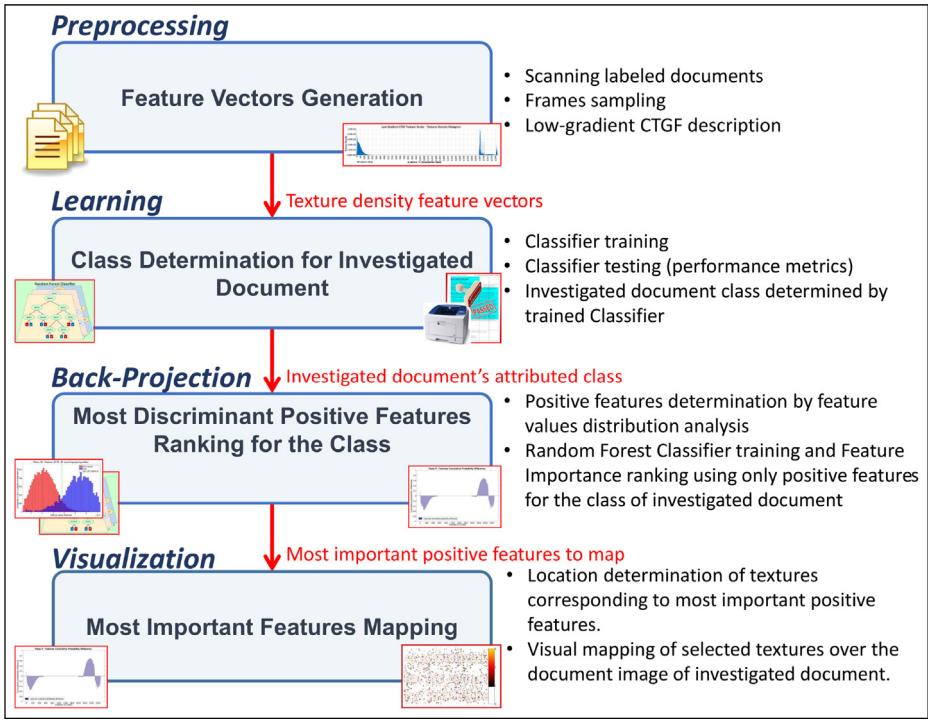

Fig. 3. Four-stage process underpinning the proposed printer attribution method in this paper.

Moving on to the learning activity, we learn the classifier responsible for differentiating documents of each suspect printer using a training set of documents associated with each one of them. To obtain the class of the investigated document, its feature vector is generated according to the same procedures for scanning, framing and description previously described, and then submitted to the trained classifiers. The ultimate result of this stage is the most likely class to have printed the investigated document.

Upon classifying a document, we call its predicted class as the target class. We then proceed to the next stage, which is the back-projection step (described in detail in Section 3.2). This stage assesses the relative importance of positive features — those associated with the class of interest or target class — and determines the minimum number necessary for classification. The positive features are identified through the analysis of the distribution of feature values for the target class (class of interest) and the remaining classes in the analyzed pool (negative cases in the training set). By using an iterative random forest feature importance algorithm, we can select the most discriminant features and attain the minimum necessary set to perform the attribution task.

Finally, the Visualization step (described in detail in Section 3.3) creates a saliency map highlighting the location of the most important features used in the classification process of the questioned document. In other words, this step finds in the investigated document, the most important aspects for attributing it to a particular printer.

3.1. Positive features determination

In this section, we analyze how to identify whether a feature is discriminant for a class against every other class in a pool through decision theory and observational causal inference. We refer to such features as *positive features*. The determination of which features are discriminant is paramount for obtaining greater insight on the classification produced by a machine-learning algorithm. In the particular case of source printer attribution, determining elements and features that are essential for classification may direct forensic investigators to areas within scanned documents that award further investigation.

We seek features which are more discriminant but are also denser in frames of the target printer when compared the other ones. As the feature vector measures the texture density, we want to identify the elements where the probability to find a frame with a higher value in the target class is higher than on frames of other classes in average. This analysis can be performed by assessing how different the distributions of a feature $\boldsymbol{x}_i$ are for classes $\mathscr{C}_0$ and $\mathscr{C}_1$ after training a classifier. To establish this comparison, we can compute a score to reflect this difference, since $p(\boldsymbol{x}_i|\mathscr{C}_0)$ and $p(\boldsymbol{x}_i|\mathscr{C}_1)$ will share the same support. The score should reflect that if a feature is important to differentiate between classes $\mathscr{C}_0$ and $\mathscr{C}_1$, then $p(\boldsymbol{x}_i|\mathscr{C}_0) \neq p(\boldsymbol{x}_i|\mathscr{C}_1)$. A simple criterion that is able to satisfy these requirements is the squared difference at each point,

$$\tau_i = \int (p(\boldsymbol{x}_i|\mathscr{C}_0) - p(\boldsymbol{x}_i|\mathscr{C}_1))^2 \mathrm{d}\boldsymbol{x}_i. \tag{10}$$

The score of Eq. (10) can be used to generate a ranking of how important each feature is to segregating the two classes.

Further to ranking which classes are more important, we can analyze what range of values for each feature is more frequent in each class. Note that partitioning the space into such areas is a classification problem in itself. For simplicity and interpretability, we have partitioned the space into only two regions, one where $\mathscr{C}_0$ is more predominant than $\mathscr{C}_1$ and another where $\mathscr{C}_1$ is more predominant than $\mathscr{C}_0$. To find the point $\beta^\star$ that provides maximal separation between these regions, we can compute:

$$\beta^\star = \underset{\beta}{\operatorname{argmax}} \int_{-\infty}^{\beta} \tau_i(p(\boldsymbol{x}_i|\mathscr{C}_0), p(\boldsymbol{x}_i|\mathscr{C}_1)) \tag{11}$$

as is illustrated in Fig. 4.

For C classes, the above-defined reasoning holds and the terms containing $p(\boldsymbol{x}_i|\mathscr{C}_1)$ should be replaced by $p(\boldsymbol{x}_i|\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_C)$ where C denotes the number of available classes in a pool of printers. Here we emphasize that the cutting point location $\beta^\star$ will be generally different

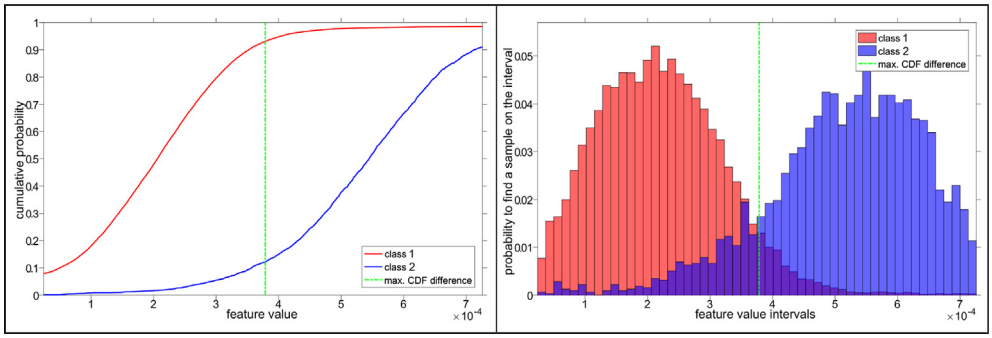

Fig. 4. Example of the distribution of two classes regarding their maximum cumulative probability difference.

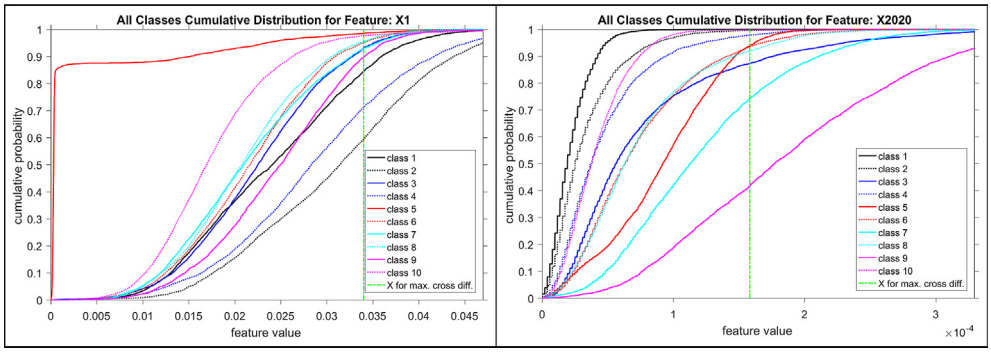

Fig. 5. $\beta^{\star}$ cutoff point determination for features X_1 and X_{2020}.

when multiple pair-wise decisions (one-versus-one) are made from the case where one class is compared to all others bundled together in a single comparison (one-versus-all). In particular, the maximum over the individual cut-points in Fig. 6 in the one-versus-one case will be lower or equal to the maximum of the one-versus-all cut-point in Fig. 5.

3.2. Backprojection

To obtain an informative visualization of discriminant features on the original document, only the subset which reflects the most relevant positive ones should be back-projected, as these features are responsible for carrying information toward accountability in document attribution.

The decision of which features to retain can be posed as an optimization problem for which we want to guarantee a minimum performance criterion for the classifier while keeping only the most discriminant positive ones. If we denote the set of most significant features as $\mathscr{I}$, this problem can be cast as

$$\begin{aligned} &\text{minimize} \quad \sum_{i=1}^{N} e_i, \\ &\text{subject to} \quad e_i = \mathbf{1}\{\boldsymbol{x}_i \in \mathscr{I}\}, \quad i = 1,...,N \\ &\qquad\qquad f(\boldsymbol{Y},\boldsymbol{x},\boldsymbol{e}) \geqslant \varepsilon, \end{aligned} \tag{12}$$

where e_i indicates whether the positive feature is in the discriminant feature set, f represents the performance criterion for Random Forest classifier using only features in $\mathscr{I}$, and ε is the minimum acceptable performance metric, which can be set by a forensic analyst, for instance.

The optimization problem in Eq. (12) is both discrete and NP-complete, and an algorithm to obtain approximate solutions is needed to efficiently find a local optimum considering the large number of features in the CTGF description method. Therefore, this paper proposes an iterative greedy algorithm to achieve this goal.

The algorithm introduced here is based on a recursive refinement of the important variable set. This refinement is performed by assessing how discriminant each feature is, regarding the final classifier metrics on a training set. Initially, the algorithm considers all positive features

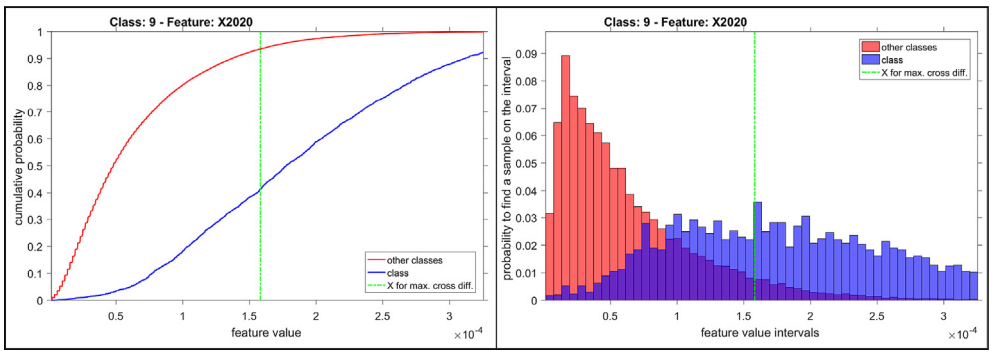

Fig. 6. Distribution comparison of features for one printer versus every other printer in a pool. Here, we show the positive feature X_{2020} of printer 9 (OKI Data model C330DN). See that the $\beta^{\star}$ cutoff point is beyond the maximum CDFs difference of class 9 versus all others, and it is the $\beta^{\star}$ determined as One versus One shown in Fig. 5.

Table 1
Printers used in the experiments.

Class	Printer	Brand	Model
1	B4070	Brother	HL-4070CDW
2	C1150	Canon	D1150
3	C3240	Canon	MF3240
4	C4370	Canon	MF4370DN
5	H1518	HP	CP1518
6	H225A	HP	CP2025A
7	H225B	HP	CP2025B
8	LE260	Lexmark	E260DN
9	OC330	OKI Data	C330DN
10	SC315	Samsung	CLP315

to belong to the discriminant feature set $\mathscr{I}$. Then, at each iteration, the algorithm removes the least relevant positive variables according to the out-of-bag criterion and trains a new Random Forest classifier. The newly trained Random Forest classifier is scored using the multi-class from binary (OvA) approach [46]. The procedure is stopped when either a minimum number of user-defined features is reached or if the classifiers performance metrics drop below the minimum acceptable value. The choice of the number of elements to remove in each step can be parametrized by a shrinkage parameter $\delta < 1$, where the number of features to drop is the ceiling of δ times the number of elements of $\mathscr{I}$.

3.3. Visualization

After finding the most discriminant features, these features need to be back-projected onto the original document to allow experts to visualize which regions are relevant for further screening.

To visualize the discriminant variables in the original document, we propose an algorithm to overlay the original scanned image with a color scale representing the ranked features in $\mathscr{I}$ using the out-of-bag assessment of the final classifier producing wgat we refer to as a saliency map. The color corresponding to the selected variable is applied to all locations in the convoluted image that match the features texture value. The related textures are colored from the least discriminant feature to the most important one, to avoid pixels belonging to the most discriminants to be overwritten by least significant ones.

For visualization purposes, the number of selected positive features must be kept low, and the color scale should be chosen for maximum contrast between adjacent textures. If the minimum number of positive features needed to achieve the minimum classification criteria is high, then the generated saliency map by the back-projection procedure would not be as informative to a human analyst.

4. Experiments and results

In this section, we present the performed experiments to validate the proposed methods as well as to compare them with methods in the literature.

4.1. Data preparation

For the experiments, we adopted the freely-available dataset introduced in [21,4], which comprises a total of 1,184 scanned images of documents printed in 10 printers (Table 1). Each image was split into eight rows by six columns of frames with the same number of pixels, after deleting six percent of pixels in each border of the raw image to eliminate external light noise during the scanning process. On average, out of 48 frames (6×8) per document, some 40 are selected as Table 2 shows. Empty frames and those with a small portion of printed material are not considered for classification, then a total of 1173 images remain from the original dataset. Those frames include text and pictures on the printed documents. Separated tests with frames containing only text and only figures are described in Section 4.7. A critical concern in the experiments is to avoid classifier overfitting. For this purpose, we randomly divided the documents (not the frames) of each printer into five partitions of approximately the same size (Table 2). One partition is then separated and isolated to participate only in the final tests. The remaining four partitions are used in four experiments each of which considering three partitions combined for the training process and the remaining one for validation (Table 3). The classifiers' performance is evaluated with the metrics presented in Section 4.2.

According to the definitions in Section 2.2, a CTGF feature vector is generated for each captured frame using a 3×3 one's matrix as the convolution kernel with gradient range from $g_{lb} = 1$ to $g_{ub} = 32$. Therefore, the possible obtained texture values vary from 0 to $3 \times 3 \times 255 = 2295$, and the final feature vector dimensionality for each frame is 2296. However, as we will discuss in Section 4.3, many of these features are not informative and, on average, we end up with less than 500 features for the actual learning and classification stages.

4.2. Performance metrics

For assessing the performance of the different methods discussed herein, we rely upon metrics associated with true positives (Tp), true negatives (Tn), false positives (Fp) and false negative (Fn), as described in Table 4. A true positive denotes a document correctly attributed to its actual printer P while a true negative is a document correctly negated as not belonging to P if another printer actually printed it. Complementarily, a false positive refers to a document incorrectly attributed

Table 2
Documents and frames distribution on the dataset partitions for the experiments.

Class	Total		Partitions														
			1			2			3			4			5		
	Doc	Frm	Doc	Frm	D/F	Doc	Frm	D/F	Doc	Frm	D/F	Doc	Frm	D/F	Doc	Frm	D/F
1	119	4,752	24	950	39.6	23	921	40.0	24	948	39.5	24	961	40.0	24	972	40.5
2	114	4,558	24	972	40.5	23	957	41.6	23	888	38.6	23	933	40.6	21	808	38.5
3	119	4,640	23	911	39.6	24	938	39.1	24	943	39.3	24	986	41.1	24	862	35.9
4	119	4,638	23	880	38.3	24	964	40.2	24	928	38.7	24	967	40.3	24	899	37.5
5	119	4,804	24	954	39.8	24	998	41.6	23	930	40.4	24	947	39.5	24	975	40.6
6	118	4,695	24	946	39.4	24	886	36.9	24	1,026	42.8	23	887	38.6	23	950	41.3
7	109	4,337	20	813	40.7	22	897	40.8	24	999	41.6	21	733	34.9	22	895	40.7
8	118	4,619	24	980	40.8	24	928	38.7	23	908	39.5	24	908	37.8	23	895	38.9
9	119	4,742	24	912	38.0	24	974	40.6	23	940	40.9	24	972	40.5	24	944	39.3
10	119	4,646	23	877	38.1	24	946	39.4	24	975	40.6	24	927	38.6	24	921	38.4
Total	1173	46,431	233	9195	39.5	236	9409	39.9	236	9485	40.2	235	9221	39.2	233	9121	39.1

Table 3
Partitions distribution of experiments for training, validation and final test. Partition 5 does not participate in training and validation, keeping separated for final test only on all experiments.

Experiment	Training			Validation			Final Test		
	Partitions	Docs	Vectors	Partition	Docs	Vectors	Partition	Docs	Vectors
1	1,2,3	705	28,089	4	235	9221	5	233	9121
2	1,2,4	704	27,825	3	236	9485	5	233	9121
3	1,3,4	704	27,901	2	236	9409	5	233	9121
4	2,3,4	707	28,115	1	233	9195	5	233	9121

Table 4
Metrics used to calculate the performance of classifiers.

Metric	Recall (RCL)	Specificity (SPC)	Precision (PRC)	F1-score (F1S)	Accuracy (ACC)
Formula	$\frac{Tp}{Tp+Fn}$	$\frac{Tn}{Tn+Fp}$	$\frac{Tp}{Tp+Fp)}$	$2\cdot\frac{Prc\cdot Rcl}{Prc+Rcl}$	$\frac{(Rcl+Spc)}{2}$

to a printer P, which was not its actual printer. Finally, a false negative refers to a document from a printer P that is incorrectly attributed to any other printer.

In our case, in addition to traditional roles of performance metrics, they are also used to establish minimum criteria to consider the results acceptable in the process of ranking and reducing vector dimensionality. Establishing criteria based on the performance metrics allows us to iterate reducing the vector length until any of these acceptance criteria is broken, indicating that features eliminated in the process are essential to keep results acceptable.

4.3. Class determination for target document

Table 5 shows obtained results for the validation partitions, while Table 6 shows final classification results with the separated test partition using Random Forests in an OvA fashion.

Upon experimenting with these documents and printers, one of the first findings is that many characteristics are not significant for all printers. Such variables were identified either as the ones whose maximum density in the vectors of all printers is less than 1×10^{-4} (which means a maximum of less than 0.01% of the frame area), or whose frequency is equal in all frames (particularly for features with value 0 for all vectors). Upon discarding those non-significant out of the 2296 features, only 547 remained to be used in the next steps. Fig. 7 shows the location and the maximum $\beta^{\star}$ of each significant feature.

A statistical comparison of `CTGF_3x3_F` SVM [4] with the `CTGFmap Random Forest` method presented here shows they are equivalent.

Looking beyond raw classification numbers, this is a compelling result as Random Forests provide us with a robust feature importance estimation method that can be used in the back-projection step later on. Recall our objective in this paper is to design and develop a robust printer attribution method, while at the same time being able to hold it accountable for its choices, i.e., to point out the main features underpinning the decision-making process and "visualize" them in the investigated document directly.

4.4. Target class positive features determination

At this stage, we already have the attribution done, and now we need to analyze the most important features used for decision making. The following steps should be applied to the class of the target document, determined in the previous phase. However, in this experimental procedure, we are interested in exploring samples from all printers and seeing the behavior of the most important features for each one. For this, we applied the frequency distribution analysis for all possible classes using concepts and procedures presented in Section 3.1. Table 7 exhibits the features classification in positive, similar and negative according to the rules in Section 3.1 for the comparison of a target printer/class ($T^f(\beta^{\star})$ to other printers ($O^f(\beta^{\star})$).

As we can see in Table 7, Class 5, which corresponds to the HP

Table 5
Average results of OvA Random Forest classifiers on the four validation experiments devised in this work with feature vector dimensionality equal to 547.

Class	Vector Length	True Pos	False Pos	True Neg	False Neg	Acc %	Rcl %	Spc %	Prc %	F1s %
1	547	91	4	841	4	97.66	95.79	99.53	95.79	95.79
2	547	88	6	841	5	96.96	94.62	99.29	93.62	94.12
3	547	85	7	838	10	94.32	89.47	99.17	92.39	90.91
4	547	87	3	842	8	95.61	91.58	99.64	96.67	94.05
5	547	87	0	845	8	95.79	91.58	100.0	100.0	95.6
6	547	77	12	833	18	89.82	81.05	98.58	86.52	83.7
7	547	83	16	837	4	96.76	95.4	98.12	83.84	89.25
8	547	93	6	839	2	98.59	97.89	99.29	93.94	95.88
9	547	95	1	844	0	99.94	100.0	99.88	98.96	99.48
10	547	95	3	842	0	99.82	100.0	99.64	96.94	98.45
					Mean	96.53	93.74	99.32	93.87	93.72
					σ	2.97	5.69	0.58	5.18	4.67

Table 6
Results considering OvA Random Forest classifiers trained with the four training partitions and tested with the final separated test partition. The feature vector dimensionality is 547.

Class	Vector Length	True Pos	False Pos	True Neg	False Neg	Acc %	Rcl %	Spc %	Prc %	F1s %
1	547	92	5	831	4	97.62	95.83	99.4	94.85	95.34
2	547	80	13	835	4	96.85	95.24	98.47	86.02	90.4
3	547	88	9	827	8	95.3	91.67	98.92	90.72	91.19
4	547	80	2	834	16	91.55	83.33	99.76	97.56	89.89
5	547	80	0	836	16	91.67	83.33	100.0	100.0	90.91
6	547	74	17	823	18	89.21	80.43	97.98	81.32	80.87
7	547	80	20	824	8	94.27	90.91	97.63	80	85.11
8	547	92	8	832	0	99.52	100.0	99.05	92	95.83
9	547	96	0	836	0	100.0	100.0	100.0	100.0	100.0
10	547	96	0	836	0	100.0	100.0	100.0	100.0	100.0
					Mean	95.6	92.07	99.12	92.25	91.95
					σ	3.86	7.47	0.87	7.64	6.09

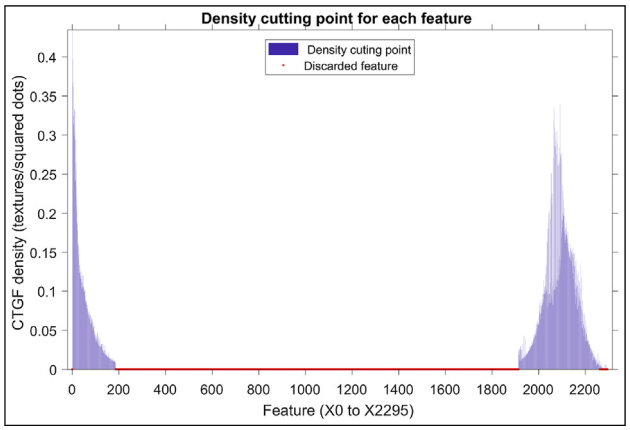

Fig. 7. Features' cutting point ($\beta^{\star}$) over all frames, showing that 547 out of the 2296 possible ones are significant.

Table 7
Result of distribution analysis for the 547 significant features. Textures density ρ is expressed in textures per squared dots.

Class	Pos. feat.	Sim. feat.	Neg feat.	ρ mean	ρ std
1	154	212	181	0.162	0.054
2	87	294	166	0.189	0.057
3	21	296	230	0.149	0.051
4	300	109	138	0.185	0.063
5	125	23	399	0.044	0.039
6	4	263	280	0.142	0.052
7	79	314	154	0.149	0.053
8	74	72	401	0.138	0.048
9	166	53	328	0.153	0.05
10	247	91	209	0.133	0.058

model CP1518, has a remarkably low density of textures captured ($\bar{\rho}$) representing approximately 4% of the frames area. This fact indicates a clean printing process and can be easily verified by mapping onto the investigated document all CTGF features of any fragment of a document printed with it, as Fig. 8 shows. As the CTGF saliency map of this printer is significantly different from others, it is indeed easy to verify a document attributed to this printer visually.

Other printers have $\bar{\rho}$ figures from 13% to 19% of textures, but we cannot conclude anything about their printing quality as we need to analyze the type of textures and their location to be more conclusive. Class 6, which represents HP model CP2025 (one of the two of the same model in the adopted dataset), has only four positive features, much less than other printers. It indicates that most of its features are common and less frequent than others, as it has $\bar{\rho}$ equivalent to the ones of other printers and a high number of negative features.

4.5. Ranking positive features of target classes

We obtained the results reported in Table 8 by applying the algorithm described in Section 3.2, using $\varepsilon = 0.6$ and $\delta = 0.1$ for each class of interest. Minimum discriminant length is highlighted, and the best F1Score achieved with positive features is also presented. Fig. 9 exemplifies the trajectory of metrics down to the performance criteria for two classes in the experiments.

We can see that by using only a few positive features, most of the classifiers already lead to good discrimination among printers — see that classification accuracy, recall, precision and f1score metrics are consistently over 80%, and most of them over 90%. The exceptions are classes 6 and 7 (both HP model CP2025) when using positive features. Class 3 (Canon model MF3240) has a minimum of 7 most discriminant positive features, which is almost on the limit of the easily distinguishable colors on the mapping scale.

Fig. 8. CTGF captured textures on fragments of the same document printed on printer 5 (HP model CP1518), left, and on printer 8 (Lexmark model E260DN), right. The difference of printer 5 with respect to the others is clearly seen when compared to others just using this map.

Table 8
Number of features of the resulting feature vectors containing the selected positive features. Blue font denotes best f1score achieve for each class, and red font indicates a metric that does not meet the criterion.

Class	len	vec len	Acc %	Rcl %	Spc %	Prc %	F1s %
1	**min**	**3**	95.21	95.74	94.68	67.33	78.96
1	best	40	97.43	96.88	97.99	84.97	90.28
1	max	154	97.38	96.88	97.88	84.35	89.86
2	**min**	**2**	84.36	72.15	96.58	71.80	70.87
2	best	40	88.53	78.49	98.58	86.68	82.23
2	max	87	85.38	72.06	98.70	85.97	78.37
3	**min**	**7**	81.21	63.13	99.29	90.99	74.51
3	best	14	85.24	70.61	99.88	98.75	81.96
3	max	21	83.98	68.43	99.53	94.14	79.05
4	**min**	**3**	84.04	70.56	97.52	76.14	73.16
4	best	300	96.44	93.70	99.17	92.74	93.15
4	max	300	96.44	93.70	99.17	92.74	93.15
5	**min**	**1**	93.68	89.49	97.87	83.83	86.30
5	best	6	95.67	91.58	99.76	97.78	94.54
5	max	125	94.97	90.53	99.41	94.96	92.50
6	max	4	57.29	14.81	99.76	90.83	24.92
7	max	79	90.97	88.74	93.20	57.48	69.47
8	**min**	**3**	91.81	90.49	93.13	60.03	72.07
8	best	66	98.25	97.92	98.58	89.16	93.10
8	max	74	97.21	95.83	98.58	89.04	92.04
9	**min**	**1**	88.33	82.11	94.56	63.46	71.20
9	best	7	99.29	100.0	98.58	88.82	94.06
9	max	166	99.11	100.0	98.23	86.74	92.78
10	**min**	**1**	91.54	85.33	97.75	81.12	82.99
10	best	12	99.82	100.0	99.64	97.00	98.45
10	max	247	99.41	100.0	98.81	90.90	95.09

All remaining printers have less than four positive features as a minimum, which makes it easy to display the connected textures on a saliency map. Except for printers 6 and 7, all other printers exhibit excellent best results of accuracy, recall, precision and f1score using only positive features correlating the printer discriminative information content on the related texture areas.

4.6. Back-projecting visualization and analysis of the three most discriminant features

Finally, visualization maps (Figs. 10 and 11) are created using the back-projection process previously described using fragments of documents of each printer and the algorithm described in Section 3.3, looking for how the most discriminant textures are distributed and located inside or outside letters. This information is also meaningful to understand the types of failure in the printing process that are producing them.

To complete the evaluation of the experiments conducted in this work, Table 9 lists the three most discriminant positive features for each printer of interest with their relative location to letter's body. Those features are the ones used in the back-projection and produced the maps exhibited in Fig. 10 for fragments of documents printed on all printers. For each of those fragments, the same digit "9" and a letter "e" are extracted and shown in Fig. 11, allowing a more detailed examination against the original grayscale scanned image of same characters. Most differences are almost imperceptible for human eye screening, but they are the textures used by the CTGF Random Forest classifiers to differentiate classes.

Patterns in documents printed in Canon D1150 (class 2 in Figs. 10 and 11) quickly identify the printer. They are almost white features (practically unseen texture patterns for readers) spread in a considerable quantity around letters and white areas. HP CP1518 (class 5 in Figs. 10 and 11) is also easy to classify as previously mentioned (see Fig. 8) by using just a few textures, consequently with a cleaner CTGF map than others.

Canon MF4370DN (class 4 in Figs. 10 and 11) has also toner sparsely spread on white areas, which are not visible by human eyes at the typical reading distance, but differently from class 2 already analyzed above, it has determinant textures in black areas within letters.

Samsung CLP315 (class 10 in Figs. 10 and 11) has an unusual pattern as it is the only one with textures in the range of 150 to 160 (grayscale mean about 16) of the CTGF description. These patterns appear on the border of letters indicating toner slide on their contour. There are just a few of these as the referenced figures depict, but enough to be discriminant as they are almost not present on other printers. This printer also has discriminant textures near black within letters.

By looking at Figs. 10 and 11 we also find that Printers 1, 3, 8, and 9 have their most discriminant features within letters, varying their patterns and density on the maps. However, Lexmark E260DN (class 8) is much more consistent with a small variation pattern aligned within

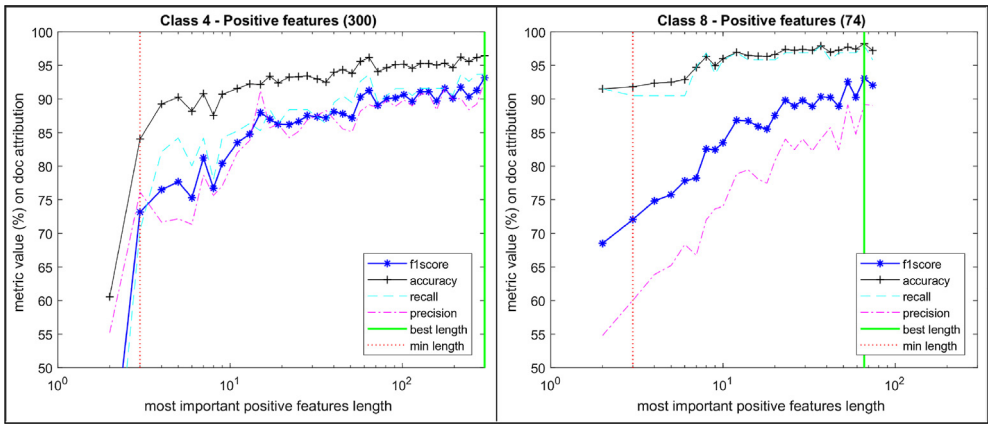

Fig. 9. Examples of most discriminant positive features search for class 4 (Canon MF4370DN) and class 8 (Lexmark E260DN).

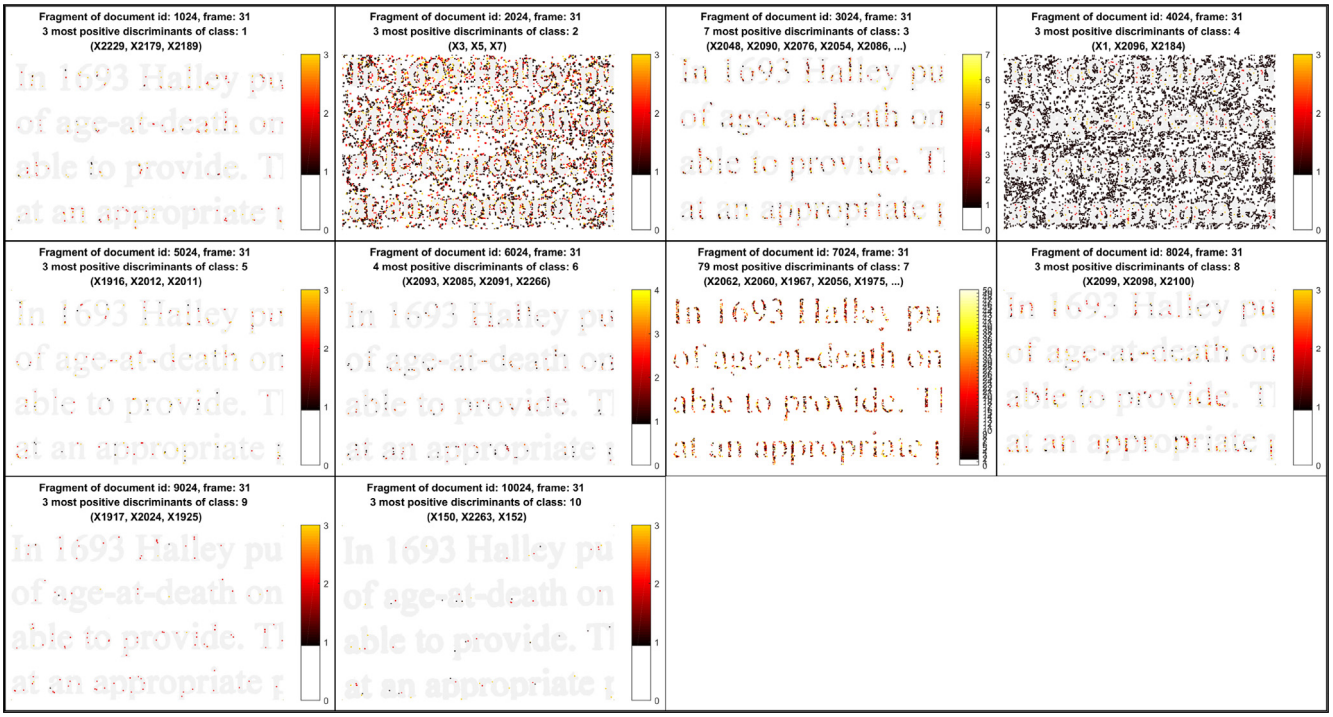

Fig. 10. Texture location maps for fragments of documents printed on each of the 10 printers. At least 3 or the minimum number of most discriminant features to achieve the minimum classification criterion (see Table 8) are mapped. Class 6 (HP model CP2025) has only two significant positive features but shows a very distinctive printing pattern.

letters' body, and Brother HL-4070CDW (class 1) has features in a range which points to more toner defects.

Pointing out the locus of each discriminant feature allows more precise examination of those patterns at subsequent evaluations. For instance, the consistent location of such patterns for some letters in a document might add extra information to the forensic analysis. This behavior has an analogy to consistent saturated pixels and pixel traps in a digital camera, which can be used for camera attribution [1]. Also, at microscopy level, such features (and their loci) could be used in microscopy to study the defects that are producing them.

4.7. Text and figures analyses

Experiments carried out in previous sections used frames containing text and figures as explained in Section 4.1. From the total of 1184 documents scanned, 595 (50.3%) are text-only frames, 578 (48.8%) includes figures which cover approximately 20% of the documents area, and remaining 11 (0.9%) are almost blank documents. Half of the documents is in English and half in Portuguese, but as all documents use alphabetic characters, this is irrelevant on the classification experiments. Most of all 56,304 frames are text-only (41,367 or 73.5%).

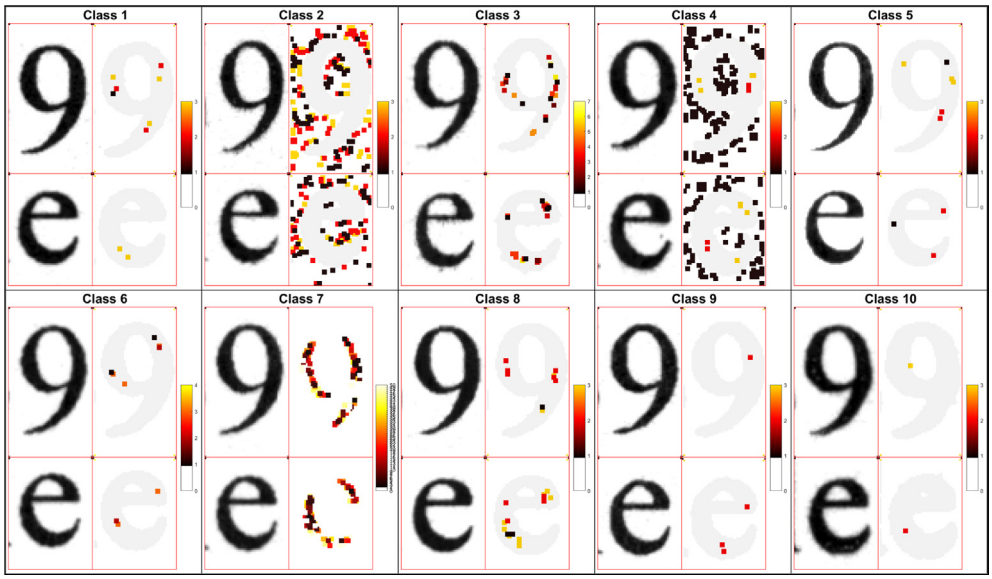

Fig. 11. Digit "9" and letter "e" extracted from each fragment in Fig. 10 magnified and juxtaposed side by side with the original grayscale scanned image.

Table 9
Three most discriminant features of each printer and their location. The two samples KS_test of all those features rejected the null hypothesis with *p values* equal to 1.

Class	Rank	Feature	$\beta^{\star}$	$D(\beta^{\star})$ %	Inside body ROI 1	Outside body: Near border ROI 2	Outside body: Far from border ROI 3
1	1	X_{2229}	7.74e−05	8.7	Few	No	No
1	2	X_{2179}	2.98e−04	29.8	Yes	No	No
1	3	X_{2189}	2.51e−04	27.3	Yes	No	No
2	1	X_{3}	1.53e−02	28.2	No	Many	Yes
2	2	X_{5}	9.37e−03	30.8	No	Many	Yes
2	3	X_{7}	6.42e−03	31.4	No	Many	Yes
3	1	X_{2048}	3.38e−04	5.8	Yes	No	No
3	2	X_{2090}	5.05e−04	5.0	Yes	No	No
3	3	X_{2076}	5.20e−04	7.8	Yes	No	No
4	1	X_{1}	3.26e−02	19.7	No	Many	Many
4	2	X_{2096}	5.29e−04	6.7	Yes	No	No
4	3	X_{2184}	3.55e−04	12.8	Yes	No	No
5	1	X_{1916}	3.68e−05	30.2	Yes	No	No
5	2	X_{2012}	1.41e−04	6.1	Yes	No	No
5	3	X_{2011}	1.14e−04	35.5	Yes	No	No
6	1	X_{2093}	2.45e−04	6.0	Yes	No	No
6	2	X_{2085}	2.18e−04	5.9	Yes	No	No
6	3	X_{2091}	2.33e−04	7.3	Yes	No	No
7	1	X_{2062}	4.28e−04	5.7	Yes	No	No
7	2	X_{2060}	3.88e−04	9.7	Yes	No	No
7	3	X_{1967}	5.02e−05	6.5	Yes	No	No
8	1	X_{2099}	3.29e−04	7.3	Yes	No	No
8	2	X_{2098}	5.24e−04	26.7	Yes	No	No
8	3	X_{2100}	5.14e−04	27.3	Yes	No	No
9	1	X_{1917}	4.64e−05	9.8	Few	No	No
9	2	X_{2024}	1.70e−04	44.1	Yes	No	No
9	3	X_{1925}	2.14e−05	48.5	Few	No	No
10	1	X_{150}	2.73e−05	49.8	No	Few	No
10	2	X_{2263}	1.55e−05	63.9	Yes	No	No
10	3	X_{152}	2.53e−05	51.6	No	Few	No

There are 4,838 (8.6%) frames with only figures (small pictures within documents), and 225 (0.4%) contain a mix of text and figures. Blank frames (9874 or 17.5%) inside documents are discarded as they do not contain printed areas (ink) and cannot be classified. By performing the experiments described in Section 4.3 using frames with only text and frames containing only figures, we obtained the final test results summarized in Table 10.

Then we statistically compared the results of recall, precision, and f1score in the three scenarios: all vectors (Table 6), text frames only and figure frames only (Table 10). As we can see, frames containing only figures account for the worst case of those experiments, but we should notice that the number of samples is much lower than in the other scenarios (about 12% of other experiments). Note, however, that documents in this dataset are mostly biographies, which generally include small portrait photos.

Despite needing to perform more tests with figure frames to better conclude what the classification performance with pictures is, we can conclude that if figures are not a significant part of the documents (in our case only about 20%), they are not significant (in our experiments results were slightly improved). However, classification with figures only should be more explored in future work.

Table 10
Final test performance metrics frames containing text only text and only figures.

	Vector Length	Acc %	Rcl %	Spc %	Prc %	F1s %
		Text Only Frames				
Mean	456	95.41	91.73	99.08	91.88	91.64
σ		4.05	7.71	0.88	7.65	6.57
		Figure Only Frames				
Mean	965	79.19	61.19	97.18	69.82	64.17
σ		13.6	26	2.53	28.07	25.49

4.8. Color and different types of paper sheet analysis

Testing questioned documents with conditions that are not included in the test set goes beyond the scope of the current work. It means, for instance, training a classifier with papers of one color and testing it with different colors. It is an open-set setup, a subject of increasing attention in the literature lately [47–51]. However, it is essential to understand the limits of current solutions to set grounds for additional research in this area. Therefore in this section, we present an initial study to discuss the impacts of this open-set formulation when considering different color paper sheets in an investigation and come up with some recommendations while further research is carried out.

We present a summary of tests using different color paper sheets because it is important to identify and establish requirements for the method's application and to guarantee that if the paper color changes, but representatives of that same color are included in the training set, then the technique would still work. Depending on the color of the paper, however, a gray background tone may be included in the scanned images distorting the feature values distribution.

For this purpose, we printed 19 documents using six different color paper sheets (white, beige, blue, green, rose, and yellow). Four of the previous printers available were used (class 1 Brother model HL-4070CDW, class 8 Lexmark model E260DN, class 9 OKI model C330DN and class 10 Samsung model CLP315). The same scanner Plustek model PL2546 scanned all documents. F1score results in Table 11 show that if the same paper color of tested documents (questioned ones) are included in the training set, then the results are comparable to those obtained on the first experiments using white paper.

4.9. Execution time analysis

All processing and performance measurements in this paper were done using a notebook Samsung 500R5H-XD3BR, Intel Core i7-5500CPU @ 2.40 GHz, 2 Cores, 4 Logical processors, 8 GB of physical memory, 1 TB HD 5400RPM SATA-III 6 GB/s. Programs were written in

Table 11
F1score metrics for experiments employing documents printed with different color paper sheets. "All" means a mix of documents with all paper colors. Bold numbers highlight that the training set includes documents with the same color as in the testing set. "NOK" denotes the test did not meet the minimum of 60% F1score while "NoTest" means the test was not performed.

		Training paper color: White	Beige	Blue	Green	Rose	Yellow	All
Tests paper color	White	**100.0**	88.15	89.58	*NOK*	*NOK*	*NOK*	**100.0**
	Beige	98.75	**100.0**	NoTest	NoTest	*NOK*	NoTest	**100.0**
	Blue	98.75	NoTest	**100.0**	NoTest	*NOK*	*NOK*	**100.0**
	Green	82.69	NoTest	NoTest	**100.0**	*NOK*	NoTest	**88.52**
	Rose	*NOK*	*NOK*	*NOK*	*NOK*	**94.44**	*NOK*	**94.44**
	Yellow	*NOK*	NoTest	*NOK*	NoTest	*NOK*	**100.0**	**100.0**
	All	82.91	75.80	74.41	61.59	*NOK*	60.75	**97.25**

Table 12
Summary of model training and certification processing times.

Step	Process Step Description	Total Elapsed Time hh:mm:ss
1	Framing: Get TIFF file for 1,184 scanned documents and split them into 48 frames (6x8) computing pixels metrics and CTGF descriptor for each frame.	01:32:29
2	Feature vectors generation: For the 56,832 frames produced on the previous step, eliminate blank frames and vector columns which has the same value in all vectors. Separate document vectors in partitions according the experiments plan.	00:03:16
3	Training with full length vectors (547 features), validation and final test using 4 experiments with partitions according to Table 3. Each classifiers has one instance per class (10 classes) of a OvA Random Forest classifier Training time per vector per OvA Random Forest classifier in μs: min = 337, mean = 482, max = 716, $\sigma = 89$ Validation or Final Test time per vector per OvA Random Forest classifier in μs: min = 44, mean = 53, max = 85, $\sigma = 11$ I/O operations: mean = 25.92 ms	01:36:47
4	Compute features cut point ($\beta^{\star}$) and classify features into Positive, Non-relevant or Negative.	00:40:25
5	Iterative Ranking: 35 iterations of 4 experiments including training and validation, reducing the vector length by a factor of 0.9 of positive features of each class in each iteration. Each iteration computes remaining features rank by sorting the feature importance over trained classifiers, and stops when all classes break the minimum performance criteria.	10:04:06
6	Final 4 experiments with training, validation and final test for each class using the best length of positive features determined in previous step.	00:32:17
	Total Process Time.	14:29:20

Table 13
Summary of document attribution and back-projecting maps processing time.

Step	Process description	Total Elapsed Time (sec.)
1	Feature vectors generation for the questioned document: Get TIFF file of scanned document and split into 48 frames (6x8) computing CTGF descriptor for each frame. Eliminate blank frames and discard columns which does not belong to the full vectors length (”547 features) included in the classifiers trained in step 3 of Table 12.	4.85
2	Source printer attribution for the questioned document using the full length vectors (547 features) classifiers trained in step 3 of Table 12.	4.90
3	Generate a visualization of the frames grid in the original document, as included in the appendix analysis report.	18.70
4	Back-projecting visualization of charts for one document frame: Gray Scale, CTGF map, CTGF most discriminant features, as included in the appendix analysis report.	3.50
5	Distribution charts for the 3 most discriminant features on the frame processed at previous step, as included in the appendix analysis report.	9.30
	Total Process Time (seconds)	41.26

Matlab script language and ran on Matlab R2017a 64-bit version 9.2.0.538062. During the computational time measurements, network interfaces were off, and no other program than Matlab was using user interfaces.

The total training time producing the model for classification and back-projecting feature selection takes about 14.5 h, including all steps described in Table 12. Note, however, this time is for the WHOLE training procedure for all documents and all printers under investigation. The source attribution and CTGF maps for a questioned document takes only 41 s according to steps in Table 13, where the most important ones are the source attribution, which takes 5 s, and the back projection and saliency map generation, with 3.5 s.

It is relevant to note that the Step 5 (Iterative Ranking) should be applied only to the classes of the questioned documents, not for all classes. The training time could be significantly improved with parallelization (cores and GPUs), but we did not focus on this aspect in this work as the accountability aspect is more significant and demanded a complete solution — the literature lacks attribution methods in this regard.

Just to compare the performance of the Random Forest classifier with SVMs in the same conditions, we replaced Random Forest by SVMs in Step 3 in Table 12 (the full-length vector training, validation, and testing). Step 3 running with SVM over the same dataset takes more than 30 h just for that step (in comparison, Random Forest takes about 1 h and 37 min), and for some classes more than 60 times the correspondent Random Forest classifier’s training time.

5. Conclusions and future work

Explainability is the most novel and significant achievement of the method we present in this paper (CTGFmap). We believe this is the first work in digital forensics to target the area of human-interpretable methods. Another novel aspect is the use of a Random Forest classifier to rank and select features. It can be utilized on other machine-learning problems whereby features selection focuses on marginal variables related to the target classes, i.e., a supervised dimensionality reduction method is required, as in those cases a non-supervised algorithm (e.g., PCA) will not be effective. Finally, the theoretical treatment we present for CTGF map is worth mentioning as another contribution of our work herein. Furthermore, we showed how this framework could be used to extend existing algorithms and be used to enhance other traditional forensic investigation analyses through the proposed CTGFmap algorithm.

Driven by the objective of mapping features, and looking for the requirements to achieve it, many aspects of printer attribution could be explored in this research, and many lessons were learned. The characteristics used for the classification should describe measurements that can be pointed back on the document. It requires a transformation that keeps the local information and an inverse one that recovers it. However, this requirement cannot always be attended.

Starting with the feature engineering part, it is worth noting how a simple image descriptor such as CTGF can capture discriminative laser printing properties. Although at this point it seems evident that printing imperfections or defects are relevant proxies for printer discrimination, to obtain such patterns using only 600-DPI documents is a remarkable

feat. Normalization is also crucial for facilitating the separation work of classifiers and also to express the relationship between classes correctly. An inadequate normalization can make classifiers work well, but destroy the local information needed for the physical characterization of mapping.

As expected, the identification of the mappable discriminant features is a challenge. Classification is usually a complicated process whereby variables are analyzed in conjunction. If the identification of positive features is easy and can be done looking at features individually, then the classification algorithm can be replaced by this analysis, and the mapping becomes a trivial task. That became apparent when we defined a substantial restriction to determine positive features just looking at the values distribution. The variables importance metric produced by the random forest algorithm evaluates how a feature affects the final classification result, and then allows us to rank the feature according to their relative relevance. The combination of the distribution analysis with feature ranking by the classifier enables the identification of features to map back into the document.

The entire method also follows a sequence of techniques and algorithms that together form a consistent and verifiable process, from the captured printers datasets generation to the investigated documents analysis report. The experiments conducted in this work show the proposed method produces good source attribution results for all printers, and most of them can have a saliency map of their most discriminant features applied. Some of them cannot lead to high discrimination as they have only a few positive features or the most frequent features are weak in the class separation process and, consequently, too many of them would be necessary to produce a minimum acceptable classification result. However, even in those cases the visualization of such features when back-projected onto the document is revealing and useful to a forensic analyst as Figs. 10 and 11 show.

We provide as supplemental material an example of an expert analysis report of an investigated document, which shows all concepts presented herein, including the attribution metrics, statistics and distribution charts of the most discriminant positive features mapping them onto the questioned document. We also publish the source-code and intermediate files of all experiments described in this work at (https://github.com/lcngit/CTGFmap.git).

Finally, the closed set of printers in the experiments could be extended to consider documents printed possibly by printers outside the investigated pool of printers, or papers with different characteristics such as paper colors not used in the training set. However, in this case, more analyses about the feature boundaries should be carried out. We regard this analysis as future work, and we believe the values distribution, feature importance, and also the mapping of decisions in trained trees can help on the task of defining boundaries for each printer on the set of interest and yield a probability evaluation of a document being printed by a printer outside the set of suspected ones.

The work presented in this paper spawns various future research avenues in the interaction between interpretable machine-learning methods and forensic science. Also, the framework prompts for generative modeling of forensic data. In this setting, one specifies a probabilistic model based on the knowledge of the underlying physical phenomena for a piece of evidence. In the printer attribution problem, the generation of an image can be distilled from the effect of the printer components in the final document. Finally, the procedure to determine which features are significant for the classifier can be used as a starting point in future work to analyze discriminant variables for other complex image classifiers such as convolutional neural networks..

Acknowledgment

We thank the financial support of Intel Strategic Research Alliance (Grant #440850/2013-4), the National Council for Scientific and Technological Development – CNPq (Grants #302224/2015-7 and #304472/2015-8), the São Paulo Research Foundation – Fapesp (DéjàVu Grant #2017/12646-3), and the Coordination for the Improvement of Higher Education Personnel – Capes (DeepEyes grant), as well as Cambridge Trusts-CAPES grant BEX 9407-11-1.

Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.jvcir.2018.04.002.

References

[1] A. Rocha, W. Scheirer, T. Boult, S. Goldenstein, Vision of the unseen: current trends and challenges in digital image and video forensics, ACM Comput. Surv. 43 (4) (2011) 26:1–26:42, http://dx.doi.org/10.1145/1978802.1978805.
[2] M. Ebrahimi, C.Y. Suen, O. Ormandjieva, Detecting predatory conversations in social media by deep Convolutional Neural Networks, Dig. Invest. 18 (2016) 33–49, http://dx.doi.org/10.1016/j.diin.2016.07.001.
[3] A. Uçar, Y. Demir, C. Güzeliş, A new facial expression recognition based on curvelet transform and online sequential extreme learning machine initialized with spherical clustering, Neural Comput. Appl. 27 (1) (2016) 131–142, http://dx.doi.org/10.1007/s00521-014-1569-1.
[4] A. Ferreira, L.C. Navarro, G. Pinheiro, J.A. dos Santos, A. Rocha, Laser printer attribution: exploring new features and beyond, Forensic Sci. Int. 247 (2016) 105–125, http://dx.doi.org/10.1016/j.forsciint.2014.11.030.
[5] T. Gal, J. Sandor, A. Karoly, Application note an#409 determining the sequence of crossed lines by ft-ir-atr-microscopy. <https://tinyurl.com/y9m9smyk> (accessed: 18.10.2017).
[6] G.M. LaPorte, Chemical Analysis for the Scientific Examination of Questioned Documents, Wiley-Blackwell, 2015, <http://www.wiley.com/WileyCDA/WileyTitle/productCd-1118897722.html>.
[7] E.B. Brauns, R.B. Dyer, Fourier transform hyperspectral visible imaging and the nondestructive analysis of potentially fraudulent documents, Appl. Spectrosc. 60 (8) (2006) 833–840, http://dx.doi.org/10.1366/000370206778062093 pMID: 16925917.
[8] P.-J. Chiang, N. Khanna, A.K. Mikkilineni, M.V.O. Segovia, S. Suh, J.P. Allebach, G.T.C. Chiu, E.J. Delp, Printer and scanner forensics, IEEE Signal Process. Mag. 26 (2) (2009) 72–83, http://dx.doi.org/10.1109/MSP.2008.931082.
[9] P.-J. Chiang, N. Khanna, A.K. Mikkilineni, M.V.O. Segovia, J.P. Allebach, G.T.C. Chiu, E.J. Delp, Printer and Scanner Forensics: Models and Methods, Berlin, Heidelberg, Berlin, Heidelberg, 2010, http://dx.doi.org/10.1007/978-3-642-11756-5_7.
[10] S. Shang, X. Kong, Printer and Scanner Forensics, John Wiley & Sons, Ltd, 2015.
[11] G.N. Ali, A.K. Mikkilineni, J.P. Allebach, E.J. Delp, P.-J. Chiang, G.T. Chiu, Intrinsic and extrinsic signatures for information hiding and secure printing with electrophotographic devices, in: NIP and Digital Fabrication Conference, vol. 2003, Society for Imaging Science and Technology, 2003, pp. 511–515. <http://www.ingentaconnect.com/content/ist/nipdf/2003/00002003/00000002/art00015>
[12] A.K. Mikkilineni, G.N. Ali, P.-J. Chiang, G.T.C. Chiu, J.P. Allebach, E.J. Delp, Signature-embedding in printed documents for security and forensic applications, in: Proc. SPIE, vol. 5306, 2004, pp. 455–466. doi:http://dx.doi.org/10.1117/12.531944.
[13] K.-Y. Lee, Y. Bang, H.-K. Choh, New measurement method of banding using spatial features for laser printers, in: Proc. SPIE, vol. 7529, 2010, pp. 75290H–75290H–7. doi:http://dx.doi.org/10.1117/12.840480.
[14] J. Zhang, S. Astling, R. Jessome, E. Maggard, T. Nelson, M. Shaw, J.P. Allebach, Assessment of presence of isolated periodic and aperiodic bands in laser electrophotographic printer output, in: Proc. SPIE, Vol. 8653, 2013, pp. 86530N–86530N–7. doi:http://dx.doi.org/10.1117/12.2008818.
[15] J. Zhang, J.P. Allebach, Estimation of repetitive interval of periodic bands in laser electrophotographic printer output, in: Proc. SPIE, vol. 9396, 2015, pp. 93960J–93960J–9. doi:http://dx.doi.org/10.1117/12.2083547.
[16] Y. Wu, X. Kong, X. You, Y. Guo, Printer forensics based on page document's geometric distortion, in: 2009 16th IEEE International Conference on Image Processing (ICIP), 2009, pp. 2909–2912. doi:http://dx.doi.org/10.1109/ICIP.2009.5413420.
[17] O. Bulan, J. Mao, G. Sharma, Geometric distortion signatures for printer identification, Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 1401–1404, , http://dx.doi.org/10.1109/ICASSP.2009.4959855.
[18] Y. Ju, D. Saxena, T. Kashti, D. Kella, D. Shaked, M. Fischer, R. Ulichney, J.P. Allebach, Modeling large-area influence in digital halftoning for electrophotographic printers, in: Proc. SPIE, Vol. 8292, 2012, pp. 82920X–82920X–9. doi:http://dx.doi.org/10.1117/12.912769.
[19] A.K. Mikkilineni, P.-J. Chiang, G.N. Ali, G.T.C. Chiu, J.P. Allebach, E.J.D. III, Printer identification based on graylevel co-occurrence features for security and forensic applications, in: Proc. SPIE, Vol. 5681, 2005, pp. 430–440. doi:http://dx.doi.org/10.1117/12.593796.
[20] A.K. Mikkilineni, N. Khanna, E.J. Delp, Forensic printer detection using intrinsic signatures, in: Proc. SPIE, Vol. 7880, 2011, pp. 78800R–78800R–11. doi:http://dx.doi.org/10.1117/12.876742.
[21] A. Ferreira, L.C. Navarro, G. Pinheiro, J.A. dos Santos, A. Rocha, Laser printer attribution: Exploring new features and beyond - datasets. <http://www.recod.ic.

unicamp.br/anselmo/printer_forensics_dataset/>.
[22] M.J. Tsai, I. Yuadi, Printed source identification by microscopic images, in: 2016 IEEE International Conference on Image Processing (ICIP), 2016, pp. 3927–3931. doi:http://dx.doi.org/10.1109/ICIP.2016.7533096.
[23] Parliament and Council of the European Union, General data protection regulation. <http://www.eugdpr.org/>
[24] B. Goodman, S. Flaxman, European union regulations on algorithmic decision-making and a "right to explanation", in: K.R.V. Been Kim, Dmitry M. Malioutov (Ed.), Proceedings of the 2016 ICML Workshop on Human Interpretability in Machine Learning (WHI 2016), 2016, pp. 26–30. Available from: < https://arxiv.org/abs/1606.08813 > .
[25] FAT/ML, Fairness, accountability, and transparency in machine learning. <http://www.fatml.org/>.
[26] K. Varshney, A. Weller, B. Kim, D. Malioutovx, Workshop on human interpretability in machine learning (whi) (August 2017). <https://sites.google.com/view/whi2017/home>.
[27] K. Talamadupula, S. Sohrabi, L. Michael, B. Srivastava, W11 - human-aware artificial intelligence (February 2017). <http://www.aaai.org/Workshops/ws17workshops.php#ws11>.
[28] CDT, Digital decisions, Tech. rep., Center for Democracy & Technology, 2017. <https://cdt.org/issue/privacy-data/digital-decisions/>.
[29] N. Diakopoulos, S. Friedler, How to hold algorithms accountable, MIT Technology Review. <https://www.technologyreview.com/s/602933/how-to-hold-algorithms-accountable/>
[30] N. Diakopoulos, S. Friedler, M. Arenas, S. Barocas, M. Hay, B. Howe, H.V. Jagadish, K. Unsworth, A. Sahuguet, S. Venkatasubramanian, C. Wilson, C. Yu, B. Zevenbergen, Principles for accountable algorithms and a social impact statement for algorithms, Tech. rep., FAT/ML Organization, 2017. <http://www.fatml.org/resources/principles-for-accountable-algorithms>.
[31] D.M. Blei, A.Y. Ng, M.I. Jordan, Latent dirichlet allocation, J. Mach. Learn. Res. 3 (2003) 993–1022 <http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf>.
[32] A. Criminisi, J. Shotton, E. Konukoglu, Decision forests: a unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning, Found. Trends®Comput. Graph. Vis. 7 (2-3) (2012) 81–227, http://dx.doi.org/10.1561/0600000035.
[33] G. James, D. Witten, T. Hastier, R. Tibshirani, An Introduction to Statistical Learning: With Applications in R, Springer Publishing Company, Incorporated, 2014. doi:http://dx.doi.org/10.1007/978-1-4614-7138-7.
[34] K.P. Murphy, Adaptive basis function models, Adaptive computation and machine learning, The MIT Press, 2012, Ch. 16, pp. 543–587.
[35] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32, http://dx.doi.org/10.1023/A:1010933404324.
[36] R. Caruana, A. Niculescu-Mizil, An empirical comparison of supervised learning algorithms, Proceedings of the 23rd International Conference on Machine Learning, ICML '06, ACM, New York, NY, USA, 2006, pp. 161–168, , http://dx.doi.org/10.1145/1143844.1143865.
[37] A. Narayanan, E. Shi, B.I.P. Rubinstein, Link prediction by de-anonymization: how we won the kaggle social network challenge, in: The 2011 International Joint Conference on Neural Networks, 2011, pp. 1825–1834. doi:http://dx.doi.org/10.1109/IJCNN.2011.6033446.
[38] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, R. Moore, Real-time human pose recognition in parts from single depth images, Commun. ACM 56 (1) (2013) 116–124, http://dx.doi.org/10.1145/2398356.2398381.
[39] L. Breiman, J. Friedman, C.J. Stone, R.A. Olshen, Classification and Regression Trees, The Wadsworth and Brooks-Cole Statistics-probability Series, Taylor & Francis, 1984, <https://books.google.com.br/books?id=JwQx-WOmSyQC>.
[40] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140, http://dx.doi.org/10.1023/A:1018054314350.
[41] D. Blackwell, Conditional expectation and unbiased sequential estimation, Ann. Math. Stat. 18 (1) (1947) 105–110, http://dx.doi.org/10.1214/aoms/1177730497.
[42] L. Breiman, Out-of-bag estimation, Tech. rep., Statistics Department, University of California, 1996. <https://www.stat.berkeley.edu/breiman/OOBestimation.pdf>.
[43] G. Louppe, L. Wehenkel, A. Sutera, P. Geurts, Understanding variable importances in forests of randomized trees, in: C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 26, Curran Associates, Inc., 2013, pp. 431–439 <https://tinyurl.com/ycrpfng9>.
[44] C. Strobl, A.-L. Boulesteix, A. Zeileis, T. Hothorn, Bias in random forest variable importance measures: illustrations, sources and a solution, BMC Bioinformat. 8 (1) (2007) 1–21, http://dx.doi.org/10.1186/1471-2105-8-25.
[45] A. Altmann, L. Tolosi, O. Sander, T. Lengauer, Permutation importance: a corrected feature importance measure, Bioinformatics 26 (10) (2010) 1340–1347, http://dx.doi.org/10.1093/bioinformatics/btq134.
[46] A. Rocha, S.K. Goldenstein, Multiclass from binary: expanding one-versus-all, one-versus-one and ecoc-based approaches, IEEE Trans. Neural Netw. Learn. Syst. 25 (2) (2014) 289–302, http://dx.doi.org/10.1109/TNNLS.2013.2274735.
[47] F.O. Costa, E. Silva, M. Eckmann, W.J. Scheirer, A. Rocha, Open set source camera attribution and device linking, Pattern Recogn. Lett. 39 (Supplement C) (2014) 92–101, http://dx.doi.org/10.1016/j.patrec.2013.09.006 advances in Pattern Recognition and Computer Vision.
[48] W.J. Scheirer, A. de Rezende Rocha, A. Sapkota, T.E. Boult, Towards open set recognition, IEEE Trans. Pattern Anal. Mach. Intell. (T-PAMI) 35 (2013) 1757–1772, http://dx.doi.org/10.1109/TPAMI.2012.256.
[49] L.P. Jain, W.J. Scheirer, T.E. Boult, Multi-class open set recognition using probability of inclusion, in: D. Fleet, T. Pajdla, B. Schiele, T. Tuytelaars (Eds.), Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part III, Springer International Publishing, 2014, pp. 393–409, , http://dx.doi.org/10.1007/978-3-319-10578-926.
[50] W.J. Scheirer, L.P. Jain, T.E. Boult, Probability models for open set recognition, IEEE Trans. Pattern Anal. Mach. Intell. (T-PAMI) 36 (11) (2014) 2317–2324, http://dx.doi.org/10.1109/TPAMI.2014.2321392.
[51] A. Rattani, W.J. Scheirer, A. Ross, Open set fingerprint spoof detection across novel fabrication materials, IEEE Trans. Inf. Forensics Secur. 10 (11) (2015) 2447–2460, http://dx.doi.org/10.1109/TIFS.2015.2464772.

Chapter 3

Discussion

It is evident in the cases analyzed in Chapter 2 that the translation of original problems into machine learning vector space took into account physical or logical properties which are the object of screening on the target solution. In the printer attribution problem, the CTGF texture [15] descriptor captures imperfections in the laser printing process which are typical and device dependent. In Android malware relationship analysis, the features extracted from the ontology graph uses the Bag of Words technique [30, 29] and expertise acquired on Android manifests analysis in the article [16]. Those are examples of the feature engineering techniques that are required to start the transformation from the complex model of the original problem into the features space required by the machine-learning algorithms.

An essential step in the proposed method is the supervised technique for finding discriminant features. The method allows for identifying the most relevant characteristics of the target class without changing the semantics of the original vector space. Usually, dimensionality reduction methods compound variables to favor classifier performance mixing into one variable semantics of different objects in the original model. That is the reason because interpretability becomes difficult and tricky.

Just as an example, *Principal Component Analysis* (PCA) when applied to the problems of articles of Chapter 2 destroyed the information which are the characteristic of the target classes, and that effect was observed during the research work to produce these articles.

Some of those reduction methods are difficult to reverse and to determine the original features which are relevant to the prediction algorithm. For example, *Partial least squares Discriminant Analysis* (PLS-DA) that uses the dependent variable information on the features reduction composition, and consequently preserves information associated to the predicted class, mix semantics of the original features into the new ones, therefore making more difficult to interpret features in the new vector space.

Fortunately the Random Forest algorithm [6], based on decision trees [9], [7] with a probabilistic interpretation of its principles in [22], provides the information about the gain of information in each decision step of the algorithm, consequently computing the importance of the feature tested in the tree node to discriminate the predicted class. This *Random Forest Variable Importance or Feature Importance*, as mentioned below, is crucial for the reduction method which preserves the semantics of the features during the reduction process. Although it seems costly to iterate the training of a classifier and measuring

prediction results, the work done in two different areas with complex models to analyze in Chapter 2 shown reasonable computing time on the problem solution, and mainly in the model interpretability. The bagging of decision trees machine learning algorithm has the following advantages when processing the data we have at hand:

- High-classification performance: Random Forest is one of the best classifiers for different problems [14].
- No need of kernel and parametrization adjustments: Random Forest is known as a non-parametrized method, which means it does not require an elaborate search of parameters, kernel transformation, neither it is sensitive to normalization of input data. Only two parameters are subject of tuning: the number of trees in the forest which is typically set between 64 to 128 for most of the cases [26], and the number of random features selected in each tree building cycle that is generally set to the square root of the vector length.
- Execution performance: A trained random forest classifier is a set of binary trees, which can be seen as a sequence of "if then else" statements being extremely fast at prediction time.
- Feature importance: Decision tree classifiers provide information about the relevance of each feature in the decision trees by evaluating how a change or omission of one feature impacts classification results. This is referred to as out of bag (OOB) evaluation concept [5], [2], [20]. Importance assessment is a crucial property of the classification algorithm to provide explainability and accountability of results achieved by the classifier.

In many practical problems, the only objective is to have a classifier that can predict a class based on the characteristics of the input data. In the case studies of the articles included in this dissertation, the process does not end when a high-performance classifier is produced. It is necessary to go beyond, identifying which of the original input variables are responsible for the high performance of the classifier, meaning that they are the crucial ones for the interpretability of the results achieved.

Accountability [10, 13, 33, 31, 1], as previously mentioned, is a demanding priority in artificial intelligence, due machine learning algorithms are involved more and more in decisions and activities which affect human life, individually or collectively. Interpretability directly involves the discriminant features determination, which is a crucial process also in research for identifying elements that entangle a complex physical or biological phenomenon.

With this as a backdrop, the primary objective of the CTGFmap article in Section 2.2 is to provide forensic analysts with a printer attribution method that fits in the explainability concept. Questioned documents' source attribution is usually part of proof and evidence presented in court trials by experts to court members, which are not familiar with machine learning methods, but they can more easily understand physical explanations with visual evidence presentation. Handwriting analysis, for example, is an old and routine technique visually presented in courts for manually-written letters, memos, and most common for signatures.

The CTGFmap method was designed based on requirements to address the explainability/accountability principle of accountable machine learning as defined in [11, 10, 33, 31] above cited. The method aims a visual identification of the regions which classifiers used in the source attribution process and are more present in the attributed class than on other documents produced by other printers. In other words, the focus of the article is not only in finding the printer source of a document, with high accuracy and precision, but also which features/parts in the investigated document lead to that conclusion of the classifier. This aspect is the most novel and significant achievement of CTGFmap in printer-source attribution methods.

The Android research started with a broad objective to create a model for the operating system vulnerabilities based on its structural components and resources. Although the work done in this dissertation addressed only a small part of the fundamental vulnerability question, the promising starting point was attained using ontologies and machine learning techniques together. Results showed that ontologies and machine learning could leverage solutions to a problem solution. The significant constraint for addressing the big picture was the availability of structured information about the "modus operandi" of the attacks, and the organization of available information that can be attached to the application samples. It means that there is an excellent opportunity for the semantic formalization and systematic organization of malware according to characteristics; to cite some: attack objectives, attack operation, main and derived code, resources involved, damage caused.

Ontology is a powerful concept for organizing information and providing tools for manipulating and extracting value. Machine learning is a very effective tool for analyzing and discovering hidden relationships inside data. Together, they form a robust combination of exploration and discovery. However, the present lack of standardized concepts and structured information (most of the descriptions are in natural language) on malware code analysis for ontology definition were strong constraints faced during the work.

Maybe one of the reasons for the difficulty in structuring information is the nature of security problems, where malicious methods have to change regularly for attacks to succeed; but we should also consider that many known attacks continue to create havoc, as they are introduced in different applications and used with different strategies.

Although Software Engineering and its software development methodologies evolved a lot during the last decades, software architecture and design methodologies did not produce consistent and shareable documentation which could be analyzed with automated methods such as proposed in this work. This lack of formalization in the software development industry is a challenge for the future and depends on industry standards, and mainly in the development of reverse modeling techniques and tools to capture information on systems already developed.

Considering that the malware detection article addressed part of the entire initial problem, we now propose some future work to extend the scope coverage.

The Android permissions model can be expanded to include information already available in the documentation about System APIs protected by each permission, and also results of static code analysis or execution monitoring using tools available for Android malware scanning. The number of malware and benign applications in the repository is high and much more work will be necessary. However, the captured data will give

more information on the relationships and attacks, and as it is organized and stored as ontologies, the model can grow and evolve without the effort of data structures redesign.

Coming from another front, ontologies, and taxonomies which classify malware by their behavior, type of attack and code metamorphosis can be easily linked with the enhanced ontology proposed in the previous paragraph. In this way, malware applications will be described with more precise semantic linking the behavior information (e.g., attack objectives and methods) with their implementation (e.g., malicious code, resources usage). That project is not only an academic exercise but a current security market need, as the vulnerability databases like CVE [4] are focused on affected products, impacted versions, and versions to resolve. The detailed technical description is always superficially touched.

As a concrete example, the AndroPermEco ontology proposed in Section 2.1 can be extended to include all information from manifests, and also captured by tools, e.g., those in articles [12, 3, 34, 35, 32, 28], and others surveyed in [23]. Then, merge the ontology with taxonomies and ontologies developed by the authors of articles [19, 18], and also with other ontologies [25] proposed for linking with official vulnerabilities, attacks and countermeasures databases [4].

A simple conclusion about the nature of the above projects shows a need in the Android Security arena for a more structured approach using ontologies to store and process data from malware information. It should include not only a way to detect them, but also to understand their heritage, evolution and mainly providing structured information which can be used to identify potential security failures at early stages of application development.

Security is a challenging subject, as the attacks change as countermeasures evolve. However, there are unstructured knowledge and much data already available which can leverage the knowledge of the cybersecurity community if they are organized to be shared in a more effective way. Ontology provides the structured and expansible way to organize the sparse and not integrated information about malware. Machine learning algorithms can extract information hidden in the data. Then a multidisciplinary and joined effort of the cybersecurity community is needed to use the tools already in place.

Last but not least, the method described here was successfully applied to the problem of identifying molecules in the human blood serum arising from Zika virus (ZIKV) infection. The article [21] recently published in Frontiers Bioengineering and Biotechnology Journal, with the collaboration of the author of this dissertation, describes the use the method described herein to produce a faster and lower cost screening test for Zika virus detection. Beyond the high accuracy and precision of the diagnosis classifier in detecting the infection, based on high-resolution mass-spectra data, the method interprets which lines of the spectrum are related to patients infected by Zika virus, allowing biochemistry researchers to identify molecules which are biomarkers for the disease. It is an outstanding and promising achievement that proves the generality and applicability of the method in very different areas.

Chapter 4

Conclusion

Results achieved in the article in Section 2.1 grant the conclusion that even data captured from applications' manifests, which are human readable and public available as part of the installation package, contain enough information that a machine learning classifier can decode to predict most common malware families. From another point of view, the method shows that permissions requested by malware are related to their malicious purposes. Thus, an analysis of the permissions used by the applications made available in their manifests was enough to identify several families of malware with accuracy in the average of 88%.

The most common malware families exploit vulnerabilities related to user behavior, such as phishing and social engineering techniques. In those cases, malware does not need to exploit system failures at deeper levels, such as code failures, but only to obtain the user's permission to access sensitive data and data transmission mechanisms, such as phone calls and SMS messages, to succeed in their undertaking. That is the reason for the surprising success to identify most of malware families modeling only the relationships defined by the requested permissions.

Malware that exploits flaws in the lower layers of the system, such as the Linux embedded in Android and infrastructure codes such as OpenSSL, are more difficult to identify by the method presented here because reverse engineering of their code is necessary in such cases. However, the modeling using ontologies can be extended to incorporate other information needed for more extended malware coverage.

As the model for structuring the captured data is based on ontologies, it can evolve to aggregate more in-depth information easily, without restructuring the model and, more importantly, without changing the analysis framework which converts the ontology graph into a machine learning process for determining the most critical structures in the original ontology model.

On the other hand, the article in Section 2.2 approached the accountability needed to use the printer source attribution into forensics reports. That achievement is significant nowadays when machine learning algorithms are escalating on the use of decisions that impact people's lives.

Creating a visual map of the investigated document, highlighting discriminant regions that attach the document to the source printer is a vital support tool for explaining the technical decision to court members. It is a practical and required solution for the immediate application.

More research is needed to improve the detection of some printers which are not able to be identified by their positive features. It is also needed to deal with the open-set problem which addresses the situation when the printer of the investigated document is not in the available pool. However, all these enhancements are subject to future work, and they will also benefit from the methods herein developed.

Finally, both articles have demonstrated the power of the conceptual framework, reducing complex models to a level for which it is possible to create visual representations or individual variable analysis of the problem. Android security and laser printer forensics are completely different arenas, but the conceptual framework was able to approach them in the same way, connecting different computer science tools and techniques, as proven by the last work done in the article of the Zika virus detection from high-resolution mass-spectra[21].

All the work from problem identification, technique research, solution design and conceptual generalization was a tremendous opportunity for learning and applying knowledge acquired during the master's program. Concepts and algorithms of large areas of knowledge such as Ontologies, Graph Theory, Machine Learning, Image Descriptors, to mention some, were studied and applied to the work done. More than the results achieved, the use of these techniques in practice provided the ground where the author could sow the seeds of lessons learned in classes and the literature, and witness this knowledge grow and evolve in the experiments while facing the difficulties of real-world problems. It was a very productive environment for discussions and discoveries.

Bibliography

[1] David W. Aha, Trevor Darrell, Michael Pazzani, Darryn Reid, Claude Sammut, and Peter Stone. Workshop on explainable ai (xai), August 2017. accessed 19.Mar.2018.

[2] André Altmann, Laura Tolosi, Oliver Sander, and Thomas Lengauer. Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10):1340–1347, 2010.

[3] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. Pscout: Analyzing the android permission specification. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 217–228, New York, NY, USA, 2012. ACM.

[4] CVE Board. Cve - common vulnerabilities and exposures - the standard for information security vulnerability names, 2017. Accessed: 2017-08-22.

[5] Leo Breiman. Out-of-bag estimation. Technical report, Statistics Department, University of California, 1996.

[6] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[7] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 161–168, New York, NY, USA, 2006. ACM.

[8] CDT. Digital decisions. Technical report, Center for Democracy & Technology, 2017.

[9] Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends® in Computer Graphics and Vision*, 7(2-3):81–227, 2012.

[10] Nicholas Diakopoulos and Sorelle Friedler. How to hold algorithms accountable. *MIT Technology Review*, 2016.

[11] Nicholas Diakopoulos, Sorelle Friedler, Marcelo Arenas, Solon Barocas, Michael Hay, Bill Howe, H. V. Jagadish, Kris Unsworth, Arnaud Sahuguet, Suresh Venkatasubramanian, Christo Wilson, Cong Yu, and Bendert Zevenbergen. Principles for accountable algorithms and a social impact statement for algorithms. Technical report, FAT/ML Organization, 2017.

[12] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 393–407, Berkeley, CA, USA, 2010. USENIX Association.

[13] FAT/ML. Fairness, accountability, and transparency in machine learning. accessed 19.Mar.2018.

[14] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

[15] Anselmo Ferreira, Luiz Claudio Navarro, Giuliano Pinheiro, Jefersson A. dos Santos, and Anderson Rocha. Laser printer attribution: Exploring new features and beyond. *Forensic Science International*, 247:105–125, March 2016.

[16] Roberto Gallo, Patricia Hongo, Ricardo Dahab, Luiz C. Navarro, Kaio Galv ao, Glauber Junqueira, and Luander Ribeiro. Security and system architecture: Comparison of android customizations. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '15, pages 12:1–12:6, New York, NY, USA, june 2015. ACM.

[17] Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". In Kush R. Varshney Been Kim, Dmitry M. Malioutov, editor, *Proceedings of the 2016 ICML Workshop on Human Interpretability in Machine Learning (WHI 2016)*, pages 26–30, June 2016.

[18] André Grégio, Rodrigo Bonacin, Olga Nabuco, Vitor Monte Afonso, Paulo Lício De Geus, and Mario Jino. Ontology for malware behavior: A core model proposal. In *2014 IEEE 23rd International WETICE Conference*, pages 453–458, June 2014.

[19] André Ricardo Abed Grégio, Vitor Monte Afonso, Dario Fernandes Sim oes Filho, Paulo Lício de Geus, and Mario Jino. Toward a taxonomy of malware behaviors. *The Computer Journal*, 58(10):2758–2777, 2015.

[20] Gilles Louppe, Louis Wehenkel, Antonio Sutera, and Pierre Geurts. Understanding variable importances in forests of randomized trees. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 431–439. Curran Associates, Inc., 2013.

[21] Carlos Fernando O. Melo, Luiz Claudio Navarro, Diogo N. De Oliveira, Tatiane M. Guerreiro, Estela d. Lima, Jeany Delafiori, Mohamed Z. Dabaja, Marta Ribeiro, Maico de Menezes, Rafael Rodrigues, Karen N. Morishita, Cibele Esteves, Aline Amorim, Caroline Aoyagui, Pierina Parise, Guilherme Milanez, Gabriela Mansano, André Ricardo Ribas Freitas, Rodrigo Angerami, Fabio T. Costa, Clarice Arns, Mariangela Resende, Eliana Amaral, Renato Junior, Carolina do Vale, Helaine Milanez, Maria

Moretti, Jose Luiz Proenca-Modena, Sandra Ávila, Anderson Rocha, and Rodrigo R. Catharino. A machine learning application based in random forest for integrating mass spectrometry-based metabolomic data: a simple screening method for patients with zika virus. *Frontiers in Bioengineering and Biotechnology*, 2018.

[22] Kevin P. Murphy. *Adaptive basis function models*, chapter 16, pages 543–587. Adaptive computation and machine learning. The MIT Press, 2012.

[23] Sunil Kumar Muttoo and Shikha Badhani. Android malware detection: state of the art. *International Journal of Information Technology*, 9(1):111–117, Mar 2017.

[24] Luiz C. Navarro, Alexandre K.W. Navarro, Anderson Rocha, and Ricardo Dahab. Connecting the dots: Toward accountable machine-learning printer attribution methods. *Journal of Visual Communication and Image Representation*, 53:257 – 272, 2018.

[25] Van Nguyen. Ontologies and information systems: a literature survey. Technical report, DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION EDINBURGH (AUSTRALIA), 2011.

[26] Thais Mayumi Oshiro, Pedro Santoro Perez, and José Augusto Baranauskas. How many trees in a random forest? In Petra Perner, editor, *Machine Learning and Data Mining in Pattern Recognition*, pages 154–168, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[27] Parliament and Council of the European Union. General data protection regulation.

[28] Andrea Saracino, Daniele Sgandurra, Gianluca Dini, and Fabio Martinelli. Madam: Effective and efficient behavior-based android malware detection and prevention. *IEEE Transactions on Dependable and Secure Computing*, 15(1):83–97, Jan 2018.

[29] Fernanda B. Silva, Rafael de O. Werneck, Siome Goldenstein, Salvatore Tabbone, and Ricardo da S. Torres. Graph-based bag-of-words for classification. *Pattern Recognition*, 74(Supplement C):266–285, 2018.

[30] Fernanda B. Silva, Salvatore Tabbone, and Ricardo da S. Torres. Bog: A new approach for graph matching. In *2014 22nd International Conference on Pattern Recognition*, pages 82–87, Stockholm, Sweden, Aug 2014. ICPR-2014.

[31] Kartik Talamadupula, Shirin Sohrabi, Loizos Michael, and Biplav Srivastava. W11 — human-aware artificial intelligence, February 2017. accessed 19.Mar.2018.

[32] Kabakus Abdullah Talha, Dogru Ibrahim Alper, and Cetin Aydin. Apk auditor: Permission-based android malware detection system. *Digital Investigation*, 13:1–14, 2015.

[33] Kush Varshney, Adrian Weller, Been Kim, and Dmitry Malioutovx. Workshop on human interpretability in machine learning (whi), August 2017. accessed 19.Mar.2018.

[34] D. J. Wu, C. H. Mao, T. E. Wei, H. M. Lee, and K. P. Wu. Droidmat: Android malware detection through manifest and api calls tracing. In *2012 Seventh Asia Joint Conference on Information Security*, pages 62–69, Aug 2012.

[35] Mu Zhang, Yue Duan, Heng Yin, and Zhiruo Zhao. Semantics-aware android malware classification using weighted contextual api dependency graphs. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1105–1116, New York, NY, USA, 2014. ACM.