RADBOUD UNIVERSITY NIJMEGEN

MASTER THESIS

Applying Supervised Learning on Malware Authorship Attribution

Author:
Coen BOOT

Supervisors:
Dr. Ir. Erik POLL
Alexandru C. SERBAN

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Digital Security Group
Institute for Computing and Information Sciences

May 1, 2019

RADBOUD UNIVERSITY NIJMEGEN

Abstract

Faculty of Science
Institute for Computing and Information Sciences

Master of Science

Applying Supervised Learning on Malware Authorship Attribution

by Coen BOOT

Malware is a problem in current digital society, since it can cause economic or physical damage and in the end disrupt society as a whole. In order to effectively fight cyber threats by coming up with (legal) consequences for the actor behind the malware, it is important to be able to provide a certain degree of proof about who is responsible for the malware. The process of linking an author to an asset is called authorship attribution. In case of malware, attribution needs to be based on binary executables, since the source code is mostly unavailable.

This thesis focusses on evaluating and comparing two promising approaches for performing authorship attribution on malware. These approaches are based on two supervised learning algorithms, namely a neural network and a random forest classifier. Both approaches use automatically generated analysis reports from a sandbox solution as input data.

Malware can be divided in two types with respect to the actors behind it: state-sponsored or criminal. This thesis focusses on the first type, since state-sponsored malware has a richer and clearer hierarchy of authorship (i.e. country-level and APT group-level) compared to criminal malware which is often attributed to a group of individuals which is not explicitly related to a nation-state.

Since no suitable dataset containing state-sponsored malware is available yet, we collected a dataset using a newly devised method, based on indicators of compromise found in threat intelligence reports. In this way we collected a dataset with 3,594 state-sponsored malware samples, which forms the first publicly available dataset of its kind.

Using the retrieved malware samples, we used 2 sandboxes to generate reports about the samples: Cuckoo and VMRay. Moreover, we downloaded the reports belonging to the samples from VirusTotal as well. All these reports were converted to a bag of words, after which they are used as input for the classification algorithms.

We evaluated the two approaches on the dataset and found that both approaches perform well (with accuracy results up to 98.8%) and match the performance described in the original papers. The neural network-based approach tends to perform slightly better compared to the approach based on a random forest classifier, whereas the latter uses considerably less time to finish training.

A trained classification algorithms contains knowledge about the characteristics of the different classes, since it needs to decide to what class an unseen sample belongs. We extracted this information, attempting to discover new insights and verify whether the classifier makes sensible decisions.

Contents

Abstract **iii**

1 Introduction **1**
1.1 Research Questions . . . 2
1.2 Structure of this Thesis . . . 2

2 Background and Related Work **5**
2.1 Legal Goals of Attribution . . . 5
2.1.1 Distinctions in Types of Attribution . . . 6
2.1.2 State-sponsored or not? . . . 6
2.1.3 Legal Goals . . . 7
2.2 Complications Inherent to Malware . . . 7
2.2.1 Unavailability of Source Code . . . 7
2.2.2 Hiding Intents . . . 8
Static Methods . . . 8
Dynamic Methods . . . 8
2.2.3 Fake Traces . . . 9
2.3 Technical Means for Authorship Attribution and Family Classification 9
2.3.1 Approaches Focused on Family Classification . . . 10
2.3.2 Approaches Focused on Authorship Attribution . . . 11
2.3.3 Approaches Using Machine Learning . . . 12
2.3.4 Selected Approaches . . . 12

3 Used Malware Classification Techniques **15**
3.1 Classification Algorithms . . . 15
3.1.1 Supervised Learning . . . 15
Methodology . . . 15
Overfitting and Underfitting . . . 16
3.1.2 Deep Artificial Neural Network . . . 16
Structure . . . 16
Basic Principles . . . 16
Peculiarities and Parameters . . . 17
3.1.3 Random Forest Classifier . . . 18
Structure . . . 18
Basic Principles . . . 18
Peculiarities and Parameters . . . 19
3.2 Classification Performance Metrics . . . 19
3.2.1 Recall / True Positive Rate (TPR) . . . 20
3.2.2 False Positive Rate (FPR) . . . 20
3.2.3 Precision . . . 20
3.2.4 F1-Score . . . 21
3.2.5 ROC-Curve . . . 21
3.2.6 Accuracy . . . 21

4 Collecting a Dataset of State-Sponsored Malware **23**
4.1 Collection Method . . . 23
4.1.1 Collecting Samples . . . 23
4.1.2 Collecting Sandbox Reports . . . 24
4.2 Data Preprocessing . . . 24
4.2.1 Duplicate Detection . . . 25
4.2.2 Filtering Sandbox Reports . . . 25
4.2.3 Extracting API Calls . . . 25
4.2.4 Creating a Bag of Words . . . 26
4.3 Overview of the Collected Dataset . . . 26
4.4 Dealing with Imbalanced Datasets . . . 26

5 Experimental Data & Setup **29**
5.1 Constructing Training and Test Sets . . . 29
5.2 Tested Scenarios . . . 30
5.3 Sandboxes Used . . . 30
5.3.1 Cuckoo . . . 30
5.3.2 VirusTotal . . . 31
5.3.3 VMRay . . . 31
5.4 Overview of Training and Test Sets . . . 32
5.4.1 Used Variants . . . 32
5.4.2 k-Fold Cross-Validation . . . 33

6 Evaluation of the Neural Network-based Approach (Ro18) **35**
6.1 Configuration of the Neural Network . . . 35
6.2 Results . . . 35
6.2.1 Country-Level Authorship Attribution with Unseen APT Groups 36
6.2.2 Country-Level Authorship Attribution with Earlier Seen APT Groups . . . 37
6.2.3 APT Group-Level Authorship Attribution . . . 38
6.3 Preliminary Conclusions . . . 40

7 Evaluation of the RFC-based Approach (Am17) **41**
7.1 Configuration of the Random Forest Classifier . . . 41
7.2 Results . . . 41
7.2.1 Country-Level Authorship Attribution with Unseen APT Groups 41
7.2.2 Country-Level Authorship Attribution with Earlier Seen APT Groups . . . 42
7.2.3 APT Group-Level Authorship Attribution . . . 43
7.3 Preliminary Conclusions . . . 45

8 Comparing the Two Approaches **47**
8.1 Comparison with Respect to Algorithm . . . 47
8.2 Comparison with Respect to Sampling . . . 47
8.3 Comparison with Respect to Metrics . . . 48
8.4 Comparison with Respect to Sandboxes . . . 48

9 Extracting Human-Interpretable Characteristics **49**
9.1 Knowledge Extraction on Random Forest Classifiers . . . 49
9.1.1 Feature Importance . . . 50
9.1.2 Manual Decision Tree Analysis . . . 50
9.2 Knowledge Extraction on Neural Networks . . . 52

10 Complications Faced and Discussion **53**
10.1 Complications Faced 53
10.2 Discussion 53

11 Conclusions and Future Work **55**
11.1 Conclusions 55
11.2 Future Work 56

Bibliography **59**

A Additional Results of the Neural Network-based Approach **67**
A.1 Country-Level Authorship Attribution with Unseen APT Groups . . . 67
A.2 Country-Level Authorship Attribution with Earlier Seen APT Groups 68
A.3 APT Group-Level Authorship Attribution 68

B Additional Results of the RFC-based Approach **71**
B.1 Country-Level Authorship Attribution with Unseen APT Groups . . . 71
B.2 Country-Level Authorship Attribution with Earlier Seen APT Groups 72
B.3 APT Group-Level Authorship Attribution 72

C Most Important Features in a Trained RFC **75**

D Extracted Decision Tree **77**

Chapter 1

Introduction

Malware is a problem in current digital society, since it can cause economic or physical damage and in the end disrupt society as a whole. Although anti-virus solutions attempt to withstand the massive flows of malicious software, they fail in blocking every single piece of malware. This is caused by the fact that they often look for exact fingerprints, without being able to recognize characteristics similar to known malware. This makes it possible to create malware which is able to successfully infect a machine and perform its (illicit) tasks. Malware is not only used by cyber criminals. State-sponsored *Advanced Persistent Threat* (APT) groups make use of malware as well to spy on their targets, contributing to an even more tumultuous cyber landscape.

An important step in successfully fighting the threat of malware infections is the technical analysis of malware. Once one is able to see and understand in detail how a malware sample works, new techniques for detecting and stopping that sample can be designed using the obtained analysis.

However, technical analysis is not the only possible counteraction. In order to effectively fight cyber threats by coming up with (legal) consequences for the actor behind the malware, it is important to be able to provide a certain degree of proof about who is responsible for the malware. The process of acquiring this evidence is called *authorship attribution* (or shortly attribution). As shown in the literature study described in Chapter 2, much progress has been made in this field, but many of developed techniques make use of the source code of the software. Since the source code of malware is unavailable for the vast majority of malware samples, author attribution needs to be based on features of a binary executable. Moreover, a lot of effort is put by malware authors into ways of preventing any form of attribution, which makes it even more difficult to track down the author.

Not only is the relation between a malware sample and its author an interesting link, links between various samples based on shared techniques, libraries, purposes or targets are useful too. Since malware samples from the same malware family share a lot of properties and are often created by the same author, an approach for detecting one sample may be useful for detecting another sample from the same family as well. Finding proper and efficient ways to perform *family classification* can therefore be helpful for performing authorship attribution.

Both authorship attribution and family classification are *classification problems* in the machine learning community: problems in which a collection of items needs to be grouped in predefined classes as accurately as possible. Any algorithm that is able to solve such problem is called a *classification algorithm*.

Supervised learning is a methodology which uses machine learning algorithms to learn from labeled data. The use of modern supervised learning techniques like

deep neural networks for classification has great potential, but is yet not fully explored; only a few publications describing malware classification with the use of supervised learning were encountered during the literature study. One of the biggest advantages of such techniques is the fact that they need no human input apart from a labeled dataset, thus saving a lot of time-consuming analysis labor. Moreover, they provide high level results, based on patterns found in the dataset. Therefore, supervised learning techniques are able to find relationships between malware samples which are hard to discover for humans [1–3].

This thesis will focus on evaluating and comparing supervised learning approaches for performing authorship attribution, guided by the methods **Ro18** and **Am17** as described in [1] and [2] respectively. Moreover, we will extract malware characteristics from trained classifiers in order to evaluate the trustworthiness of the classifiers and to see if new characteristics of authors can be discovered.

1.1 Research Questions

The main research question for this Master Thesis is as follows:

"To what extent is it possible to perform state-sponsored malware authorship attribution using supervised learning?"

This question is divided in the following research questions:

1. Which supervised learning techniques are optimal for performing malware authorship attribution?
2. What type of dataset is needed for reaching satisfying attribution performance?
3. To what extent can intelligible author characteristics be derived from a trained supervised learning algorithm?

The choice for state-sponsored malware is made because of the fact that it has a more interesting hierarchy in terms of authorship compared to 'regular' malware (malware which is not related to any nation-state actor). This difference in hierarchy consists for example of the fact that a single nation-state actor often has multiple APT groups (again consisting of multiple developers) to its disposal for writing malware, whereas 'regular' malware often is produced by independent groups. Therefore, the amount of information with respect to authorship that is available for any known malware family differs as well; in case of 'regular' malware, only the name of the group behind the malware may be known, whereas in case of state-sponsored malware, the responsible nation-state actor and APT group are often available [4].

1.2 Structure of this Thesis

This remaining chapters of this thesis are structured as follows: Chapter 2 describes the related work about the current state of research into authorship attribution and family classification. Moreover, this chapter provides an overview of the most important approaches available for performing authorship attribution and family classification. Although this thesis focuses on authorship attribution, approaches focusing on family classification are considered as well, since we suspect both problems

to be closely related to each other. Two of the described approaches, **Ro18** (based on a neural network) and **Am17** (based on a random forest classifier), will form the basis for this thesis. Chapter 3 explains the use of supervised learning techniques and metrics to evaluate classifier performance, followed by a description of the collected dataset and the way is has been constructed out of publicly available sources in Chapter 4. A description of the setup of the experiments performed for testing the performance of both approaches is described in Chapter 5. Chapters 6 and 7 describe the evaluated performance of **Ro18** and **Am17** respectively, both providing some preliminary conclusions as well. The performance of both approaches are compared in Chapter 8, after which the process and results of extracting human-interpretable characteristics from trained classifiers is described in Chapter 9. The thesis ends with a discussion about the achieved results and complications faced in Chapter 10 and the final conclusions and description of future work in Chapter 11.

Chapter 2

Background and Related Work

In order to effectively fight cyber threats by coming up with (legal) consequences for the actor behind the malware, it is important to be able to provide a certain degree of proof about who is responsible for the malware. The process of acquiring this evidence is called *authorship attribution* (or shortly *attribution*). Due to the complexity of the problem, still no silver bullet is provided by academia to solve this problem [5, 6].

Authorship attribution may be seen as a specific *malware classification problem*, since it seeks to classify malware samples by author. Other common malware classification problems include *malware detection*, where samples are classified based on whether they are benign or malicious, and *family classification*, where samples are classified based on the malware family they belong to.

This chapter describes the current state of research into attribution and family classification, focusing firstly on the question about the legal goals of authorship attribution in Section 2.1. After that, an overview of technical, malware inherent complications hindering attribution and family classification is provided in Section 2.2, followed by a summary of the technical means to solve classification problems in Section 2.3.

One of the conclusions that can be drawn from this chapter, is that the problem of (state-sponsored) malware classification, and especially authorship attribution on binaries, is not covered extensively by academia. This makes the the overview given in this chapter fairly comprehensive.

From all literature available on this topic, [1, 2, 7, 8] form the main source of inspiration for this thesis, since those publications contain relevant considerations and the most recent developments in the field of (state-sponsored) malware attribution and provide pointers to multiple paths for new research.

2.1 Legal Goals of Attribution

In the first place, malware attribution is considered to be not trivial at all. While any programmer leaves traces of authorship, most traces can easily be faked on purpose as well. Therefore, statements of attribution must be perceived as substantiated evidence and not as facts. Since it is important to know to what extent authorship attribution can be applied, several research projects were set up to investigate real-world problems related to attribution, such as feasibility and the legal complications of authorship attribution of malware. The conclusions of this research will be summarized and discussed in this section.

2.1.1 Distinctions in Types of Attribution

Shamsi et al. [6] justly points out the difference between *technical* and *human attribution*. The goal of *technical attribution* is to identify which machine is being used to create or launch the attack and to provide proofs which back up such a claim. However, in order to start legal prosecution, the attack needs to be linked to one or more individuals, which is done in the process of *human attribution*. This process is in many cases more extensive, since it often requires technical attribution first.

Another distinction that must be made is between *authorship attribution* and *authorship discrimination*. Whereas authorship attribution tends to find the author given a certain asset, authorship discrimination tries to establish proof that two given assets where created by the same author [9]. This distinction is important to keep in mind, while the first problem seeks to find an author (*who is the author of X?*), based on prior knowledge such as the authors former work, whereas the latter accepts or rejects a given hypothesis of shared authorship (*are Y and Z written by the same author?*).

This thesis focuses on human attribution and authorship attribution. Therefore, the classifiers that are used are trained to learn from the characteristics of an APT group or nation-state and used to determine to what actor a given sample belongs.

2.1.2 State-sponsored or not?

Imagine a case where criminals have been trying to hack a financial institution, but that they were caught in the act. In order to get the criminals convicted, the members of court need to be convinced that the evidence is valid beyond reasonable doubt [10]. However, things get different when one nation-state actor tries to hack into the systems of another nation-state. Consensus exist that a lower level of proof is required before taking further steps in cases regarding nation security compared to civil cases [11].

This difference is due to consequences that are related to attribution: conclusions about the responsible actors of nation-state attacks are being politicized, whereas criminal attacks need to pass the domestic court. Moreover, in the case of attacks on nation-state level, the process of attribution seeks to establish proof about the *sponsors* of the attack, whereas criminal cases seek to come up with burden of proof related to the *launcher* of the attack.

Two conceptual attribution workflows could be drafted, considering the distinction between criminal and state-sponsored cases. The workflow for criminal cases involves discovering the attack, finding the attacker, seizing evidence and bringing this to court, hoping to be able to provide 'proof beyond reasonable doubt'. However, when the attacker is located in a remote country, that remote country needs to be willingly to cooperate, by e.g. supplying the match between IP-address and name.

This cooperation does not exist when considering attacks on nation-state level. Therefore, the workflow for cases involving sponsorship by a nation-state consists of discovering the attack, finding evidence, making an assessment and acting upon this assessment. In this workflow, political judgments play a role in the process of attribution, implying that attribution could become a matter of 'beliefs and reasons' instead of 'proof beyond reasonable doubt'. It is the task of intelligence services to make assessments which will lead to those judgments, since they are the most appropriate organization, having both political and technical expertise in place to perform proper attribution [10].

Getting to know whether an attack really is sponsored or launched by a nation-state requires a thorough approach. The popular belief that sophistication is the main indicator for making this decision, is refuted by Guitton and Korzak, who note that the term 'sophistication' is used inconsistently [12]. Moreover, the implication that sophistication does involve sponsorship by a nation-state (and vice versa) is rejected as well.

When the outcome of an investigation suggest that an attack is state-sponsored, a nuanced judgment is still needed. This is due to the fact that the involved nation-state could be held responsible in several degrees, ranging from minimal responsibility (because the act of attacking has explicitly been prohibited by the government) up to full responsibility (because the government actively has been working on the attack) [12, 13].

2.1.3 Legal Goals

Given the two conceptual workflows drafted above, we may conclude that two legal goals exist for malware authorship attribution:

1. Get evidence 'beyond reasonable doubt' to bring a case to court and prosecute a criminal (organization).
2. Get evidence as a matter of well-substantiated 'beliefs and reasons' to use as input for political judgments regarding a case where nation-states are involved.

Since it is likely that most cases fall either in a criminal context or in a national security context, a combination of the mentioned legal goals will seldom occur. Since the first goal requires human attribution on the level of one or a few individuals in the end, it is hardly feasible to develop an algorithm which provides sufficient evidence to meet the required standards. In case of the latter legal goal, several possibilities loom up while there are several means to automatically come up with substantiated pointers to responsible nation-state actors or APT groups. Several means to come up with those substantiated pointers will be explored in this thesis.

It must be noted that such substantiated pointers are not only useful for nation-states which are under attack, but that e.g. cyber security companies try to gain this type of information as well, in order to deepen their understanding of malware samples and campaigns.

2.2 Complications Inherent to Malware

Working with malware poses inherent complications to analysis [8]. Since malware is designed to be as undetectable and untraceable as possible, a lot of effort is put into hiding malicious goals and procedures and adding fake procedures and traces to trick both human analysts and automated analyzers. Because of this, the application of detection, classification and attribution techniques is strongly impeded. This section provides an overview of three such complications.

2.2.1 Unavailability of Source Code

First of all, most authorship attribution approaches are developed to determine authorship solely based on stylistic features of source code (see for example [14]). However, regarding the case of malware, the unavailability of source code in the far majority of cases throws a spanner in the works for this approach. Therefore, only techniques using binaries for analysis can be deployed [8].

2.2.2 Hiding Intents

Secondly, malware is often processed in order to hide its intents, using all sorts of techniques such as obfuscation, encryption and analysis evading techniques. Therefore, approaches that are developed for authorship attribution of benign binaries do not always perform well when analyzing malware, simply due to the fact that they are not designed to handle obstacles caused by intent-hiding techniques [5, 8]. The techniques for hiding intents can be divided into static (happening before and during compilation) and dynamic methods (happening at runtime).

Static Methods

Static techniques for hiding intents focus on making a compiled binary has hard to understand as possible. The most important techniques to do so include [15]:

- **Packers:** The most common static way for malware to circumvent the anti-virus (AV) signature checks is by using so-called packers. Such packers compress binaries into a smaller and sometimes encrypted asset and add a small part of code to extract the compressed asset at runtime. Although packers are often used for malicious intents, they are also helpful for benign purposes such as protecting software copyrights and performing binary compression to save diskspace or bandwidth. Therefore, not all packed software can simply be considered as being malicious.

  The problem that packers introduce is the fact that automatic analysis of software without running it first becomes a very complicated and nearly infeasible task. Moreover, several advanced forms of encryption methods, such as *polymorphism* and *metamorphism* are used, increasing the complexity of detecting malware due to the fact that the malware mutates every time it is executed. This causes signature-based approaches to fail in fighting malware effectively, but also opens new ways to detect malware by observing e.g. side channels such as CPU workload [16].

  One way of dealing with packed malware is described in [17], where an approach is proposed which uses dynamic analysis for unpacking malware and static analysis based on flow graph matching for malware family classification.

- **Code Virtualization:** Another intent-hiding static method is *code virtualization*. This technique sets up a virtual machine (VM) with a custom, unfamiliar instruction set and translates the malicious code to that instruction set. The VM and the translated code are included in the malware. Once the malware gets executed, it initializes the VM and runs the translated code on it. The main problem that this approach introduces, is the fact that in order to reverse the malware to retrieve the original code, one needs to understand the VM interpreter, the translation process and the translated code. This process can get even harder when multiple VM's and a broad variety of instruction sets are used [18].

Dynamic Methods

Apart from the obfuscation techniques that make the functionality of a binary hard to understand, several tricks are used to check at runtime whether the malware is being analyzed. When this would be the case, the malware could change its behavior

or terminate, hindering analysis in this way. These tricks can roughly be categorized as follows (note that most tricks are specific to Windows) [19]:

- **Sandbox Evasion:** Sandboxes are often used for malware analysis, since they are a good way of looking into a sample's behavior, by executing it [20]. Although the sandbox gets infected by doing so, the infection can easily be undone, since sandboxes are designed to be revertible to an earlier state. To impede the use of sandboxes for analysis, a sample could observe certain artifacts which provide clues about the use of sandboxes, such as the fact whether the MAC address of the infected machine is known to be a MAC address of a virtual machine, or the presence of certain processes, such as `VmwareService.exe`.
- **Debugging Evasion:** Another type of evasion is debugging evasion. A piece of malware can easily query the Windows API to check whether a debugger is active or not. Also, the activity of known debuggers, such as OllyDbg, can easily be noticed by requesting the titles of currently opened windows.
- **Anti-virus Evasion:** In order to evade anti-virus solutions, malware could perform several tricks at runtime, such as creating enormous files in order to crash a file scanner or trying to disable anti-virus tools. Although numerous techniques are around, they are not covered here, since the analysis done in this research will not be impeded by these kind of evasion techniques.

Although all intent-hiding methods seek to make analysis harder, they sometimes provide additional information about the author of the malware or the relation to other malicious software (e.g. by using similar techniques, having shared *XOR*-keys for string obfuscation or shared hashes for hiding API calls) [7].

2.2.3 Fake Traces

Thirdly, authors of malware try to trick researchers in drawing faulty conclusions by adding fake traces to other authors or malware families. These traces can include timestamps from other timezones, adding text in different languages, using deceiving name-giving or reusing exploits from different actors [21]. Therefore, it is important to check whether clues of authorship are genuine or fake.

2.3 Technical Means for Authorship Attribution and Family Classification

Although the problems as described in Section 2.2 are comprehensive, several techniques exist to make attribution (partially) possible. This section provides an overview of binary attribution and classification techniques. First, techniques which classify malware in general are given in Section 2.3.1, since more advanced techniques often build on those principal ideas (e.g. the ideas proposed in [22] build on techniques described in [23] when it comes to using API calls for analysis). The more advanced techniques then, which are especially designed for authorship attribution, are discussed in Section 2.3.2, followed by the application of machine learning for performing attribution and family classification in Section 2.3.3.

2.3.1 Approaches Focused on Family Classification

Approaches for classifying software (hence also malware) based on code-similarities and code-reuse are well-described in literature, and focus mostly on:

- **Metadata**: One of the easiest ways to compare malware is to compare the metadata of samples. Many forms of metadata are easy to extract, such as used strings, hashes, PE headers, compilation time, used packers etc. [24]. The presence or absence of overlap in several parts of the gathered metadata provide clues whether two samples are related.

- **Internal Functions**: Malware, just like any other piece of software, consists of a collection of functions. When analyzing the functions inside a malware sample, one can classify the sample, since research has proven that function (call) frequency and function length can be used as measures for classification [25, 26].

- **API Calls**: Instead of only looking to functions inside a binary, functions outside the binary that are called by the binary can be regarded as well. A specific example of such external function calls are API calls. API calls are requests, made by a piece of software to the operating system (OS) in order to execute tasks, like `MoveFile` or `GetFileInformation`. When analyzing the calls of a sample, one can get a rough idea of what such a program tries to achieve [23]. Moreover, such calls can be used for establishing malware signatures and detecting malware families based on the achieved data [27–30].

- **Code Blocks**: Malware could easily be classified as well by looking at blocks of binary code that samples share. Whenever pieces of malware share one or more code blocks, chances are that those samples are related [31]. Using the technique proposed in [32], signatures suitable for comparing code blocks can be generated, making this process more efficient compared to a more naive approach. Also machine learning can be applied to make such an approach work [33].

- **Control Flow Graphs**: The structure of a piece of software can be visualized using a *control flow graph*, a graph visualizing all possible paths that the execution of the program could traverse. A special variant of such control flow graph is the *call graph*, where the nodes represent the routines of a program. These kind of graphs enrich the previous two approaches (API Calls and Code Blocks) and are very suitable for the analysis and classification of malware. Nevertheless, other types, such as a *data flow diagram* could be used as well [34–39].

- **Visualization and Image Processing**: A totally different approach can be taken by converting a binary file to an image. This can basically be done by converting binary code to a 8-bit vector, and using this vector as a pixel in a grayscale image. However, more sophisticated approaches exist, such as the approach proposed in [40], which generates an image based on the *opcodes* in a sample and uses similarity hashing to calculate the similarity between two images, and thus between two malware samples. The resulting image shows the structure of a binary file surprisingly well. On these images, the whole range of image processing techniques can be deployed, making it possible to classify

malware [40–43]. 'Surprisingly, this approach also seems to be resilient to contemporary packing strategies and can robustly classify large corpus of malware with both packed and unpacked samples' [42].

- **Behavioral Fingerprints** Instead of statically looking at malware samples, it is also possible to execute a sample and watch its behavior closely. Since it may be assumed that malware samples from a single family behave quite similar, it is possible to classify malware based on behavior [44].

- **Constructing Family Genetics:** In the field of biology, many studies have been made about the matter of genetics. The principles of genetics can be applied to malware family classification as well. When taking multiple features into account, such as compilation time and malware family indicators, one is able to link samples of a malware family to each other, put them in chronological order and detect inheritance. This type of method has for example been applied in combination with machine learning in the Malware Analysis and Attribution using Genetic Information (MAAGI) system, a major component in the US sponsored Cyber Genome Project [22, 45].

- **Combining Technical Features:** A well-chosen combination of technical features, such as the ones mentioned above, can be used to link similar malware to each other. An excellent example of this is the method as described in [46], where 13 different types of features are combined to form input for classification algorithm.

Looking at the means mentioned above, it must be noted that they tend to classify software or malware in general, but that no explicit attention is given to properties that indicate possible shared authorship. Therefore, such features cannot be used for direct authorship attribution, since they might contain no information about the author at all [8]. Because of this, they must be used with great caution, or be modified to suit the goal of authorship attribution.

2.3.2 Approaches Focused on Authorship Attribution

Apart from general approaches to classify software, several specific techniques exist, which do focus on the specific problem of authorship attribution. As pointed out in Section 2.2.1, approaches which use source code are out of scope, since the source code is not available for the far majority of malware samples. When only considering binaries, the techniques described in current literature can be grouped as follows:

- **Stylistic Features Surviving Compilation:** Although we assume the availability of source code to be out of scope, some of the principles of extracting stylistic features from the source code can still be applied to binary malware samples, as long as these features survive the phase of compilation. Examples of stylistic features that survive compilation are branch constructs and used integer types. Moreover, flow graphs, as described in the previous section, provide significant data as well [5, 47, 48]. However, when a piece of malware is written by a team or includes external libraries (thus involving multiple authors), these stylistic features become weaker [7].

- **Combining Technical Features:** The benefit of performing authorship attribution on malware is that malware, unlike written text, also contains technical

features. Examples of technical features include timestamp formatting, multi-threading model, programming language, compiler, (shared) encryption keys or passwords, data exfiltration techniques, code reuse, shared exploits or the preference in use of certain IP addresses, email services, C&C-servers or the features mentioned in Section 2.3.1. In case of authorship attribution however, features need to be selected with care, because not all imaginable features are related to authorship of software, but may be more related to e.g. the purpose of the software [7, 8, 21].

Although these methods do work on software in general, they are not always specifically designed for analyzing malware. Due to the complications that malware entails, as described in Section 2.2, those features may not always be suitable when working with (complicated) malware.

2.3.3 Approaches Using Machine Learning

Recent developments include machine learning approaches for the detection and categorization of malware, using supervised as well as unsupervised machine learning methods. These developments also include techniques that can be used to make machine learning algorithms perform better when working with malware. For example, similarity hashing can be used as an technique to train an algorithm for detecting semantically similar files [49].

- **Based on Binary Features:** Examples of applying machine learning to features like those mentioned in Sections 2.3.1 and 2.3.2, are described in [28], [29], [30], [33], [43], [45] and [48]. Most of them perform malware detection or classification; OBA2, the only approach which focuses on authorship attribution uses techniques which are not designed for and tested on obfuscated binaries, making the proposed method less useful for malware analysis [48]. Also approaches using machine learning based on other techniques than those mentioned above exist, such as an approach by Annachhatre et al., which generates a hidden Markov model based on extracted *operation codes* and uses a k-means clustering algorithm to classify malware [50].

- **Based on Sandbox Reports:** Since most mentioned machine learning approaches only use static features for malware analysis, they do not have information about the behavior of (packed or obfuscated) malware samples. This behavioral information can be extracted by skilled analysts, making large-scale analysis a very costly process. This is where sandboxes come in helpful, because sandboxes are able to extract numerous malware characteristics in a short time [20], including information about the behavior of a sample. The approaches known to us at the moment of writing are described in [1], [2] and [3].

2.3.4 Selected Approaches

It is worth examining to what extent supervised learning is able to classify malware based on authorship. In order to find this out, we selected known approaches based on sandbox reports. Approaches based on sandbox reports look the most promising to us, since sandbox reports should provide a good overview about numerous characteristics of a sample, without requiring human effort or additional tools.

From the 3 approaches available, we left out the approach described in [3], since this approach is less recent than, partially written by the same author as, and similar

to **Ro18**. In this way, we selected **Ro18** and **Am17**, and evaluated and compared their performance on a single dataset.

- **Ro18** is described in [1], a scientific paper published by a company that sells deep learning anti-virus solutions. This paper illustrates how machine learning can be applied in a simple way to analyze state-sponsored samples. Automated sandboxes like Cuckoo Sandbox [51] provide the possibility to upload a sample and retrieve a report with the outcome of the analysis by the sandbox. After performing some minor processing to this Cuckoo report, the modified report is provided as input to a deep neural network [1]. Most principles of this paper by Rosenberg et al. seem to be based on [3], a paper describing earlier research of one of the authors, in which exactly the same method for preprocessing the data is proposed. Furthermore, the papers overlap in terms of the neural network architecture that is used.

- **Am17** is an approach similar to **Ro18**, in which reports from sandboxes are used as input for machine learning as well. However, in contrast to **Ro18**, a Random Forest Classifier is used and the input data is more tailored to the algorithm [2].

  Moreover, the classifier is trained to solve family classification on criminal malware instead of authorship attribution on state-sponsored malware. Although we take into account that these differences may lead to a worse classification performance, we do not expect major incompatibilities using **Am17** to solve authorship attribution, since we suspect both problems to be closely related to each other.

Using these approaches, the problem of authorship attribution seems to be dealt with in an effective way. Both methods are easy to implement and require minimal human effort, yet having high accuracy scores reported. Since we selected **Ro18** and **Am17** as approaches to evaluate and compare, the following chapter will describe the fundamentals of the techniques used in these approaches.

Chapter 3

Used Malware Classification Techniques

We will use two classification algorithms and different metrics to evaluate the performance of the trained classification algorithms. These used techniques are described in this chapter.

3.1 Classification Algorithms

Ro18 and **Am17**, the two approaches that we are going to evaluate and compare, use different classification algorithms. **Ro18** uses a *deep artificial neural network* to classify malware samples, whereas **Am17** makes use of a *random forest classifier* (RFC). Since we evaluate and compare both approaches, we will work with the same classification algorithms as well. A brief description of both algorithms will be given, as well as a general introduction to supervised learning.

3.1.1 Supervised Learning

Artificial neural networks and random forest classifiers are both used in this context for supervised learning. Supervised learning is a methodology which uses machine learning algorithms to learn from labeled data. Classification problems are one kind of problems that can be solved using supervised learning.

Methodology

The process of performing supervised learning consists of 3 phases: preparing a labeled dataset, training a supervised learning algorithm and testing the trained algorithm. The fact that the dataset needs to be labeled, means that every sample in the dataset needs to be accompanied with the class it needs to be classified in.

The dataset is then split into a training and a test set, which can happen in various ways. One commonly used and simple approach is called *holdout*, which involves keeping back a part of the dataset to use as a test set (often around $1/3^{th}$ of the full set) [52, 53]. Another way is by using *k-fold cross-validation*, a method in which the dataset is split in k parts ($k = 5$ is proven to be a good starting point [54]) and the algorithm is ran k times, where each time a single different part of the dataset serves as test set. After k runs, the average performance is calculated.

Following the splitting strategy chosen, the parts intended for training are provided with the corresponding labels to the algorithm, and the algorithm starts to look for relations between the properties of the given samples and the labeled classes to which the samples belong to.

When training has finished, a well-trained algorithm should be able to correctly classify any presented sample. In order to test to what extent the algorithm is well-trained, the algorithm is executed on the part of the dataset intended for testing. The predicted outcome of the algorithm is compared to the labels belonging to the test set, and the performance of the algorithm is calculated [55].

Overfitting and Underfitting

It is not uncommon that a trained classification algorithm does not perform well on classifying samples in the test set. In many cases, this is due to the fact the the algorithm is either overfitted or underfitted.

When the algorithm is overfitted, it has constructed a model that is over-adjusted to the training data. The negative consequence of this is the fact that noise in the training set is treated as useful information, making the model way too specific. When the test set is provided to the algorithm, the algorithm makes decisions based on the noise in the test set, but since noise is random and meaningless, the results are less accurate. Overfitting can be recognized by a high training, but low testing performance [56].

The opposite of overfitting is underfitting. When a model is underfitted, it has not been able to capture details in the training set that indicate to which class a sample belongs. Therefore, the model does not perform well on classifying the test set. Underfitting can be recognized by a low training and testing performance [57].

3.1.2 Deep Artificial Neural Network

Structure

Artificial neural networks are acyclic graphs that try to simulate the human brain. They consist out of so-called *neurons*, which are nodes that are capable of storing a value. These neurons are to some extent interconnected via edges which hold a certain weight w, thus forming a network.

Each neural networks is structured into several layers. It has an input and an output layer, which consist of m and n nodes respectively, where m is the number of features a sample has, and n the number of classes that the data must be grouped in. Between the input and output layer are a number of hidden layers, which may vary in size. A general model of a neural network is shown in Figure 3.1. When a network has multiple hidden layers, it may be qualified as a *deep artificial neural network* [58, 59].

Basic Principles

A single neuron i, located in any non-input layer of a trained neural network, behaves as follows: the vector of values of the previous layer, which we will call $\boldsymbol{x}$, is passed on through the edges, and multiplied by the weights $\boldsymbol{w_i}$ that the node assigns to the edges. The bias b_i of the node is added to the outcome of the multiplication, resulting in $a_i = \boldsymbol{w_i}^\intercal \cdot \boldsymbol{x} + b_i$. An *activation function* f, which maps the input to a fixed range, is then applied to a_i, resulting in a final value of $y_i = f(\boldsymbol{w_i}^\intercal \cdot \boldsymbol{x} + b_i)$. A schematic overview of this calculation is shown in Figure 3.1.

This calculation is performed simultaneously for an entire layer, by using the full matrix $\boldsymbol{W}$ and the full vector $\boldsymbol{b}$. The calculation for a complete layer then looks as follows:

$$\boldsymbol{y} = f(\boldsymbol{W}^\intercal \cdot \boldsymbol{x} + \boldsymbol{b})$$

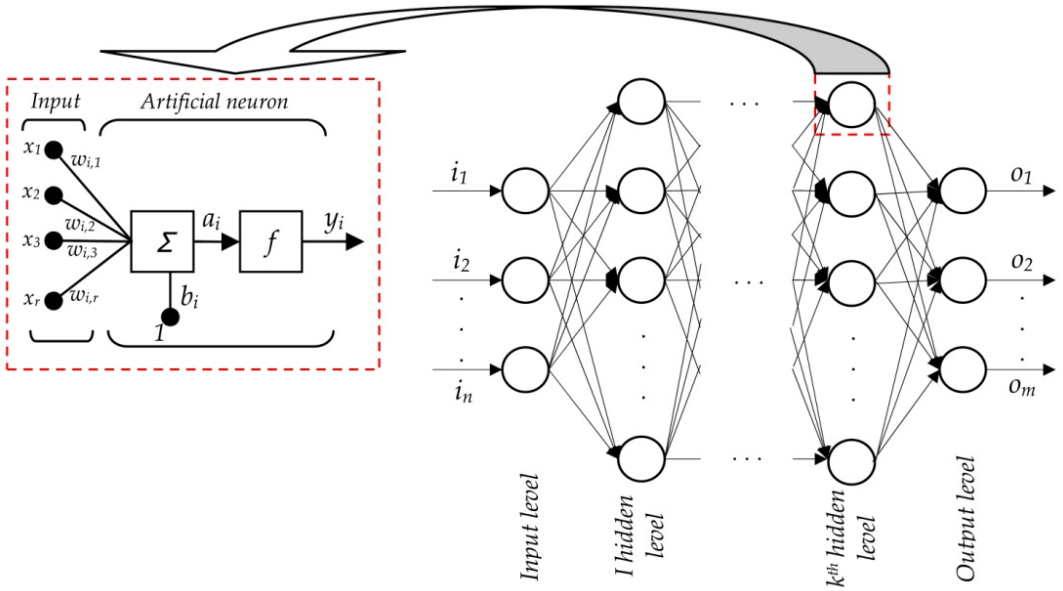

FIGURE 3.1: Schematic overview of a neural network and the propagation function of a neuron [60].

where the activation function f is applied elementwise to the vector. All neurons in the network perform this calculation, layer after layer, starting at the first hidden layer. This process is called *forward propagation*.

Because of this, the value of every node in the next layer is influenced by a well-chosen mix of nodes in the previous layer. Since nodes in the previous layer contain low level features, they can serve as building blocks for every node in the next layer, making the next layer able to capture higher level features. This process repeats until the output layer, where a final class is chosen for every sample.

During training, several *batches* are created from the training samples. The network processes the training set batch-by-batch and compares its output to the desired output. When the output differs from the desired output, the difference in values is propagated back through the network and the weights of the edges are adjusted in order to minimize the *loss* (i.e. the outcome of a chosen *loss function* which calculates the numerical difference between the given output and the desired output). These adjustments are determined by a chosen optimizer function, which is designed to choose adjustments reducing the loss as efficient as possible [61]. This process is called *backpropagation*.

Every full training cycle of the network on the dataset is called an *epoch*, whereas a full training on a single batch is called an *iteration*. So, if the training set consists of 1000 samples and the chosen batch size equals 500, it will take two iterations to complete an epoch.

Peculiarities and Parameters

Since neural networks are, like any other supervised learning technique, susceptible to overfitting, several measures exist to prevent the algorithm from focusing on irrelevant details. Examples of such measures are the use of *early stopping*, *input noise* and *dropout*. The intuition behind these methods will be explained in this section.

The first example of overfitting-preventing methods is called early stopping, because it causes the algorithm to stop training once training does not cause significant

improvement anymore. There are several ways to detect this, such as checking the accuracy at the end of every epoch and checking if the accuracy has not improved compared to the accuracy results of the previous n epochs. In this way, the algorithm has no chance of fitting the model too much to the training set [62].

The use of input noise causes that some of the values inside the input layer are set to a different value. By doing this, the algorithm needs to be more robust in order to keep classifying the samples correctly, since it cannot rely on every single detail in the input data anymore. In this way, the algorithm tends to stop overfitting [63].

The third example, dropout, removes a fraction of the edges. When a dropout of 0.1 is applied to the edges between layer r and s, 10% of all edges between r and s are removed. In this way, some of the information cannot traverse any further through the network. In case of details which are only present in a few nodes, the chance that they are lost is quite large; more generally related features that are stored in multiple nodes and therefore passed on over multiple edges, remain available in the network [64]. In this way, the algorithm is forced to generalize its knowledge, and thus the chance of overfitting is reduced.

A neural network has many parameters. Not only do the number and size of the layers vary, also different choices in activation functions exist, as well as parameters regarding e.g. dropout and input noise. Because the choice of parameters influences the performance of the algorithm, the most optimal choice of parameters needs to be found. Having many parameters makes it hard to find this optimal choice. Nevertheless, the optimal choice can be approximated using hyperparameter optimization techniques, such as grid search and random search [65].

3.1.3 Random Forest Classifier

Structure

A random forest classifier is a classification algorithm that uses multiple decision trees. A decision tree is build up from a root node at the top, followed by decision nodes and leaf nodes. These nodes are connected in a tree-like fashion. The root node and every decision node contain a question for which, in most cases, 2 answers are available: yes or no. Going through this tree from root to leaf will lead to a decision. A general model of a decision tree can be found in Figure 3.2.

Basic Principles

Each decision tree is trained on a subset of features of the provided training set. To predict the class belonging to a sample during the testing phase, the sample is fed to all trees, and the predicted outcomes of all trees are collected. Once all outcomes are available, the average outcome is taken or majority voting is performed [66].

During training, a decision tree is set to find the right questions for determining the class of a sample as efficient as possible. This happens by searching through the available features and checking what the result of a possible split would be. The progress made by a split is often expressed using the *Gini index* or the *information gain*. By starting off at the root and keep making splits until no further useful splits can be made, a decision tree is constructed [67, 68].

Because the decision trees in a random forest classifier are trained on different features, different results are provided upon predicting a sample's class. By training multiple trees on slightly different sets of features and taking the average of the

[1] Source: https://www.packtpub.com/sites/default/files/Article-Images/B03905_05_01.png

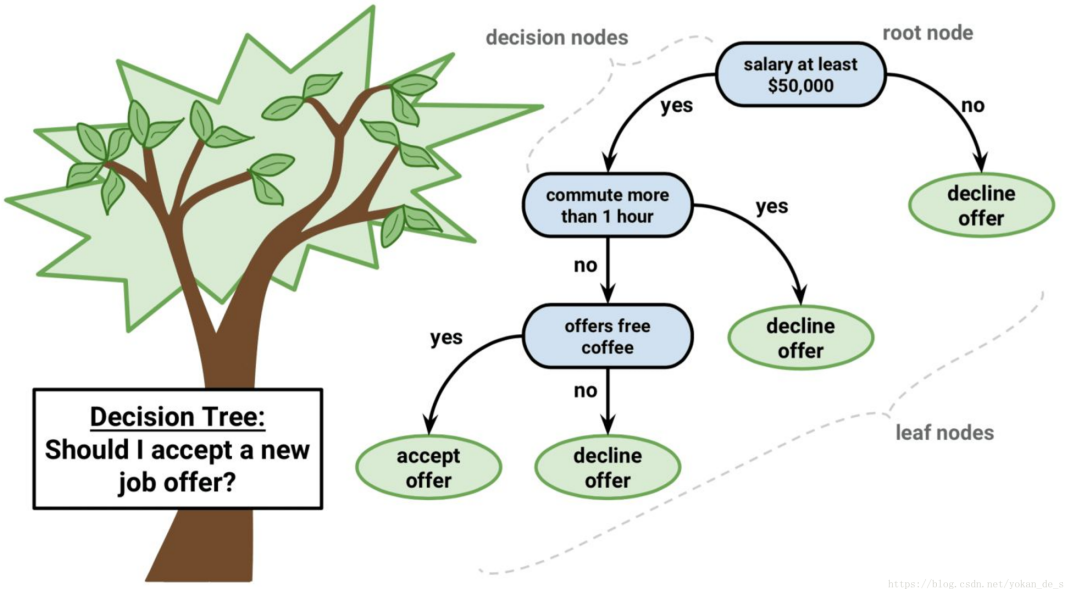

FIGURE 3.2: General model of a decision tree.[1]

prediction results, the result is less prone to overfitting and noise in the training set compared to an approach where a single decision tree is trained on the complete set of features [66].

Peculiarities and Parameters

A random forest classifier does not have as many parameters as a neural network. One of the most important parameters for a random forest classifier is the number of decision trees it has. Other parameters include limits to the width and depth of the trees and whether the progress made by a split is calculated using the Gini index (used to calculate the difference in purity between the current and the resulting distribution of classes) or the information gain (representing the difference in entropy between the current and the resulting distribution of classes). The Gini index is used as metric for finding the optimal decision for all experiments in this thesis.

3.2 Classification Performance Metrics

In order to evaluate and compare the performance of both classification algorithms, we adopted the metrics used in [1] and [2], since this enables us to compare our evaluation results to the performance stated in [1] and [2]. This section briefly explains these metrics briefly.

Several classification metrics are based on values which can directly be derived from a confusion matrix, a matrix showing the relation between the actual class of samples and their predicted class. A general example of a confusion matrix is shown in Figure 3.3.

Note that some classification problems are binary (e.g. classifying whether an email message is considered to be spam or not, classifying if a patient has a disease or not, or checking if a product may be sold or not), whereas other classification

[2] Source: https://gabrielelanaro.github.io/public/post_resources/multiclass/text4384.png

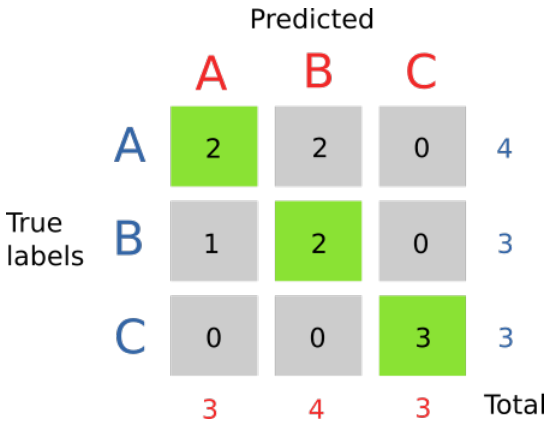

FIGURE 3.3: General model of a confusion matrix.[2]

problems are *multiclass* classification problems. Examples of multiclass classification problems are authorship attribution and malware family classification with more than 2 different authors or families.

In case of multiclass classification problems, performance metrics like recall or the F1-score are not directly applicable for the problem as whole, but only for evaluating the performance on a single class (*does sample x belong to class A?* instead of *to what class does sample x belong?*). However, the (weighted) average of such scores over all classes can be used as metric for the problem as a whole.

This section describes the metrics that are used in this thesis.

3.2.1 Recall / True Positive Rate (TPR)

The *recall*, also known as *true positive rate* or *sensitivity*, of a class A is defined as the number of samples *correctly* identified by a classification algorithm as A divided by the total number of samples labeled as A in the dataset, and can be denoted as:

$$\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

It therefore indicates how well an algorithm is able to classify the relevant samples from a certain class as being in that class.

3.2.2 False Positive Rate (FPR)

Likewise, the *false positive rate*, also known as *fallout*, *aspecificity* or *1 - specificity*, of a class A is defined as the number of samples *incorrectly* identified by a classification algorithm as A divided by the total number of samples not labeled as A in the dataset, and can be denoted as:

$$\frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

It therefore indicates to what extent an algorithm fails in classifying samples which do not belong to class X as being not in class X.

3.2.3 Precision

The precision, also known as *positive predictive value*, expresses the ability of the classifier to not mark a sample which is not in class A as being in class A. It can be

denoted as:

$$\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

3.2.4 F1-Score

The F1-score is defined as the harmonic mean of precision and the recall, which can be written as:

$$2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

3.2.5 ROC-Curve

The *Receiver Operating Characteristic*-curve, is a plot of the TPR against the FPR under various thresholds. The threshold determines in this case what minimal outcome probability is needed to classify a sample as a positive. By shifting this threshold, the behavior and robustness of a binary classifier can be explored.

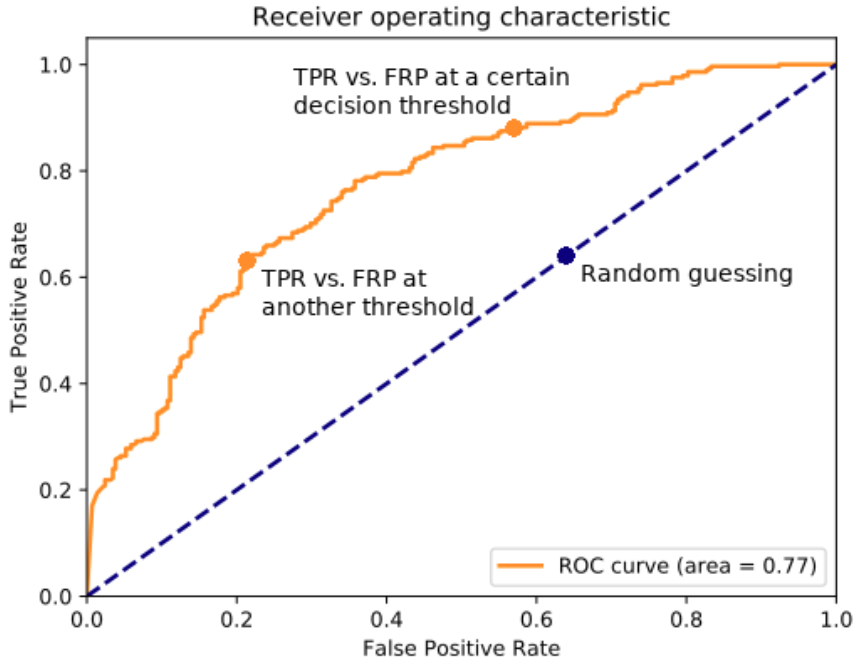

FIGURE 3.4: Example of a ROC-curve.

When the ROC-curve is plotted, the *area under the curve* (AUC) can be calculated. This allows us to reason about different models. Ideally, we want a model that has 100% TPR and 0% FPR, which will result in an ROC that only hits the point $(1, 0)$, and has a corresponding AUC of 1. The higher the AUC, the better a classification algorithm is considered to be. As shown in Figure 3.4, random guessing results in a straight ROC-curve and an AUC of 0.5; every algorithm that performs better than random, has a higher AUC.

3.2.6 Accuracy

Accuracy is a basic and commonly used classification performance metric, which is directly applicable on both binary and multiclass problems. It represents the number

of good predictions divided by the number of all predictions made, denoted as:

$$\frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$$

In Figure 3.3, this number of good predictions equals the sum of the values of the green cells. The advantage of working with accuracy is the fact that it is easy to calculate and very intuitive to understand.

However, when working with imbalanced datasets, accuracy could be a misleading metric. Imagine that a given dataset would consist for 95% of samples from class A, and that the remaining 5% belongs to class B. When a classifier would classify the whole dataset as belonging to class A, without even looking at the samples itself, it would retrieve a accuracy of 95%, which is considered to be quite good. However, the algorithm would be useless as meaningful classifier.

To mitigate this problem, additional metrics which do not take class imbalance into account must be used, e.g. by taking the unweighted average of F1-scores over all classes. By taking an unweighted average, misclassification of samples from smaller classes have a larger impact on the resulting score.

Chapter 4

Collecting a Dataset of State-Sponsored Malware

A supervised classification algorithm is only able to learn based on labeled data that serves as a ground truth. It is therefore of major importance for this research to have proper, labeled malware data at our disposal. Since the dataset used in [1] is not publicly available, the dataset used in [2] lacks state-sponsored samples and the fact that no ready-made datasets with state-sponsored malware are available, we need to come up with our own dataset of state-sponsored samples.

However, it is not easy to obtain labeled state-sponsored malware samples from open sources, due to the fact that not many samples are around. This is partially caused by the fact that the samples are kept secret, and are only occasionally revealed to the public [69]. Moreover, many malware databases that we encountered have a lack of structure, making it hard to select state-sponsored samples. The need for proper data on the one hand, and the lack of publicly available state-sponsored samples on the other hand, is a problem. This chapter describes a way to solve this problem, involving the use of threat intelligence reports to gather malware samples and the use of sandbox reports of the gathered samples as input for the classification algorithm.

4.1 Collection Method

Both approaches that we evaluate and compare make use of sandbox reports as input. Therefore, we need to come up with a dataset of samples and corresponding sandbox reports. So, the collection of the dataset consists of the collection of malware samples and the collection of sandbox reports. Both processes are described in the sections below.

4.1.1 Collecting Samples

Attributed state-sponsored malware samples can be found with the help of threat intelligence reports published by companies like FireEye, F-Secure and Kaspersky (e.g. [4, 70, 71]). Such reports often include an appendix with so-called *indicators of compromise* (IOCs), which consist of file hashes of samples or network traces identifying a specific piece of malware; in this way, such IOCs are used as an unique reference to a malware sample or family.

By looking for many threat intelligence reports and copying the mentioned file hashes from those reports, we created an aggregated list of file hashes of malware samples. This list is then used to download the samples from any larger malware database, since every hash refers to an unique malware sample.

Most intelligence reports where found using an overview of APT groups by FireEye [4] as starting point. Also overviews like the Google Docs Spreadsheet 'APT Groups and Operations' by Florian Roth [72] provide links to many useful malware hashes.

Although threat intelligence reports contain numerous file hashes, it would be better for some classes to have more samples, since the smallest class only consists of 32 samples. The lack of samples from reliable sources is mainly due to several complications described in Chapter 10. Therefore, we've investigated additional sources of malware samples like ThreatMiner [73], but since such websites fail in substantiating claims to which family or actor a sample belongs, these additional sources are neglected for this particular research.

Apart from keeping the reliability of the used sources as high as possible, a somewhat equal distribution of nation-states and families is sought for as well during the collection of the samples. This means for example that many available samples that allegedly originate from China are ignored, since the dataset already contained many Chinese samples.

For this research, VirusTotal [74] is used to download samples from, using the VirusTotal API. In total, 4,449 samples were requested from VirusTotal, from which 3,594 unique samples were available for download. The samples allegedly originate from 12 different state-sponsored APT groups spread over 5 countries, namely China, North-Korea, Pakistan, Russia and the USA. All retrieved samples and an overview of all requested samples and the sources from which their file hashes are obtained can be found online [1].

4.1.2 Collecting Sandbox Reports

Having collected the samples, we need to retrieve sandbox reports about the sample as well, since these reports are used as input for the classifier. For each sample, a basic sandbox report is downloaded from VirusTotal as well. This report is written in JSON-format and contains some file information, findings of anti-virus companies and a brief overview of the loaded libraries and behavior of the malware.

Moreover, the samples are fed through an API to two other sandboxes, namely Cuckoo [51] and VMRay [75]. This is done in order to receive additional information about the samples which is not included in the reports from VirusTotal. An example of such information is a list of all API calls with arguments a sample performs, which is needed in the approach of **Am17**.

There is a high probability that not all samples are executed successfully in the sandbox, due to a sudden termination (caused by a bad configuration) or the appearance of misleading behavior (caused by sandbox evasion techniques as described in Section 2.2.2). However, unsuccessful runs still generate sandbox reports that provide some information which can be used for classification. Nevertheless, it is worth finding out what the influence of unsuccessful runs is on the performance of the classification algorithms.

4.2 Data Preprocessing

After the malware samples have been downloaded from VirusTotal and the corresponding sandbox reports have been retrieved as well, several steps are performed

[1] The repository containing the dataset is available via https://cyber-resear.ch/APTMalware.

to complete and improve the dataset. These steps consist of duplicate detection, filtering the sandbox reports to remove unusable information and thus reduce the filesize. Moreover, the data is transformed into a format that suits a supervised learning algorithm.

4.2.1 Duplicate Detection

Because different types of hashes are used online to refer to malware samples, some samples are requested twice (e.g. once using the corresponding `MD5` hash, and once using the corresponding `SHA-1` hash of a sample). These duplicate samples are detected by calculating the `SHA-256` hash for every sample and checking if some hashes appear multiple times. This was the case for multiple pairs of samples. For each of the pairs, one sample and corresponding sandbox reports have been kept.

4.2.2 Filtering Sandbox Reports

Some of the collected sandbox reports have a filesize of more than 300MB, mainly made up of large chunks of data which is irrelevant for performing authorship attribution. In the case of the reports generated by Cuckoo, these chunks are filled with buffers which are used as an argument of API calls. Such buffers are often encountered more than once in a single report, since the corresponding API calls were executed multiple times during execution in the sandbox.

In case of the reports generated by VMRay, the large filesize is owing to an extensive amount of reported information on process dumps, which hardly contain any information that is useful in a bag of words approach.

These irrelevant chunks are removed using jq [76], a command-line JSON processor which is capable of modifying large JSON-files. The reports from VirusTotal do not contain large chunks of irrelevant data, and are therefore not filtered.

4.2.3 Extracting API Calls

Some extra preprocessing is performed using jq [76] in order to extract the API calls (referred to as *primary features* in [2]) and their arguments (referred to as *secondary features* in [2]) from the Cuckoo reports. This API call extraction is needed to evaluate the performance of **Am17** stated in the original paper, since the original paper uses API call extraction. The approach of using only API calls and their arguments as features is specifically designed for Cuckoo reports and cannot be used on other sandbox reports used in this paper, since these sandbox reports do not mention the executed API calls.

In order to establish the secondary features as described in [2], the 500 most common arguments per APT group are extracted, after which the arguments that occur in multiple APT groups are removed. The resulting arguments are candidates for static secondary features. From these secondary features, one feature per class is used. This features needs to be manually selected on beforehand, based on how characteristic the API call argument is for a given class. Therefore, the use of these secondary features is questionable, since it entails the provision of manually selected features to the algorithm, whereas one could argue that a proper classification algorithm needs to discover the relevant features on its own.

4.2.4 Creating a Bag of Words

In order to use the dataset as input for a supervised learning algorithm, the data needs to be transformed into a suitable format.

Since we want to compare **Ro18** and **Am17** on our dataset, we convert the dataset using the *bag-of-words* model, because this is used by **Ro18** and **Am17** as well. A *bag of words* is a matrix indicating the frequency of a set of words in each file. This matrix, denoted by **M**, is constructed by extracting all words from every file, and selecting the top n most frequent words for further use. The matrix is filled, such that element $\mathbf{M}_{ij}$ represents the frequency of word j in file i.The bag of words is created using the `CountVectorizer` from scikit-learn [77].

A boolean value, indicating whether a given word appears or not in a report, can be chosen as well instead of the frequency of a word, but since a boolean value contains less information, the choice is made to use the frequency of appearance instead of a boolean value. However, it is still worthwhile to compare the performance achieved on a binary bag of words compared to a non-binary bag of words.

Moreover, other models for representing the data could have been used as well, such as *tf-idf*. Future work might include research to find out which method causes best performance.

4.3 Overview of the Collected Dataset

The characteristics of the final dataset, after being preprocessed, can be found in Table 4.1. Several assumptions have been made during the construction of the dataset. Firstly, we assume that an APT group is only sponsored by a single country, unless explicitly stated otherwise, which is quite a regular assumption. Secondly, whenever a threat intelligence report describes a single APT, all samples that are referred to without explicit information are considered to have that specific APT group as only author. This assumption is not trivial, since it excludes the possibility that a sample is used by multiple APT groups. However, it is necessary in order to perform classification, since the responsible APT group would otherwise stay uncertain.

The construction of the dataset did cost approximately a month. This was largely due to the fact that it is hard to obtain state-sponsored samples, and the fact that some unreliable sources were used as well at first. With the knowledge of some useful resources and methods to collect samples, the process could be repeated in approximately 1 or 2 weeks.

Note that the number of Chinese and Pakistani samples is significantly higher than the number of samples originating from the USA (2028 and 1085 Chinese and Pakistani samples respectively, compared to 395 American samples) and that the distribution of APT groups is not equal as well. Such imbalance causes the dataset to be harder to classify compared with perfectly balanced datasets, since the algorithm tends to learn less about minority classes or even ignore those classes. However, imbalanced distributions cause the trained algorithms to be more robust to handling with real-world scenarios where data imbalance may occur as well.

4.4 Dealing with Imbalanced Datasets

A possible way of dealing with imbalanced datasets is the use of *undersampling* and *oversampling* [78]. *Random undersampling* reduces the number of samples in the

Country	APT Group	Family	# Requested	# Downloaded	# Missing VMRay
China	APT 1		1007	405	0
China	APT 10	i.a. PlugX	300	244	12
China	APT 19	Derusbi	33	32	1
China	APT 21	TravNet	118	106	5
Russia	APT 28	'Bears'	230	214	11
Russia	APT 29	'Dukes'	281	281	0
China	APT 30		164	164	0
North-Korea	DarkHotel	DarkHotel	298	273	2
Russia	Energetic Bear	Havex	132	132	0
USA	Equation Group	Fannyworm	395	395	0
Pakistan	Gorgon Group	Different RATs	1085	961	12
China	Winnti		406	387	2
Total			4449	3594	45

TABLE 4.1: Characteristics of the Collected Dataset

majority classes and can be implemented by simply picking less samples from the majority groups. Using oversampling, extra samples are generated for the minority classes, making the classes balanced again. Possible techniques for this include SMOTE [79] and ADASYN [80], or just a naive approach involving sampling with replacement called *random oversampling*, where samples of minority classes are picked multiple times. Both random undersampling and random oversampling are implemented in this research using the imbalanced-learn library [81].

However, using both undersampling and oversampling with a bag of words-based approach gives rise to some problems. An imbalanced dataset influences the resulting bag of words, because only the n most frequent words are included. Therefore, keywords for minority classes are likely to get ignored since their frequency in the total dataset is too low. Because of this, undersampling and oversampling need to take place before the bag of words is constructed, otherwise the keywords of minority classes are lost before anything is done to fix the imbalance. Some advanced undersampling and oversampling methods like SMOTE and ADASYN generate new samples based on existing samples; translated to this domain, it would mean that realistic sandbox reports need to be generated, based on existing reports. Since this is very hard to achieve, such advanced undersampling and oversampling methods are not used in this research.

Chapter 5

Experimental Data & Setup

Before **Ro18** and **Am17** are compared in Chapter 8, they are first individually evaluated on the data set obtained in Chapter 4, followed by a comparison of the obtained results and the results noted in the original papers describing **Ro18** and **Am17**. The goal of this evaluation is to get the maximum performance out of **Ro18** and **Am17** for this dataset without changing the input format fundamentally, and checking if this performance gets close to the performance described in the original papers.

This chapter describes the general experimental setup for this evaluation. Chapters 6 and 7 describe the configuration used for **Ro18** and **Am17** respectively, followed by the results achieved by each approach.

5.1 Constructing Training and Test Sets

Once the data is retrieved and preprocessed as described in Chapter 4, it is divided into a training and a test set. However, this brings in an important issue.

If malware from all different APT groups are contained in both the training and test set, the neural network could tend to learn which *APT groups* belong to a certain country, and is thus only able to answer Question B. However, when APT groups are strictly separated in training and test set, the neural network is forced to learn what *APT group-transcending malware characteristics* are indicators for a given country, and is thus able to answer Question A.

Although the latter approach gains more information on a successful attempt, it is way more difficult to do so, since it strengthens requirements on input data [1] and it is harder to extract such high-level properties which are not directly present in brief JSON-reports.

> The problem of choosing the right split of data in order to learn about class-transcending properties goes analogous with the following example:
>
> Imagine that you need to classify 50 people from 10 different households on the city they live in. All people live in either New York or Washington. You first get to see 40 people, of which you get to know many properties, like their height, weight, facial features, family name, level of education, income etc., as well as the city they live in (analogous to the training set). When looking at those 40 people, you try to figure out rules that are helpful for determining in which city the other 10 people live in (analogous to the test set).

[1] This mostly concerns the distribution of both the training and test set. Instead of only requiring two equal-sized populations from two different nation-states, an additional requirement is imposed. This requirement is that the population of all samples from any nation-state can be split into a training and test set using a 80-20 ratio, without having a APT group with samples spread over both the training and test set.

It could be possible that the only rule you make is based on the family name of the people; in this way you could see from which household they are and by looking at the people from that household you saw earlier, you could determine in which city they live. However, the actual question you are answering then is *'From which household is this person?'*, instead of finding more general ground truths about people living in New York and Washington.

This problem can be solved by splitting the people in such a way, that 2 households of 5 people (one household from New York and one from Washington) form the test set, and that the other households make up the training set. In this way, making rules based on family names would not work anymore, since the test set only contains family names that you have never seen before. Because of this, you are forced to learn *family-transcending* properties about the inhabitants of New York and Washington.

5.2 Tested Scenarios

The paper describing **Ro18**, [1], describes 3 different authorship attribution scenarios to train a classification algorithm for. These scenarios can be formulated as follows:

A. Authorship attribution on country level, based on unseen APT groups

B. Authorship attribution on country level, based on earlier seen APT groups

C. Authorship attribution on APT group level

The use of different scenarios enables us to gain more information about the classification capabilities of both approaches compared to using only one scenario. Note that the separation technique described in Section 5.1 is needed in order to perform authorship attribution on country level, based on unseen APT groups (Scenario A).

5.3 Sandboxes Used

This section provides an overview of the content of the 3 sandboxes that are used.

5.3.1 Cuckoo

Cuckoo reports contain information about many different aspects of a sample. Both static as well as dynamic analysis is performed and all results are included in a single report. Moreover, paths to screenshots and other output files (e.g. the dropped file in case the sample drops a file) are included in the report as well. The lay-out of the report can be found in Table 5.1.

Section	Content
info	Details about sandbox run, like duration and type of VM used
signatures	Hits on any signatures configured in the sandboxes
target	File information about the sample, like several hashes
network	Logs and statistics about network activity
static	Structural characteristics, like content of PE headers
behavior	Execution data, including evoked API calls and processes
debug	Logging, useful for debugging incorrect or crashing runs
screenshots	Paths to the screenshots made by the sandbox
strings	Output of the `strings` command on the sample
metadata	Additional output files as a result of the sandbox run

TABLE 5.1: Lay-out of a Cuckoo report

Since Cuckoo reports contain both proper static and dynamic analysis, we consider Cuckoo reports as a good representation of a malware sample. The API calls that are extracted for as described in Section 4.2.3, can be found in the section behavior.

5.3.2 VirusTotal

The reports from VirusTotal contain information about the submission of the sample, as well as the results of scans performed by numerous anti-virus solutions. The lay-out of VirusTotal reports can be found in Table 5.2.

Section	Content
vhash	Unique VirusTotal hash
submission names	Other filenames used for the same file
additional info	Basic static analysis, like API imports
scans	Results of numerous anti-virus scans
tags	Tags linked to the sample
other	File hashes, timestamps etc.

TABLE 5.2: Lay-out of a VirusTotal report

Since VirusTotal reports only contain some basis static analysis and the results of scans of anti-virus solutions, and no behavioral analysis, we consider these reports to be helpful in the process of classification, but not as a complete representation of a malware sample.

5.3.3 VMRay

For most submissions to the VMRay sandbox, several reports are generated. When the sandbox qualifies the uploaded sample as an executable sample, it generates 3 reports: one containing static analysis and two containing dynamic analysis using two different execution environments. The structure of the reports generated during a static run can be found in Table 5.3. The sections that are added in case of dynamic analysis can be found in Table 5.4.

Section	Content
analysis details	Details about sandbox run, like timestamp and duration
artifacts	Extracted characteristics, like used domains or registry keys
classification	Result of the classification performed by VMRay
extracted files	Paths to files which are extracted during sandbox execution
remarks	Any errors of warnings encountered during sandbox execution
sample details	File information about the sample, like several hashes
type	Type of the report (summary)
version	Version of the report
vm and analyser details	Configuration of the VM
vti	Results from the VMRay Thread Identifier
yara	Settings regarding the use of Yara rules

TABLE 5.3: Lay-out of a VMRay report containing static analysis

Section	Content
process dumps	Dumps made by the sandbox of running processes
processes	Details about the running processes during sandbox execution
screenshots	Paths to the screenshots made by the sandbox

TABLE 5.4: Additional sections of a VMRay report in case of dynamic analysis

Compared to the reports generated by Cuckoo, the VMRay reports focus more on process dumps and less on network activity. Moreover, the reports generated on static analysis do not contain much information. For example, basic information such as PE headers are not present in these reports.

In order to use all information available, we merge the reports generated by both static and dynamic analysis into a single VMRay report. Just as it is the case with Cuckoo reports, we consider the combination of static and dynamic VMRay reports as a good representation of a malware sample.

5.4 Overview of Training and Test Sets

The original paper describing **Am17** uses the extracted API calls and arguments as only features to prevent overfitting [2]. However, methods exist which use random forest classifiers to select relevant features, implying that a random forest classifier is able find relevant features in large feature sets by on its own [82]. Therefore, we evaluated **Am17** using the full sandbox reports as well, to see if a selection of features is really needed to prevent overfitting.

5.4.1 Used Variants

To test all relations in several variants of the dataset, 36 variants of the datasets are constructed to evaluate **Ro18** and **Am17**. A schematic overview of the used variants can be found in Figure 5.1. The used variants are constructed as follows:

- A set with a strict APT group separation between the training and test set, labeled by country (used for Scenario A). Note that **only Chinese and Russian samples** are included, since China and Russia are the only countries with

multiple APT groups in the data set. This causes the problem to be a binary classification problem, because only two classes are involved.

- A set with no APT group separation between the training and test set, labeled by country (used for Scenario B)
- A set with no APT group separation between the training and test set, labeled by APT group (used for Scenario C)

All mentioned sets occur in 3 different variants with respect to undersampling or oversampling:

- No undersampling or oversampling conducted, thus kept imbalanced
- Random oversampling
- Random undersampling

Moreover, all mentioned sets occur in 4 fashions with respect to the used data source:

- Using extracted API calls and arguments from Cuckoo reports
- Using filtered Cuckoo reports
- Using the original VirusTotal reports
- Using filtered VMRay reports

The methods for the sampling and filtering of the data sources, as well as the extraction of the API calls and arguments, are described in Sections 4.2 and 4.4.

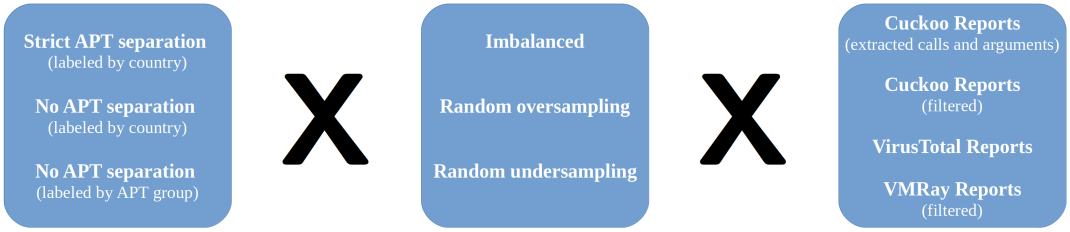

FIGURE 5.1: Schematic overview of the used variants of the dataset. There are 3 different properties (setup, sampling and data source) which each occur in different variants, leading to 36 unique variants of the dataset.

5.4.2 k-Fold Cross-Validation

In order to get a good performance estimate without reducing the number of samples in the training set too much, k-fold cross-validation is applied on the datasets, using $k = 5$. This means that the dataset is split into 5 parts, and that in each of 5 runs, 4 partitions are used as training set and 1 partition as test set. The average of the results of the 5 runs are used as a solid approximation of the performance of the algorithm.

Chapter 6

Evaluation of the Neural Network-based Approach (Ro18)

In this chapter, the configuration of the used neural network is described, followed by the experimental results and some preliminary conclusions.

6.1 Configuration of the Neural Network

The neural network that we used is based on the one described in [1], but small modifications are made to get better results. Instead of a 10-layers fully connected neural network, one with 8 layers is used. These layers are build up following a 50,000-2,000-1,000-1,000-1,000-1,000-500-n structure, i.e. the network has an input layer of 50,000 neurons, followed by a hidden layer of 2,000 neurons etc. The value of n is determined by the number of output classes.

An output softmax layer is used as well, just like the described dropout and input noise rates of 0.5 and 0.2 respectively. The ReLU activation function is used and categorical cross entropy is used to calculate the loss. The Adam optimizer is chosen as optimizer with the fixed learning rate of 0.0001. This network is implemented using Keras [83] on top of TensorFlow [84]. It is used for training for answering all questions mentioned in Section 5.2.

The learning rate is not taken straight from [1], but determined using a trial-and-error approach, since the method used in [1] is not compatible with Keras. The chosen learning rate of 0.0001 turns out to perform slightly better in our tests compared to a commonly used learning rate of 0.001[1], but it consumes more time and computation power. Future work includes performing hyper-parameter tuning using a technique like random search [65] or bayesian optimization [85], to find the best set of parameters in a more structured way.

6.2 Results

Running the neural network over the datasets as described in Section 5.4, the results described below were obtained. Note that only accuracy is used as classification performance measures in **Ro18**, and only results using the same metrics are comparable. However, we still use multiple metrics in order to get a more comprehensive

[1] For example, the best results in Table 6.4 would equal 0.894 (σ: 0.015) for imbalanced Extracted Cuckoo, 0.939 (σ: 0.015) for imbalanced Filtered Cuckoo, 0.980 (σ: 0.008) for imbalanced VirusTotal and 0.948 (σ: 0.009) for oversampled Filtered VMRay. Note that the deviation of the results increases when this higher learning rate is used, whereas the accuracy slightly drops.

view on the performance of **Ro18**, which we can use for the comparison with **Am17**. Each run of training and testing the algorithm took appropriately 15-45 minutes.

Since 5-fold cross-validation is used, 5 runs are executed using staggered parts of the dataset as training and test set. The average of those runs is noted as result, together with the standard deviation σ.

6.2.1 Country-Level Authorship Attribution with Unseen APT Groups

The accuracy of the classification neural network trained for country-level authorship attribution with unseen APT groups (thus belonging to Scenario A) is tested to be up to **76.6%**.

Dataset	Imbalanced	Undersampled	Oversampled	Ro18 in [1]
Extracted Cuckoo	0.572 (σ: 0.016)	0.573 (σ: 0.103)	**0.583 (σ: 0.044)**	-
Filtered Cuckoo	0.424 (σ: 0.044)	0.416 (σ: 0.075)	**0.429 (σ: 0.041)**	**0.986**
VirusTotal	0.727 (σ: 0.044)	**0.741 (σ: 0.019)**	0.725 (σ: 0.048)	-
FilteredVMRay	**0.766 (σ: 0.057)**	0.758 (σ: 0.053)	0.732 (σ: 0.029)	-

TABLE 6.1: **Accuracy** Results Evaluating **Ro18** on Scenario A

A noticeable fact is that the use of Cuckoo reports leads to the worst results in our case, whereas in [1], Cuckoo reports are used by default and result in good scores. The accuracy results, shown in Table 6.1, show that the accuracy of the classifier trained on filtered Cuckoo reports has a maximum of 0.429, which much lower compared to an accuracy of 0.986 as achieved in [1].

Dataset	Imbalanced	Undersampled	Oversampled
Extracted Cuckoo	0.554 (σ: 0.017)	0.570 (σ: 0.103)	**0.573 (σ: 0.038)**
Filtered Cuckoo	0.345 (σ: 0.027)	**0.387 (σ: 0.097)**	0.355 (σ: 0.052)
VirusTotal	0.616 (σ: 0.084)	**0.669 (σ: 0.028)**	0.613 (σ: 0.098)
FilteredVMRay	**0.687 (σ: 0.090)**	0.682 (σ: 0.143)	0.647 (σ: 0.070)

TABLE 6.2: **F1-Score** Results Evaluating **Ro18** on Scenario A

The quality of the classifier performing authorship attribution with unseen APT groups appears to be strongly dependent on the sandbox reports that are used. The results in Table 6.2 show that the classifier which uses filtered VMRay reports achieves a F1-score of 0.687, whereas the classifier using filtered Cuckoo reports only gets up to a F1-score of 0.387. Looking at the sampling strategy, all variants perform more or less on the same level, although the undersampled dataset causes less stable results.

Dataset	Imbalanced	Undersampled	Oversampled
Extracted Cuckoo	0.417 (σ: 0.020)	**0.351 (σ: 0.113)**	0.367 (σ: 0.050)
Filtered Cuckoo	0.658 (σ: 0.033)	**0.590 (σ: 0.136)**	0.648 (σ: 0.055)
VirusTotal	0.386 (σ: 0.067)	**0.340 (σ: 0.029)**	0.384 (σ: 0.083)
FilteredVMRay	0.319 (σ: 0.094)	**0.286 (σ: 0.114)**	0.351 (σ: 0.071)

TABLE 6.3: **FP Rate** Results Evaluating **Ro18** on Scenario A

This F1-score and false positive rate are calculated in such a way, that they do not take class imbalance into account. Therefore, these metrics are not prone to misleading results caused by class imbalance. Since the classes in this setup are imbalanced

indeed, we can use this metric to see to what extent the accuracy could be off due to class imbalance, since this metric is prone to distortion caused by class imbalance. In case of this experiment, the accuracy follows the same patterns as the F1-score, and seems therefore not to be distorted.

The false positive rate indicates to what extent the algorithm fails in classifying samples which do not belong to class X as being not in class X. The results in Table 6.3 show that the VMRay reports lead to the lowest false positive rate, whereas filtered Cuckoo reports lead to the highest. Since a low false positive rate implies that only a few samples are attributed to an actor incorrectly and the fact that such incorrect attribution (i.e. stating that nation-state A launched a malware campaign whilst they did not) could have large consequences, solutions with a low FPR are strongly preferred. Again, the undersampled datasets cause less stable results, although the average result of the undersampled variant is are slightly better.

This means that for authorship attribution with unseen APT groups using a neural network-based approach, it is best to use the reports generated by VMRay and worst to use Cuckoo reports. With regard to sampling methods, there are significant performance differences between the different strategies, but no overall best sampling strategy can be selected. The classifier performs mediocre with an accuracy of 0.766, and is thus not suitable for non-experimental use. The results regarding recall, precision and AUC can be found in Tables A.1 to A.3 respectively.

6.2.2 Country-Level Authorship Attribution with Earlier Seen APT Groups

The accuracy of the classification neural network trained for country-level authorship attribution with earlier seen APT groups (thus belonging to Scenario B) is tested to be up to **98.8%**.

Dataset	Imbalanced	Undersampled	Oversampled	Ro18 in [1]
Extracted Cuckoo	**0.896 (σ: 0.011)**	0.748 (σ: 0.131)	0.878 (σ: 0.003)	-
Filtered Cuckoo	**0.945 (σ: 0.010)**	0.907 (σ: 0.006)	0.930 (σ: 0.011)	**0.978**
VirusTotal	**0.988 (σ: 0.006)**	0.973 (σ: 0.006)	0.987 (σ: 0.003)	-
FilteredVMRay	0.943 (σ: 0.007)	0.918 (σ: 0.010)	**0.952 (σ: 0.007)**	-

TABLE 6.4: **Accuracy** Results Evaluating **Ro18** on Scenario B

Because this experiment is executed on a highly imbalanced dataset (the smallest class, North-Korea, containing 298 samples, whereas the largest class, China, has 2028), chances exist that the calculated accuracies are affected by the class imbalance. The accuracy results, shown in Table 6.4, are indeed affected by class imbalance. For example, the classifier using the oversampled variant of the extracted API calls has the lowest false positive rate, but a significant lower accuracy score compared to the accuracy score of the imbalanced version of the extracted Cuckoo set. This means that the accuracy results do represent the actual performance on the current dataset with its corresponding class distribution, but they may fail to indicate on an imbalanced dataset to what extend the algorithm has learned to differentiate every class from other classes.

Dataset	Imbalanced	Undersampled	Oversampled
Extracted Cuckoo	**0.885 (σ: 0.016)**	0.737 (σ: 0.150)	0.868 (σ: 0.005)
Filtered Cuckoo	**0.941 (σ: 0.011)**	0.898 (σ: 0.012)	0.916 (σ: 0.016)
VirusTotal	0.985 (σ: 0.008)	0.970 (σ: 0.006)	**0.986 (σ: 0.004)**
FilteredVMRay	0.937 (σ: 0.008)	0.906 (σ: 0.012)	**0.946 (σ: 0.008)**

TABLE 6.5: **F1-Score** Results Evaluating **Ro18** on Scenario B

The undersampled variants of the dataset scores lower and less stable compared to the other variants, especially in the case of extracted API calls from Cuckoo reports. This is likely due to the fact that the undersampled version of the dataset containing extracted API calls is smaller[2], making it harder for the classifier to find prevailing patterns which work well for classification.

The best F1-scores are achieved using reports from VirusTotal, as shown in Table 6.5. The fact that the F1-scores are this high, means that both the precision and recall scores are high, thus indicating that all datasets work well for authorship attribution using earlier seen APT groups.

Dataset	Imbalanced	Undersampled	Oversampled
Extracted Cuckoo	0.031 (σ: 0.003)	0.064 (σ: 0.036)	**0.030 (σ: 0.001)**
Filtered Cuckoo	**0.015 (σ: 0.003)**	0.023 (σ: 0.000)	0.017 (σ: 0.002)
VirusTotal	0.003 (σ: 0.002)	0.007 (σ: 0.001)	**0.003 (σ: 0.001)**
FilteredVMRay	0.015 (σ: 0.002)	0.021 (σ: 0.003)	**0.014 (σ: 0.002)**

TABLE 6.6: **FP Rate** Results Evaluating **Ro18** on Scenario B

When we compare the results that we achieved to the results presented in [1], the results are similar. Although the results on the filtered Cuckoo dataset are slightly worse, the results achieved on other datasets are better than the results presented by Rosenberg et al.

With accuracies ranging from 0.748 to 0.988, the neural network-based classifier is useful for solving authorship attribution with earlier seen APT groups. Especially the variants which use VirusTotal and filtered VMRay reports are suitable for deployment in non-experimental environments. The results regarding recall, precision and AUC can be found in Tables A.4 to A.6 respectively.

6.2.3 APT Group-Level Authorship Attribution

The accuracy of the classification neural network trained for APT group-level authorship attribution (thus belonging to Scenario C) is tested to be up to **98.7%**.

An important remark is that the approach used here differs from the one described in [1], where the results were retrieved on the validation set. Although it is not clear which validation method has been used, the use of a validation set to test the final accuracy of an approach is wrong. This is because the validation set has been involved in training as well. When the performance of a supervised learning algorithm is tested on data it has been training on, the outcome is biased, since the algorithms capability of classifying unseen data is not tested at all. Unlike **Ro18**, our results were retrieved on the test set.

[2] Since the country represented the least in the dataset is North-Korea with 298 samples. Due to 5-fold cross-validation, 80% of the samples are included in each variant of the training set, so at most 239 samples per class are included in the undersampled dataset.

Dataset	Imbalanced	Undersampled	Oversampled	Ro18 in [1]
Extracted Cuckoo	**0.843 (σ: 0.017)**	0.053 (σ: 0.014)	0.816 (σ: 0.015)	-
Filtered Cuckoo	**0.935 (σ: 0.009)**	0.356 (σ: 0.153)	0.920 (σ: 0.011)	**0.998**
VirusTotal	0.984 (σ: 0.004)	0.204 (σ: 0.108)	**0.987 (σ: 0.003)**	-
FilteredVMRay	0.929 (σ: 0.012)	0.130 (σ: 0.013)	**0.929 (σ: 0.009)**	-

TABLE 6.7: **Accuracy** Results Evaluating **Ro18** on Scenario C

Apart from the undersampled variants, the classifier performs well, especially when VirusTotal or filtered VMRay reports are used, with accuracies up to 0.987. This makes this classifier suited for use in non-experimental environments. The results based on filtered Cuckoo reports are not as high as described in [1], but our best results are comparable to the performance described by Rosenberg et al., as shown in Table 6.7.

Dataset	Imbalanced	Undersampled	Oversampled
Extracted Cuckoo	**0.753 (σ: 0.016)**	0.024 (σ: 0.017)	0.733 (σ: 0.017)
Filtered Cuckoo	0.896 (σ: 0.022)	0.266 (σ: 0.142)	**0.901 (σ: 0.016)**
VirusTotal	0.961 (σ: 0.018)	0.114 (σ: 0.064)	**0.973 (σ: 0.007)**
FilteredVMRay	0.877 (σ: 0.023)	0.056 (σ: 0.029)	**0.887 (σ: 0.008)**

TABLE 6.8: **F1-Score** Results Evaluating **Ro18** on Scenario C

Just as described in Section 6.2.2, we notice again in the results that the undersampled variants of the dataset perform worse than other datasets, and this time the effect is more severe, as can be derived from Table 6.8. The fact that this effect shows up more severe, is likely to be caused by the fact that the smallest class is even smaller, leading to a big loss of information[3]. The imbalanced and especially the oversampled datasets meanwhile, perform well, with F1-scores based on VirusTotal reports reaching up to 0.973.

Dataset	Imbalanced	Undersampled	Oversampled
Extracted Cuckoo	**0.014 (σ: 0.002)**	0.083 (σ: 0.001)	0.016 (σ: 0.001)
Filtered Cuckoo	**0.007 (σ: 0.001)**	0.057 (σ: 0.014)	**0.007 (σ: 0.001)**
VirusTotal	0.002 (σ: 0.000)	0.072 (σ: 0.010)	**0.001 (σ: 0.000)**
FilteredVMRay	**0.007 (σ: 0.001)**	0.078 (σ: 0.004)	**0.007 (σ: 0.001)**

TABLE 6.9: **FP Rate** Results Evaluating **Ro18** on Scenario C

Since there are 12 different APT groups in the dataset compared to 5 different countries, the false positive rate much lower than one would expect based on the previous 2 experiments. This is caused by the fact that in the case of multi-class classification, the per-class number of false positives is lower than the number of true negatives. Table 6.9 proves this: the false positive rates for the undersampled case in this table are lower compared to those in Table 6.3, although the F1-scores in Table 6.8 are lower than those in Table 6.2.

Moreover, the false positive rates follow the pattern when looking at Table 6.7, from which we may conclude that the calculated accuracies are not distorted due to

[3] Since the smallest APT group in the dataset is APT 19, which contains 32 samples. Due to 5-fold cross-validation, 80% of the samples are included in each variant of the training set, so at most 26 samples per class are included in the undersampled dataset.

class imbalance. The results regarding recall, precision and AUC can be found in Tables A.7 to A.9 respectively.

6.3 Preliminary Conclusions

The following preliminary conclusions can be drawn from the evaluation of **Ro18**:

- The results differ to the results described in [1], which is not surprising since the experiments were performed on different datasets.
- The performance of our classifiers is lower than the results described in [1] under similar circumstances (i.e. using filtered Cuckoo reports). This could be due to the fact that the dataset used by Rosenberg et al. is larger, perfectly balanced and containing less different families than the dataset described in Chapter 4, thus leading to better scores.
- The neural network used is capable of finding features which serve as indicators of a country or APT group linked to a malware sample, since most classifiers score decently. This also proves the fact that such features indeed exist.
- The performance achieved on answering Scenario A is not comparable to the results achieved by **Ro18** in the original paper. However, since the performance achieved goes up to 0.766, the algorithm must have grasped to some extent what patterns are characteristic for a certain country. This proves that even different malware developing groups within a country tend to use similar techniques, infrastructures or coding styles. Therefore, a trained network could be used to extract characteristics that provide specific clues to malware analysts about where to look for traces indicating the state behind the sample, even when the sample comes from an unseen APT group.
- The performance achieved on answering Scenarios B and C is comparable to the results mentioned in earlier research like [1], the paper describing **Ro18**, and [3]. This proves the point that it is possible to accurately classify malware based on country and APT group using sandbox reports.

Chapter 7

Evaluation of the RFC-based Approach (Am17)

In this chapter, the configuration of the used random forest classifier is described, followed by the experimental results and some conclusions.

7.1 Configuration of the Random Forest Classifier

A Random Forest Classifier is used with 100 trees, just like described in [2], the original paper proposing **Am17**. The Random Forest Classifier is implemented using the `RandomForestClassifier` from scikit-learn [77].

7.2 Results

Running the random forest classifier over the datasets as described in Section 5.4, the results presented below were obtained. Note that the same metrics are used as classification performance measures in **Am17**, in order to get comparable results. Each run of training and testing the algorithm took approximately 2 seconds.

7.2.1 Country-Level Authorship Attribution with Unseen APT Groups

The accuracy of the random forest classifier trained for country-level authorship attribution with unseen APT groups (thus belonging to Scenario A) is tested to be up to **74.1%**.

Dataset	Imbalanced	Undersampled	Oversampled
Extracted Cuckoo	0.465 (σ: 0.037)	0.458 (σ: 0.096)	**0.572 (σ: 0.074)**
Filtered Cuckoo	0.475 (σ: 0.083)	**0.494 (σ: 0.067)**	0.458 (σ: 0.039)
VirusTotal	0.721 (σ: 0.031)	**0.741 (σ: 0.013)**	0.737 (σ: 0.020)
FilteredVMRay	0.669 (σ: 0.027)	0.585 (σ: 0.112)	**0.710 (σ: 0.066)**

TABLE 7.1: **Accuracy** Results Evaluating **Am17** on Scenario A

The performance of the RFC-based classifier **Am17** performing authorship attribution with unseen APT groups is worse compared to the neural network-based classifier **Ro18**, which can be seen when Table 7.1 is compared to Table 6.1.

Dataset	Imbalanced	Undersampled	Oversampled
Extracted Cuckoo	0.340 (σ: 0.012)	0.396 (σ: 0.113)	**0.422 (σ: 0.105)**
Filtered Cuckoo	0.392 (σ: 0.116)	**0.467 (σ: 0.075)**	0.347 (σ: 0.019)
VirusTotal	0.595 (σ: 0.057)	**0.676 (σ: 0.014)**	0.651 (σ: 0.038)
FilteredVMRay	0.651 (σ: 0.028)	0.582 (σ: 0.112)	**0.688 (σ: 0.064)**

TABLE 7.2: **F1-Score** Results Evaluating **Am17** on Scenario A

The use of filtered VMRay reports leads to the best performance with a F1-score of 0.688, whereas Cuckoo reports lead to the worst results, with a maximum F1-score of 0.467. The use of undersampling and oversampling has a positive influence on the results. This could be due to the fact that random forest classifiers are more sensitive to imbalanced datasets. It is hard to tell why undersampling sometimes works better than oversampling and vice versa. Probably, reports that do not convey much information could better be oversampled since no information is lost, whereas reports containing too much detail could better be undersampled in order to reduce the information available and thus simplify the problem, although it is hard to substantiate this claim.

Dataset	Imbalanced	Undersampled	Oversampled
Extracted Cuckoo	0.651 (σ: 0.022)	0.603 (σ: 0.126)	**0.548 (σ: 0.104)**
Filtered Cuckoo	0.589 (σ: 0.152)	**0.510 (σ: 0.101)**	0.649 (σ: 0.026)
VirusTotal	0.405 (σ: 0.047)	**0.332 (σ: 0.012)**	0.358 (σ: 0.033)
FilteredVMRay	0.317 (σ: 0.031)	0.328 (σ: 0.098)	**0.287 (σ: 0.057)**

TABLE 7.3: **FP Rate** Results Evaluating **Am17** on Scenario A

Not only are the F1-scores low in this experiment, also the achieved false positive rates are high, as shown in Table 7.3. This means that the algorithm makes in some cases of this binary problem per class more false positives then true negatives and so fails to identify which samples do no belong to a certain actor.

When comparing the FPR results to the accuracy results in Table 7.1, we see that the accuracy results follow the patterns of the FPR results in the sense that results with a lower FPR have a higher accuracy and vice versa. This means, as explained in Section 6.2.1, that the accuracy results are not distorted by class imbalance.

Overall, the results of this experiment show that **Am17** performs mediocre as approach for authorship attribution with unseen APT groups. The accuracy results shown in Table 7.1 prove this as well, since the maximum accuracy achieved equals 0.741. This means that the algorithm can be used for experimental purposes, but it should not be used in non-experimental environments. The results regarding recall, precision and AUC can be found in Tables B.1 to B.3 respectively.

7.2.2 Country-Level Authorship Attribution with Earlier Seen APT Groups

The accuracy of the random forest classifier trained for country-level authorship attribution with earlier seen APT groups (thus belonging to Scenario B) is tested to be up to **98.4%**.

Dataset	Imbalanced	Undersampled	Oversampled
Extracted Cuckoo	**0.946 (σ: 0.008)**	0.907 (σ: 0.007)	0.937 (σ: 0.010)
Filtered Cuckoo	**0.949 (σ: 0.003)**	0.913 (σ: 0.021)	0.936 (σ: 0.007)
VirusTotal	0.976 (σ: 0.006)	0.957 (σ: 0.011)	**0.984 (σ: 0.006)**
FilteredVMRay	0.956 (σ: 0.007)	0.928 (σ: 0.009)	**0.964 (σ: 0.005)**

TABLE 7.4: **Accuracy** Results Evaluating **Am17** on Scenario B

A striking difference with the results presented earlier is that in this experiment, the Cuckoo reports perform nearly as good as the reports from VirusTotal and VMRay. It therefore seems that a RFC is more able to interpret the content of Cuckoo reports compared to a neural network, but it is difficult to explain why this is the case.

Dataset	Imbalanced	Undersampled	Oversampled
Extracted Cuckoo	**0.943 (σ: 0.009)**	0.904 (σ: 0.006)	0.929 (σ: 0.012)
Filtered Cuckoo	**0.947 (σ: 0.003)**	0.903 (σ: 0.023)	0.921 (σ: 0.007)
VirusTotal	0.971 (σ: 0.010)	0.955 (σ: 0.013)	**0.982 (σ: 0.007)**
FilteredVMRay	0.953 (σ: 0.009)	0.925 (σ: 0.012)	**0.964 (σ: 0.005)**

TABLE 7.5: **F1-Score** Results Evaluating **Am17** on Scenario B

Apart from this, we clearly see in Table 7.5 that the undersampled variants perform worse than the imbalanced and oversampled ones. This is due to, as explained in Section 6.2.2, the fact that undersampling leads to a very small dataset and thus a huge loss of information to learn from.

Dataset	Imbalanced	Undersampled	Oversampled
Extracted Cuckoo	**0.016 (σ: 0.002)**	0.024 (σ: 0.003)	0.017 (σ: 0.002)
Filtered Cuckoo	**0.015 (σ: 0.001)**	0.021 (σ: 0.004)	0.015 (σ: 0.002)
VirusTotal	0.007 (σ: 0.002)	0.011 (σ: 0.003)	**0.004 (σ: 0.002)**
FilteredVMRay	0.014 (σ: 0.002)	0.018 (σ: 0.002)	**0.011 (σ: 0.002)**

TABLE 7.6: **FP Rate** Results Evaluating **Am17** on Scenario B

The classifier is able to grasp per class which samples do not belong to the class, causing the classifier to perform well in the sense that the chance of accusing a country of authorship falsely is kept low. This is proved by the low false positive rates as shown in Table 7.6.

Moreover, the patterns in Tables 7.6 and 7.4 follow each other, meaning that the accuracy results, though susceptible to distortion due to class imbalance, show no signs of such distortion.

Ro18 is a good approach to perform authorship attribution with earlier seen APT groups. The maximum accuracy results range from 0.946 on the extracted arguments from Cuckoo reports up to 0.984 using VirusTotal reports, as shown in Table 7.4. This means that this classifier can be used in non-experimental environments. The results regarding recall, precision and AUC can be found in Tables B.4 to B.6 respectively.

7.2.3 APT Group-Level Authorship Attribution

The accuracy of the random forest classifier trained for APT group-level authorship attribution (thus belonging to Scenario C) is tested to be up to **98.1%**.

Dataset	Imbalanced	Undersampled	Oversampled	Am17 in [2]
Extracted Cuckoo	0.919 (σ: 0.012)	0.776 (σ: 0.023)	**0.923 (σ: 0.011)**	**0.930**
Filtered Cuckoo	**0.943 (σ: 0.012)**	0.833 (σ: 0.026)	0.932 (σ: 0.007)	-
VirusTotal	0.977 (σ: 0.002)	0.925 (σ: 0.006)	**0.981 (σ: 0.003)**	-
FilteredVMRay	0.948 (σ: 0.010)	0.840 (σ: 0.015)	**0.956 (σ: 0.003)**	-

TABLE 7.7: **Accuracy** Results Evaluating **Am17** on Scenario C

When comparing the results of our experiment to the results presented by Aman et al. in [2] using the extracted API calls from the Cuckoo reports, our results are worse compared to the results in [2]. However, results using VirusTotal reports are better than those presented by Aman et al., as shown in Table 7.7.

Dataset	Imbalanced	Undersampled	Oversampled	Am17 in [2]
Extracted Cuckoo	0.861 (σ: 0.007)	0.708 (σ: 0.017)	**0.864 (σ: 0.012)**	**0.931**
Filtered Cuckoo	0.912 (σ: 0.023)	0.805 (σ: 0.034)	**0.916 (σ: 0.011)**	-
VirusTotal	0.945 (σ: 0.012)	0.883 (σ: 0.021)	**0.953 (σ: 0.007)**	-
FilteredVMRay	0.901 (σ: 0.016)	0.785 (σ: 0.011)	**0.921 (σ: 0.008)**	-

TABLE 7.8: **F1-Score** Results Evaluating **Am17** on Scenario C

Moreover, we again notice the effect that undersampling has on this experiment, just like described in Section 6.2.3. However, the effect is much lower using a RFC-based classifier compared to a classifier based on a neural network. This could be caused by the fact that a RFC is able to cope better with less complex input data since the size of the trees may depend on the size of the input data, whereas the structure of the neural network is chosen beforehand. Once the input data is less complex or containing less information, it could be possible that a large neural network tends to seek complex relations and struggles to generalize its knowledge.

Dataset	Imbalanced	Undersampled	Oversampled	Am17 in [2]
Extracted Cuckoo	0.008 (σ: 0.001)	0.020 (σ: 0.002)	**0.007 (σ: 0.001)**	0.009
Filtered Cuckoo	**0.006 (σ: 0.001)**	0.015 (σ: 0.002)	**0.006 (σ: 0.001)**	-
VirusTotal	**0.002 (σ: 0.000)**	0.007 (σ: 0.000)	**0.002 (σ: 0.000)**	-
FilteredVMRay	0.005 (σ: 0.001)	0.015 (σ: 0.001)	**0.004 (σ: 0.000)**	-

TABLE 7.9: **FP Rate** Results Evaluating **Am17** on Scenario C

Since the number of classes and their distribution used in [2] and the dataset described in Chapter 4 differs (20 vs. 12 different classes), the false positive rates as presented in [2] may differ from the numbers achieved in this experiment. As we can see in Table 7.9, this is indeed the case, since our results for the imbalanced and oversampled cases achieve a lower FPR compared to the results in [2].

Overall, the results achieved on datasets other than the extracted API calls from Cuckoo reports are better than those presented in [2], the paper described **Am17**. This difference in performance could be due to the fact that the extracted API calls do not convey as much information as full sandbox reports, which can be substantiated by the results in Table 7.7. The best classifier has an accuracy up to 0.981, making it suitable for use in a non-experimental environment.

The results regarding recall, precision and AUC can be found in Tables B.7 to B.9 respectively.

7.3 Preliminary Conclusions

The following preliminary conclusions can be drawn from the evaluation of **Am17**:

- The tested performance of the RFC-based approach matches the performance achieved by **Am17** in the original paper. In fact, the tested performance is even better than described in [2]. A possible reason for this could be that malware written by different APT groups differs more strongly than different criminal-context malware families.
- The random forest classifier used is capable of finding features which serve as indicators of a country or APT group linked to a malware sample, since most classifiers score decently. This also proves the fact that such features indeed exist.
- Since the performance achieved on answering Scenario A goes up to 0.741, the algorithm must have grasped to some extent what patterns are characteristic for a certain country. This proves that even different malware developing groups within a country tend to use similar techniques, infrastructures or coding styles. Therefore, a trained RFC could be used to extract characteristics that provide specific clues to malware analysts about where to look for traces indicating the state behind the sample, even when the sample comes from an unseen APT group.
- The classifiers used for answering Scenarios B and C are so accurate, that they are suitable for everyday use. From this we may conclude that it is possible to perform authorship attribution on APT and country level using a RFC.

Chapter 8

Comparing the Two Approaches

In this chapter, a comparison is made between **Ro18** and **Am17**. Conclusions are drawn about the performance, advantages and disadvantages of both approaches, based on the results described in Chapter 6 and 7.

8.1 Comparison with Respect to Algorithm

When looking at the performance of **Ro18** and **Am17** in general, some major differences can be found.

Firstly, the time a RFC spends on training is several orders of magnitude lower compared to the NN. The average RFC takes at most several minutes to train, whereas the NN takes around 15-45 minutes. The speed of the neural network algorithm could drastically be improved by making it run on the GPU instead of the CPU [86]. Moreover, the size of the neural network could be reduced, decreasing the time needed for training as well.

Secondly, the NN performs generally speaking slightly better than the RFC. This difference in performance was expected, since it is generally assumed that a NN is more able to cope with complex problems compared to a RFC. However, the difference between the performance of the NN and RFC is not large. It could be possible that the problem is less complex than assumed (making the RFC able to cope better with the problem), because the sandbox reports could be giving too much information away. This effect is checked in Chapter 9 and discussed more extensively in Chapter 10.

8.2 Comparison with Respect to Sampling

Since the dataset is imbalanced, undersampling and oversampling is used to mitigate the effects of an imbalanced dataset, as described in Section 4.4. When looking at the influence of sampling on the results achieved, it is impossible to find the most optimal sampling strategy for all cases. In some cases, an undersampled set outperforms an oversampled set and vice versa.

Both effects can partially be explained by analyzing the *information entropy* of the sandbox reports. Whenever a report has a high entropy, it means most words in the report entail new, useful information, whereas in reports with a low entropy, most words do not provide new information, but can (partially) be predicted by looking at words mentioned earlier. We can therefore say that information entropy provides useful insights into the *information density* of sandbox reports.

In case a dataset with high entropy reports is undersampled, the amount of information that is removed is high. In this way, the classes may be balanced, but the

classifier is supplied a lot less information to learn from. Therefore, it could in such cases be more helpful to oversample the dataset. Although the entropy drops then, the amount of available information is retained and the classes are balanced.

On the other hand, in case the dataset has a low entropy, oversampling would lower the entropy even further. Although the classes are balanced then, it can become too hard for the classifier to find the right information in the dataset, which leads to overfitting. In such cases, it could be more helpful to undersample the dataset. This would mean that the absolute amount of information available is lowered, but the entropy remains equal.

A pattern that we noticed in the results, is that undersampling leads to higher deviation of the results. This is caused by the fact that the reduction of the dataset inevitably leads to the increase of variance in the results [87, 88].

8.3 Comparison with Respect to Metrics

An important metric to consider is accuracy, since it describes the overall correctness of the predictions made by the classifiers. The accuracy results are above 90% for all setups belonging to Scenarios B and C, with maximums per use case above 98% using VirusTotal reports. The experiments belonging to Scenario A achieved a lower accuracy: the maximum accuracies equal 76.6% (NN) and 74.1% (RFC). This makes all tested classifiers perform decently, but the classifier belonging to Scenario A is not suitable for non-experimental use due to its low accuracy.

For the specific use case of attributing malware in a nation-state context, it is generally speaking better to have a low false positive rate than a low false negative rate, since it is preferable to not accuse any actor at all than to accuse an actor falsely. Looking at the results of the evaluation of **Ro18** and **Am17**, a FPR of less than 1% is reached for the use cases belonging to Scenarios B and C, making them very suitable for the goal of authorship attribution in a nation-state context. In case of Scenario A, the FPR does not get lower than 28% (28.6% for the NN and 28.7% for the RFC).

8.4 Comparison with Respect to Sandboxes

In general, the use of VirusTotal reports leads to the best performance, whereas the use of filtered Cuckoo reports or arguments extracted from Cuckoo reports leads to the worst performance. Since the VirusTotal reports only contain the results of the scans of several anti-virus solutions, this result seems counter-intuitive. However, this could be due to the fact that VirusTotal reports could give away the classification labels by including the results of anti-virus file scans. Sometimes, the results of these scans contain direct links to the author of the malware. This effect is checked in Chapter 9 and discussed more extensively in Chapter 10.

Apart from that, the accuracy results achieved by only using the section containing the API calls from the Cuckoo reports are better than the results achieved by using the complete Cuckoo reports in Scenario A. This could be due to the fact that the extracted arguments contain some extra biased information as described in Section 4.2.3. Another reason could be that because a bag of words approach is used with a maximum size set to the used amount of words, the creation of a bag of words for the whole report could lead to a loss of details, since only n words are used. When only the API calls are used, there is a higher chance that n words are enough to capture details as well.

Chapter 9

Extracting Human-Interpretable Characteristics

When a classifier has been trained successfully, it contains knowledge which is used for classifying samples. When this knowledge would be extracted from the classifier, two important applications are made possible.

Firstly, new insights can be discovered from the relations learned by the classifier. Since the classifiers used in this research are capable of discovering complex relations involving multiple parameters, it could be possible that new characteristics of an APT group or actor are discovered that have not been noticed before by malware analysts. Such characteristics can then be further examined and verified. Moreover, it would be possible to automate the process of knowledge extraction and keep track of trends regarding the characteristics of an actor.

Secondly, the quality of the classifier can be examined by checking which features are important to the classifier, and by looking how the classifier uses the features presented to it. It could for example be possible that the classifier is paying attention to the wrong features. An occurrence of this problem is described in [89], where Shane noticed that Microsoft Azure's computer vision API seemed to classify images as containing sheep by solely looking at the type of landscape that was presented in the pictures. This meant that pictures with a certain type of landscape without any sheep were labeled as containing sheep, whereas pictures containing sheep in unusual places were not classified as containing sheep. Having such a problem would mean that the classifier needs to be retrained using a different setup.

We concluded in Section 8.4 that it could be possible that the classifiers take information from sandboxes into account that contain direct classifications or indicators of authorship, such as the description of triggered Yara-rules and the result of scans by anti-virus solutions. Therefore, we use this chapter to see to what extent this is the case. Moreover, we will briefly try to examine whether unknown characteristics of actors can be discovered.

We will use the RFC-based approach to extract knowledge from, since this is less complex compared to the neural network-based approach. Nevertheless, some information is provided on how knowledge extraction could possibly be achieved on a neural network.

9.1 Knowledge Extraction on Random Forest Classifiers

We use two approaches to extract information about the features used in the RFC. The first approach extracts all used features with their importances from a trained RFC, after which the most important features are selected. The second approach

involves the manual inspection of several decision trees inside the RFC, to see in which context the features are used and to what results they lead.

9.1.1 Feature Importance

As already mentioned in Section 3.1.3, a RFC is made up of multiple distinct decision trees, which are trained on slightly different subsets of all available features. Those trees are constructed during training as efficient as possible. Therefore, the decisions that are made in the top of the tree, are the most discriminating and therefore the most important decisions. The most important features of an entire RFC can easily be extracted using the function `get_feature_importances` in scikit-learn [77].

We trained the RFC again to solve the problem of APT group-level authorship attribution (belonging to Scenario C) using the imbalanced variants of the 4 different datasets. We did apply 5-fold cross-validation again, in order to get a better estimate of the most important features compared to training the algorithm once. This means that the algorithm is trained 5 times on the same problem, each time followed by the extraction of the most important features. After these 5 runs, the average importance is computed for all features. Finally, we choose to select the 50 most important features per dataset. The results can be found in Tables C.1 and C.2 in Appendix C.

The most noticeable result is that the list of most important features extracted from Cuckoo reports does not contain any direct links to malware actors or families, whereas the list of most important features using VirusTotal and VMRay does contain such links. Examples include references to Cloud Hopper (APT 10), MiniDuke (APT 29), Fanni (better known as Fannyworm by the Equation Group) and direct references to APT 10 and Winnti. The fact that these terms are in the list of most important features confirms the suspicion that there are direct identifiable links in the sandbox reports and that these terms are paid much attention to by the classifiers. This could lead to several problems, which we will discuss in Section 10.2. Future work could include the removal of direct links from sandbox reports and examination of its effect on the performance of the classifiers.

9.1.2 Manual Decision Tree Analysis

In order to perform manual analysis on the decision tree, we examine several trees from the list of trees given by the function `get_esitmators` in scikit-learn [77]. The extracted trees are visualized using graphviz [90].

The trees that we extracted, were build up from around 425 (filtered Cuckoo) to 650 nodes (extracted API calls from Cuckoo, VirusTotal and filtered VMRay), including the leaves. Many different features are being used, some features multiple times. An example of a part of a decision tree extracted from a RFC trained on a set containing filtered VMRay reports can be found in Figure D.1 in Appendix D.

The fact that some direct links are in the list of most important characteristics also comes back in the extracted trees from classifiers which have been trained on VirusTotal and VMRay reports. Figures 9.1 and 9.2 show that in a trained RFC decision trees exist which can detect almost an entire class based on a single feature. In this case, the Equation Group can easily be identified by the features `eqdrug` and `6eb00-b34d1daffa49b2f4c90841705b2c994563bde672bf35eb1c46cdb19a1ed`. The first feature is a direct identifier to the Equation Group from the results of a malware scan by AVG [91], the second feature is the value of the SHA-256 hash of a file dropped by the executed malware. Although the latter feature can be used to classify nearly a complete class at once, we do not consider it as direct feature, since a hash value

does not reveal the actor behind the sample in a direct way, unlike the first feature, where `eq` stands for Equation Group.

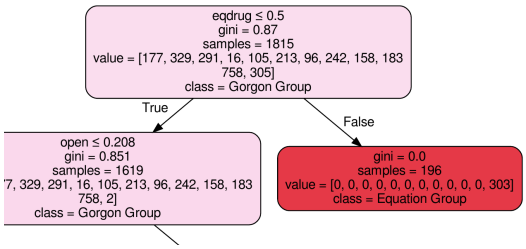

FIGURE 9.1: Top nodes of an extracted decision tree from a RFC trained on VirusTotal reports

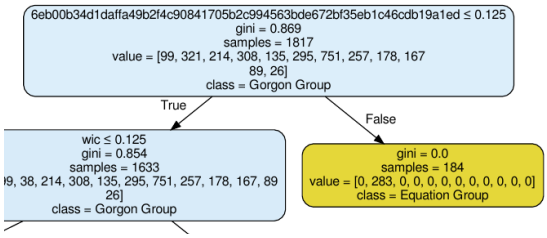

FIGURE 9.2: Top nodes of an extracted decision tree from a RFC trained on filtered VMRay reports

Not only do direct classifications occur at the top of the tree, the same phenomenon occurs occasionally somewhat deeper in the tree. Figure 9.3 shows a split that was encountered after 3 splits from the root node, where a feature called `apt30` is used to classify malware. This feature is based on the name of a Yara-rule identifying traffic from malware by APT 30.

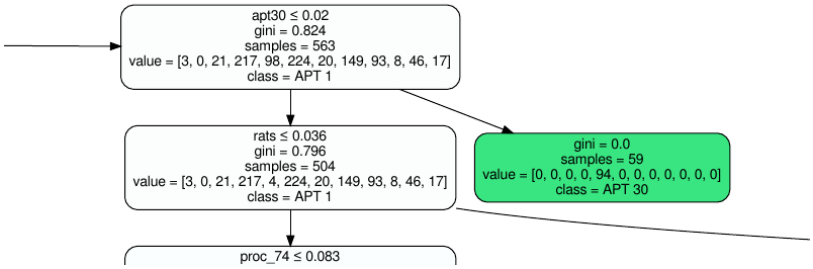

FIGURE 9.3: Split based on a feature directly linked to APT 30

Although the examples presented only show features that directly lead to the classification of almost an entire class, the far majority of features are common to different APT groups, since the decision trees contain many more nodes than the given number of classes.

A further examination of the fully extracted decision tree also reveals information about the extent to which different APT groups develop similar malware. Since all classes are assigned a color and all nodes have the same color as the majority class of the node, one can in an easy manner track visually what decisions lead to a certain class. When a branch located further down the tree is made up out of 2 colors, it means that the classifier has taken several decisions to get to the branch, and then needs several more decisions to distinguish the classes from each other. This could indicate that the APT groups are closely related to each other and hard to distinguish. On the other hand, two colors that never appear together in a lower branch, could indicate that the malware from the APT groups related to those colors is easy to distinguish.

When we apply this analysis method to the extracted decision tree, we may firstly conclude that the malware allegedly related to the Equation Group is one of its kind and to a small extent comparable with malware from Energetic Bear. This because the majority of the samples from the Equation Group is set apart after the

first decision and the remaining samples appear in the same branch as samples from Energetic Bear, taking multiple decisions to sort the malware of those APT groups out.

Malware samples belonging to the Gorgon Group appear in very many different branches in the tree, which indicates that it is hard to distinguish their samples from other APT groups. A possible explanation could be that they reuse malware from different actors, or that their malware samples have a great variety and are thus hard to characterize. The last theorem is substantiated in a blog from Unit42, which describes the Gorgon Group as a 'larger crew of individuals', performing both targeted as well as diverse criminal attacks using different crimeware families [92].

Another striking pattern is that the samples of APT 1 and Winnti are often in the same branches. This may be caused by the fact that Winnti could be related to the PLASSF, the Strategic Support Force of the People's Liberation Army. Since APT 1 is part of the PLA as well, actors may overlap. This theory is substantiated by 401TRG [93].

9.2 Knowledge Extraction on Neural Networks

Whereas it is easy to extract rules and features importances from a RFC, a neural network has a more complex architecture that is not suitable for easily extracting and understanding how a combination of different features adds up to the desired result.

Nevertheless, several approaches are described for performing analysis based on the weights and biases in a trained neural networks. One of the first approaches, described by Tsukimoto, is able to extract logical rules by approximating single units of a neural function as boolean functions [94]. One of the most recent approaches is described by Zilke et al. and is based on the use of decision trees to describe the working of the neural network. Their approach described as algorithm called DeepRED, which is designed to cope well with deep neural networks [95].

Both approaches do not contain an easy method for quickly determining the importance of features, which can be solved by sensitivity analysis. This involves leaving out different parts of the sandbox reports and providing these incomplete reports to a trained classifier. If the performance of the classifier remains stable, the features from the part that is left out are not very important. If the performance of the classifier drops, it is apparently dependent on the features that were left out. Nevertheless, this method does not provide an efficient and thorough analysis of the used features by the classifier, since it requires iterations to find out the importance of every single feature and does not easily provide information about features that only work well pairwise.

Chapter 10

Complications Faced and Discussion

10.1 Complications Faced

In the process of evaluating and comparing the methods **Ro18** and **Am17**, several complications were faced, mainly concerning the input data. The most important complications were as follows:

- **Lack of Reliable Sources:** Since state-sponsored APTs mostly use targeted attacks, malware samples are not as widely spread as 'regular' malware samples are [1, 7]. Moreover, samples can be kept secret by governments, making them unavailable for research until they occasionally get released [69]. This makes it more difficult to get enough samples to perform supervised learning on.
- **Unequal Distribution of Countries and APT Groups:** In order to properly train a supervised learning algorithm, the distribution of samples per countries and APT groups needs to be roughly uniform. Whenever this is not the case, a supervised learning algorithm performs sub-optimal [96, 97]. Although this problem can be mitigated, it is better to avoid imbalanced data sets at all. However, some actors are having more aggressive malware campaigns and are so way more on the radar (e.g. Russia or China) in comparison to other actors (e.g. Iran or the USA). Given this fact, it is harder to have an equal distribution of countries and APT groups.
- **No Strict Separation of APT Groups:** As described in Section 5.1, the strict separation imposes an additional requirement to the input data. Because of the lack of reliable sources and the difficulty of getting an equal distribution of countries and APT groups, it is harder to meet this additional requirement.
- **Diversity of Measures:** The measures that are used in the papers describing **Ro18** and **Am17** are different, causing a first proper comparison based on literature only to be infirm. Especially a proper evaluation of **Ro18** is made more difficult, since hardly any results are available.

10.2 Discussion

Although the results of the conducted experiments largely confirm the results presented in [1] and [2], we discuss several remarks to put the results into context.

Firstly, two assumptions are made in Section 4.3 to make it feasible to reason about attribution. One of our assumptions states that a malware sample is used by

at most one actor, enabling the possibility to label a single APT group to a malware sample as responsible group. However, real-world scenarios may include a malware sample or family that is used by multiple actors. For example, several core elements of a piece of malware could have been bought on the dark web or stolen from other actors [21]. Therefore, attribution must be performed with great care, since these assumption do not always hold in reality and actors do their utmost to add misleading traces.

Secondly, the dataset described in Chapter 4 is based on claims by anti-virus companies and malware researchers. Although most claims about authorship are substantiated in corresponding malware analysis reports, they cannot be proved with irrefutable evidence. Therefore, ground truth of the classifier is based on the 'beliefs and reasons' of the authors of the malware analysis reports.

Moreover, the evaluated approaches **Ro18** and **Am17** make use of sandbox reports for classification. Though sandbox reports are a great asset to easily extract numerous malware characteristics [20], they could also contain information that makes the classification biased. Looking at the contents of the report, we noted that the description of Yara-rules and the detection information of anti-virus solutions contains information about the alleged actor in some cases. It could be possible that the classifier pays much attention to these sections in the sandbox reports. This raises two problems:

1. The classifier is not looking at characteristics of the samples anymore, but at claims that are done by analysts or anti-virus companies, making the results biased. Moreover, the classifier would be incapable of solving the real problem of classification, since it still requires (manual) analysis by others.

2. The classifier gets dependent on the quality of the anti-virus solutions and Yara-rules that are contained in the sandboxes. When novel and not earlier analyzed malware would be classified, these sections of the sandbox reports would not contain as much useful information as in cases where the malware is known to the anti-virus solutions. In such cases, it could become possible that the quality of the classifier becomes unsatisfactory.

Apart from that, much information is thrown away using the bag of words approach, since it removes all context from a sandbox report. Whenever a JSON sandbox reports contains boolean information for example, the boolean values are separated from the property, making them provide almost no information at all. Moreover, the use of a bag of words involves for practical reasons an upper boundary n to the number of words used. When a bag of words is created for a whole sandbox report, this could lead to a larger loss of detail (since the information does not fit in n words) compared to the creation of a bag of words on only a small section of the report (where the information can be expressed using n words).

In the end, it is questionable how useful the obtained results are for mitigating threats or obtaining in-depth threat detail. Although the classifier reaches a high accuracy, it is hard to extract usable, human-interpretable knowledge from a trained classifier. In case of random forest classifiers, it is easy to visualize the underlying decision trees, but it is hard to extract high-level characteristics about different actors. For neural networks, it is even harder to extract knowledge, since the only two techniques known are sensitivity analysis and IF-THEN-rule extraction by an algorithm like DeepRED. Furthermore, many features like the ones described in Sections 2.3.1 and 2.3.2 are not contained in the sandbox reports described in Section 5.3, which could lead to superficial classification since the classifier could lack detailed information to learn from.

Chapter 11

Conclusions and Future Work

11.1 Conclusions

During the literature study, it has become clear that the topic of authorship attribution and malware family classification has been covered in literature, but that the amount of research is limited. This is possibly due to the fact that the topic is quite specific and plays a smaller role in academia compared to other types of institutions, like banking, defense or government institutions or business parties like anti-virus vendors.

Authorship attribution is a problem which involves different legal and technical complications. Legal complications include the fact that the level of proof required before taking further steps may vary, based on the fact whether it regards criminal cases or cases regarding the involvement of nation-states. Technical complications, which inhibit both attribution and classification, include the unavailability of source code and the fact that different techniques are applied to malware to hide its origin and intend. The possible presence of fake traces which are added to trick analysts in drawing faulty conclusions makes the problem even more complex.

Different technical means are available to perform malware family classification and authorship attribution on malware binaries. Also the use of supervised learning for family classification or attribution is described. Of all approaches that use supervised learning, the ones proposed by Rosenberg et al. (**Ro18**) and Aman et al. (**Am17**) are the two most promising state of the art supervised learning approaches to solve malware classification problems using raw input. Therefore, we selected these approaches and compared and evaluated them in this thesis, conducting a first evaluation and comparison of both algorithms.

There are no labeled datasets available which contain state-sponsored malware from different actors. Therefore, we came up with our own dataset, which will become the first publicly available dataset of its kind. This dataset is constructed using a yet never scientifically described technique involving threat intelligence reports. Such reports contain hashes of different malware samples, together with an analysis about the alleged actor behind the samples. By collecting numerous reports, a list of hashes of samples is constructed. This list is then submitted to VirusTotal in order to retrieve the corresponding samples, resulting in a dataset of 3,594 samples from 12 different APT groups, linked to 5 different countries. Used this dataset, we answered the 3 research questions:

Which supervised learning techniques are optimal for performing malware authorship attribution?
Both methods **Ro18** and **Am17** perform well on our dataset, retrieving similar results as the papers describing the approaches. In the cases where the classifier needs to

identify APT groups, the accuracy reaches over 98%, just as in case where the classifier needs to identify countries based on samples from earlier seen APT groups. The difference between the NN-based method **Ro18** and the RFC-based method **Am17** is the fact that in our setup, the RFC takes significantly less time to train, whereas the neural network achieves slightly better results. Therefore, we conclude that a method based on a RFC is preferred for performing first experiments and in applications which demand fast training. We suggest to use an approach based on a neural network when the setup is not hindered by time and computational power constraints and high accuracy plays an important role.

What type of dataset is needed for reaching satisfying attribution performance?
Although the used algorithm and its configuration plays a big role in the performance of the algorithm, the input data is of crucial importance since all decisions are based on this data. We evaluated the two algorithms on 4 different types of reports from 3 different sandboxes. The use of VirusTotal reports results in the highest accuracy, but since the reports from VirusTotal and VMRay include direct linkable data (e.g. the name of the APT group or malware family), the results are likely to be distorted to some extent. Therefore, we conclude that the dataset needed for reaching satisfying attribution performance should consist of (a combination of) sandbox reports without identifiable data, like the reports from Cuckoo.

To what extent can intelligible author characteristics be derived from a trained supervised learning algorithm?
Intelligible author characteristics can to some extent be derived from a trained supervised learning algorithm. When one wants to have insight into the way the classifier makes it decisions, one can perform analysis on the decision trees inside a RFC or the weight-matrix inside a neural network in order to see on basis of what steps or calculations the classifier established its decision. In some cases, this could lead to finding some characteristics about the actor of malware family, e.g. the act that every sample drops the same executable on the infected system. Moreover, one can derive from the decision trees inside a RFC to what extent the malware from different APT groups is similar. However, it is hard to extract high-level characteristics like the preference for a certain cryptographic mechanism or a default purpose of the sample. Since there are still multiple techniques to extract characteristics apart from the methods used by us, we conclude that further research is needed to fully answer this question.

11.2 Future Work

Although this thesis contains numerous conclusions as mentioned above, some limitations and inchoate ideas are encountered as well. These limitations and ideas could be explored further in future work. The most important topics for future work include:

- **Finding the Optimal Preprocessing Strategy:** Many choices are made with respect to preprocessing, which influence the performance of the classifier. For example, the choice is made to use a non-binary bag of words and to treat all reports separately. However, it would be good to explore what the effects are on the performance of the classifier when a binary bag of words approach would be used or when all sandbox reports are combined. Moreover, the size

of the bag of words could be changed to see what size causes the best performance. An approach where the keys and values of a JSON-file are treated as one word could be tested as well, since this will likely reduce the loss of information. Also, looking at a more fundamental level, the use of a bag of words could be replaced by another method, like tf-idf.

- **Improve the Quality of Sandbox Reports:** Not all sandbox executions that are performed, represent the behavior of the malware well. It could be that the execution of malware is hindered by evasion techniques or an incompatible sandbox configuration. The detection and prevention (e.g. by unpacking packed samples before feeding them to the sandbox or reducing the amount of obfuscation) of unsuccessful sandbox runs could lead to better results. Moreover, as mentioned before, some parts of the sandbox reports contain information provided by third parties about the malware sample, like Yara-rules and anti-virus solutions scans. These parts could be removed, to see how the classifier performs without clues provided from external sources.

- **Extending the Dataset:** Although the dataset used contains over 3,000 state-sponsored samples from different countries, additional samples could still be useful to examine the performance of the classifier on a more diverse dataset. Therefore, further research could include experiments with a dataset that also includes samples from other nation-states, such as Iran. Furthermore, malware used in criminal context (e.g. the samples contained in [98]), or benign samples could be added to the dataset. This would provide the opportunity to see whether malware detection and classification can at once be performed using a single classifier.

- **Include More Features:** Instead of only using sandbox reports, multiple other features can be used as well as input data. Features from e.g. [7], [44] or [46] could be involved, to see if the information gained from the use of these features leads to better performance.

- **Extract More Knowledge from Trained Classifiers:** The problem of authorship attribution remains a difficult task to perform, since it is extremely hard to point to a specific individual or organization without having lots of secondary resources like customer databases of ISPs. Moreover, this thesis did not perform an in-depth analysis of the knowledge inside the neural network as well as the random forest classifier. Therefore, future work could include a further investigation of the knowledge gained by the classifiers. Moreover, research could be performed to come up with a tool that automatically extracts all important features involving traces of authorship in order to keep track of development with respect to the characteristics of authors.

- **Usage of Unsupervised Learning:** One big issue is the lack of open, sound and labeled malware data. Since unsupervised learning approaches are able to find (meaningful) patterns in unlabeled data, it is useful to find out how effective such methods are in this domain.

- **Solving Related Problems Using Transfer Learning:** Once a neural network is trained, it is possible to use transfer learning to use the learned information for a slightly different task. The example that is used in [1], the paper describing **Ro18**, involved retraining the model for classifying samples on basis of family instead of actor. It would be good to examine whether transfer learning can be

used to solve related problems, like classifying on the behavior of malware, its targets or e.g. intrusion mechanisms used by the malware.

Bibliography

[1] Ishai Rosenberg, Guillaume Sicard, and Eli David. "End-to-End Deep Neural Networks and Transfer Learning for Automatic Analysis of Nation-State Malware". In: *Entropy* 20.5 (May 2018), p. 390. ISSN: 1099-4300. DOI: 10.3390/e20050390.

[2] Naqqash Aman, Yasir Saleem, Fahim H. Abbasi, and Farrukh Shahzad. "A Hybrid Approach for Malware Family Classification". In: *Applications and Techniques in Information Security*. Springer, 2017, pp. 169–180. ISBN: 978-981-10-5421-1. URL: https://link.springer.com/chapter/10.1007/978-981-10-5421-1_14.

[3] O. E. David and N. S. Netanyahu. "DeepSign: Deep learning for automatic malware signature generation and classification". In: *2015 International Joint Conference on Neural Networks (IJCNN)*. July 2015, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280815.

[4] *Advanced Persistent Threat Groups*. URL: https://www.fireeye.com/current-threats/apt-groups.html.

[5] Aylin Caliskan Islam et al. "When Coding Style Survives Compilation: De-anonymizing Programmers from Executable Binaries". In: *2018 Network and Distributed System Security Symposium (NDSS)* (Dec. 2017). arXiv: 1512.08546v3.

[6] Jawwad A. Shamsi, Sherali Zeadally, Fareha Sheikh, and Angelyn Flowers. "Attribution in cyberspace: techniques and legal implications". In: *Security and Communication Networks* 9.15 (2016), pp. 2886–2900. DOI: 10.1002/sec.1485.

[7] Morgan Marquis-Boire, Marion Marschalek, and Claudio Guarnieri. "Big game hunting: The peculiarities in nation-state malware research". In: *Black Hat, Las Vegas, NV, USA* (2015). URL: https://blackhat.com/docs/us-15/materials/us-15-MarquisBoire-Big-Game-Hunting-The-Peculiarities-Of-Nation-State-Malware-Research.pdf.

[8] Saed Alrabaee, Paria Shirani, Mourad Debbabi, and Lingyu Wang. "On the Feasibility of Malware Authorship Attribution". In: *Foundations and Practice of Security*. Springer International Publishing, 2017, pp. 256–272. ISBN: 978-3-319-51966-1.

[9] S. Swain, G. Mishra, and C. Sindhu. "Recent approaches on authorship attribution techniques — An overview". In: *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*. Vol. 1. Apr. 2017, pp. 557–566. DOI: 10.1109/ICECA.2017.8203599.

[10] Clement Guitton. "Modelling Attribution". In: *Proceedings of the 12th European Conference on Information Warfare and Security*. Vol. 1. 2013, pp. 91–97. URL: https://cryptome.org/2014/12/ECIW2013.pdf.

[11] David D. Clark and Susan Landau. "Untangling attribution". In: *Harvard National Security Journal* 2 (2011), p. 323. URL: http://harvardnsj.org/2011/03/untangling-attribution-2/.

[12] Clement Guitton and Elaine Korzak. "The Sophistication Criterion for Attribution". In: *The RUSI Journal* 158.4 (2013), pp. 62–68. DOI: 10.1080/03071847.2013.826509.

[13] Jason Healey. *Beyond attribution: Seeking national responsibility for cyber attacks*. Atlantic Council of the United States, 2012. URL: http://www.atlanticcouncil.org/images/files/publication_pdfs/403/022212_ACUS_NatlResponsibilityCyber.pdf.

[14] Efstathios Stamatatos. "A survey of modern authorship attribution methods". In: *Journal of the American Society for Information Science and Technology* 60.3 (Dec. 2008), pp. 538–556. DOI: 10.1002/asi.21001.

[15] Pieter Arntz. *Explained: Packer, Crypter, and Protector*. Mar. 2017. URL: https://blog.malwarebytes.com/cybercrime/malware/2017/03/explained-packer-crypter-and-protector/.

[16] P. OKane, S. Sezer, and K. McLaughlin. "Obfuscation: The Hidden Malware". In: *IEEE Security Privacy* 9.5 (Sept. 2011), pp. 41–47. ISSN: 1540-7993. DOI: 10.1109/MSP.2011.98.

[17] S. Cesare, Y. Xiang, and W. Zhou. "Malwise – An Effective and Efficient Classification System for Packed and Polymorphic Malware". In: *IEEE Transactions on Computers* 62.6 (June 2013), pp. 1193–1206. ISSN: 0018-9340. DOI: 10.1109/TC.2012.65.

[18] Kaiyuan Kuang et al. "Enhance virtual-machine-based code obfuscation security through dynamic bytecode scheduling". In: *Computers & Security* 74 (2018), pp. 202 –220. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2018.01.008.

[19] Thomas Roccia. *Witepaper: An Overview of Malware Self-Defense and Protection*. Dec. 2016. URL: https://securingtomorrow.mcafee.com/mcafee-labs/overview-malware-self-defence-protection/.

[20] M. Vasilescu, L. Gheorghe, and N. Tapus. "Practical malware analysis based on sandboxing". In: *2014 RoEduNet Conference 13th Edition: Networking in Education and Research Joint Event RENAM 8th Conference*. Sept. 2014, pp. 1–6. DOI: 10.1109/RoEduNet-RENAM.2014.6955304.

[21] Brian Bartholomew and Juan Andres Guerrero-Saade. "Wave your false flags! Deception tactics muddying attribution in targeted attacks". In: *Virus Bulletin Conference*. 2016. URL: https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2017/10/20114955/Bartholomew-GuerreroSaade-VB2016.pdf.

[22] A. Pfeffer et al. "Malware Analysis and attribution using Genetic Information". In: *2012 7th International Conference on Malicious and Unwanted Software*. Oct. 2012, pp. 39–45. DOI: 10.1109/MALWARE.2012.6461006.

[23] M. Alazab, S. Venkataraman, and P. Watters. "Towards Understanding Malware Behaviour by the Extraction of API Calls". In: *2010 Second Cybercrime and Trustworthy Computing Workshop*. July 2010, pp. 52–59. DOI: 10.1109/CTC.2010.8.

[24] M. Sikorski and A. Honig. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press Series. No Starch Press, 2012. ISBN: 9781593272906. URL: https://books.google.nl/books?id=DhuTduZ-pc4C.

[25] Kyoung Soo Han, Boojoong Kang, and Eul Gyu Im. "Malware Classification Using Instruction Frequencies". In: *Proceedings of the 2011 ACM Symposium on Research in Applied Computation*. RACS '11. ACM, 2011, pp. 298–300. DOI: 10.1145/2103380.2103441.

[26] R. Tian, L. M. Batten, and S. C. Versteeg. "Function length as a tool for malware classification". In: *2008 3rd International Conference on Malicious and Unwanted Software (MALWARE)*. 2008, pp. 69–76. DOI: 10.1109/MALWARE.2008.4690860.

[27] V. Sai Sathyanarayan, Pankaj Kohli, and Bezawada Bruhadeshwar. "Signature Generation and Detection of Malware Families". In: *Information Security and Privacy*. Springer, 2008, pp. 336–349. ISBN: 978-3-540-70500-0.

[28] William Hardy et al. "DL4MD: A deep learning framework for intelligent malware detection". In: *Proceedings of the International Conference on Data Mining (DMIN)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 2016, p. 61.

[29] Wenyi Huang and Jack W. Stokes. "MtNet: A Multi-Task Neural Network for Dynamic Malware Classification". In: Springer, July 2016, pp. 399–418. ISBN: 978-3-319-40667-1. URL: https://www.microsoft.com/en-us/research/publication/mtnet-multi-task-neural-network-dynamic-malware-classification/.

[30] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. "Automatic Analysis of Malware Behavior Using Machine Learning". In: *J. Comput. Secur.* 19.4 (Dec. 2011), pp. 639–668. ISSN: 0926-227X. URL: http://dl.acm.org/citation.cfm?id=2011216.2011217.

[31] Boojoong Kang et al. "Malware Classification Method via Binary Content Comparison". In: *Proceedings of the 2012 ACM Research in Applied Computation Symposium*. RACS '12. San Antonio, Texas: ACM, 2012, pp. 316–321. ISBN: 978-1-4503-1492-3. DOI: 10.1145/2401603.2401672.

[32] J. Upchurch and X. Zhou. "Malware provenance: code reuse detection in malicious software at scale". In: *2016 11th International Conference on Malicious and Unwanted Software (MALWARE)*. Oct. 2016, pp. 1–9. DOI: 10.1109/MALWARE.2016.7888735.

[33] Abdurrahman Pektaş, M Eriş, and Tankut Acarman. "Proposal of n-gram based algorithm for malware classification". In: *SECURWARE 2011 - 5th International Conference on Emerging Security Information, Systems and Technologies*. Jan. 2011, pp. 14–18. URL: https://pdfs.semanticscholar.org/be3b/d545fba1cf243e9b06a9b0e906d431841cae.pdf.

[34] Xin Hu, Tzi-cker Chiueh, and Kang G. Shin. "Large-scale Malware Indexing Using Function-call Graphs". In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. CCS '09. ACM, 2009, pp. 611–620. DOI: 10.1145/1653662.1653736.

[35] Q. Zhang and D. S. Reeves. "MetaAware: Identifying Metamorphic Malware". In: *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. Dec. 2007, pp. 411–420.

[36] Silvio Cesare and Yang Xiang. "Classification of Malware Using Structured Control Flow". In: *Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing - Volume 107*. AusPDC '10. Australian Computer Society, Inc., 2010, pp. 61–70. URL: http://dl.acm.org/citation.cfm?id=1862294.1862301.

[37] Joris Kinable and Orestis Kostakis. "Malware classification based on call graph clustering". In: *Journal in Computer Virology* 7.4 (Nov. 2011), pp. 233–245. ISSN: 1772-9904. DOI: 10.1007/s11416-011-0151-y.

[38] Deguang Kong and Guanhua Yan. "Discriminant Malware Distance Learning on Structural Information for Automated Malware Classification". In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '13. ACM, 2013, pp. 1357–1365. DOI: 10.1145/2487575.2488219.

[39] Saed Alrabaee, Paria Shirani, Lingyu Wang, and Mourad Debbabi. "SIGMA: A Semantic Integrated Graph Matching Approach for identifying reused functions in binary code". In: *Digital Investigation* 12 (2015). DFRWS 2015 Europe, S61 –S71. ISSN: 1742-2876. DOI: 10.1016/j.diin.2015.01.011.

[40] KyoungSoo Han, Jae Hyun Lim, and Eul Gyu Im. "Malware Analysis Method Using Visualization of Binary Files". In: *Proceedings of the 2013 Research in Adaptive and Convergent Systems*. RACS '13. ACM, 2013, pp. 317–321. DOI: 10.1145/2513228.2513294.

[41] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. "Malware Images: Visualization and Automatic Classification". In: *Proceedings of the 8th International Symposium on Visualization for Cyber Security*. VizSec '11. ACM, 2011, 4:1–4:7. DOI: 10.1145/2016904.2016908.

[42] Lakshmanan Nataraj, Vinod Yegneswaran, Phillip Porras, and Jian Zhang. "A Comparative Assessment of Malware Classification Using Binary Texture Analysis and Dynamic Analysis". In: *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*. AISec '11. ACM, 2011, pp. 21–30. DOI: 10.1145/2046684.2046689.

[43] M. Kalash et al. "Malware Classification with Deep Convolutional Neural Networks". In: *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. Feb. 2018, pp. 1–5. DOI: 10.1109/NTMS.2018.8328749.

[44] Michael Bailey et al. "Automated Classification and Analysis of Internet Malware". In: *Recent Advances in Intrusion Detection*. Springer, 2007, pp. 178–197. ISBN: 978-3-540-74320-0.

[45] Avi Pfeffer et al. "Artificial Intelligence Based Malware Analysis". In: *arXiv preprint* (2017). arXiv: 1704.08716.

[46] Mansour Ahmadi et al. "Novel feature extraction, selection and fusion for effective malware family classification". In: *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. 2016, pp. 183–194. arXiv: 1511.04317v2.

[47] Nathan Rosenblum, Xiaojin Zhu, and Barton P. Miller. "Who Wrote This Code? Identifying the Authors of Program Binaries". In: *Computer Security – ESORICS 2011*. Springer, 2011, pp. 172–189. ISBN: 978-3-642-23822-2.

[48] Saed Alrabaee et al. "OBA2: An Onion approach to Binary code Authorship Attribution". In: *Digital Investigation* 11 (2014). Proceedings of the First Annual DFRWS Europe, S94 –S103. ISSN: 1742-2876. DOI: 10.1016/j.diin.2014.03.012.

[49] Kaspersky Lab. *Whitepaper: Machine Learning for Malware Detection*. 2017. URL: https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf.

[50] Chinmayee Annachhatre, Thomas H. Austin, and Mark Stamp. "Hidden Markov models for malware classification". In: *Journal of Computer Virology and Hacking Techniques* 11.2 (May 2015), pp. 59–73. ISSN: 2263-8733. DOI: 10.1007/s11416-014-0215-x.

[51] *Cuckoo Sandbox - Automated Malware Analysis*. URL: https://cuckoosandbox.org.

[52] Kevin K Dobbin and Richard M Simon. "Optimally splitting cases for training and testing high dimensional classifiers". In: *BMC medical genomics* 4.1 (2011), p. 31.

[53] Ron Kohavi et al. "A study of cross-validation and bootstrap for accuracy estimation and model selection". In: *Ijcai*. Vol. 14. 2. Montreal, Canada. 1995, pp. 1137–1145.

[54] Juan D Rodriguez, Aritz Perez, and Jose A Lozano. "Sensitivity analysis of k-fold cross validation in prediction error estimation". In: *IEEE transactions on pattern analysis and machine intelligence* 32.3 (2010), pp. 569–575.

[55] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. "Supervised machine learning: A review of classification techniques". In: *Emerging artificial intelligence applications in computer engineering* 160 (2007), pp. 3–24.

[56] Douglas M Hawkins. "The Problem of Overfitting". In: *Journal of Chemical Information and Computer Sciences* 44.1 (2004), pp. 1–12.

[57] S. Narayan and G. Tagliarini. "An analysis of underfitting in MLP networks". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2. July 2005, 984–988 vol. 2. DOI: 10.1109/IJCNN.2005.1555986.

[58] Kunihiko Fukushima and Sei Miyake. "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition". In: *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[59] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), p. 436.

[60] Dejan Tanikiç and Vladimir Despotovic. "Artificial intelligence techniques for modelling of temperature in the metal cutting process". In: *Metallurgy-Advances in Materials and Processes*. InTech, 2012.

[61] Richard Lippmann. "An introduction to computing with neural nets". In: *IEEE Assp magazine* 4.2 (1987), pp. 4–22.

[62] Lutz Prechelt. "Early stopping-but when?" In: *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[63] Yixin Luo and Fan Yang. *Deep Learning with Noise*. 2014. URL: https://pdfs.semanticscholar.org/d79b/a428e1cf1b8aa5d320a93166315bb30b4765.pdf.

[64] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[65] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *Journal of Machine Learning Research* 13.02 (2012), pp. 281–305.

[66] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[67] S. R. Safavian and D. Landgrebe. "A survey of decision tree classifier methodology". In: *IEEE Transactions on Systems, Man, and Cybernetics* 21.3 (May 1991), pp. 660–674. ISSN: 0018-9472. DOI: 10.1109/21.97458.

[68] Laura Elena Raileanu and Kilian Stoffel. "Theoretical Comparison between the Gini Index and Information Gain Criteria". In: *Annals of Mathematics and Artificial Intelligence* 41.1 (May 2004), pp. 77–93. ISSN: 1573-7470. DOI: 10.1023/B:AMAI.0000018580.96245.c6.

[69] Lee Mathews. *U.S. Cyber Command Shares Malware Samples To Help Thwart Bad Actors*. Nov. 2018. URL: https://www.forbes.com/sites/leemathews/2018/11/08/u-s-cyber-command-shares-malware-samples-to-help-thwart-bad-actors.

[70] F-Secure. *Whitepaper: The Dukes: 7 Years of Russian Cyberespionage*. 2015. URL: https://www.f-secure.com/documents/996508/1030745/dukes_whitepaper.pdf.

[71] Kaspersky Lab. *Whitepaper: The NetTraveler (aka 'TravNet')*. 2013. URL: https://kasperskycontenthub.com/wp-content/uploads/sites/43/vlpdfs/kaspersky-the-net-traveler-part1-final.pdf.

[72] Florian Roth. *APT Groups and Operations*. URL: http://apt.threattracking.com/.

[73] *ThreatMiner.org*. URL: https://www.threatminer.org/.

[74] *VirusTotal*. URL: https://www.virustotal.com/.

[75] *VMRay - Automated Malware Analysis & Detection Solutions*. URL: https://www.vmray.com/.

[76] *jq, a lightweight and flexible command-line JSON processor*. URL: https://stedolan.github.io/jq/.

[77] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[78] Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al. "Handling imbalanced datasets: A review". In: *GESTS International Transactions on Computer Science and Engineering* 30.1 (2006), pp. 25–36.

[79] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[80] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. "ADASYN: Adaptive synthetic sampling approach for imbalanced learning". In: *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE. 2008, pp. 1322–1328.

[81] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning". In: *Journal of Machine Learning Research* 18.17 (2017), pp. 1–5. URL: http://jmlr.org/papers/v18/16-365.html.

[82] Robin Genuer, Jean-Michel Poggi, and Christine Tuleau-Malot. "Variable selection using random forests". In: *Pattern Recognition Letters* 31.14 (2010), pp. 2225–2236.

[83] François Chollet et al. *Keras*. https://keras.io. 2015.

[84] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[85] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, pp. 2951–2959. URL: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf.

[86] John Lawrence et al. "Comparing TensorFlow Deep Learning Performance Using CPUs, GPUs, Local PCs and Cloud". In: (2017).

[87] Damien Brain and Geoffrey I Webb. "On the effect of data set size on bias and variance in classification learning". In: *Proceedings of the Fourth Australian Knowledge Acquisition Workshop, University of New South Wales*. 1999, pp. 117–128.

[88] Andrea Dal Pozzolo, Olivier Caelen, and Gianluca Bontempi. "When is undersampling effective in unbalanced classification tasks?" In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2015, pp. 200–215.

[89] Janelle Shane. *Do neural nets dream of electric sheep?* Mar. 2018. URL: http://aiweirdness.com/post/171451900302/do-neural-nets-dream-of-electric-sheep.

[90] John Ellson et al. "Graphviz — open source graph drawing tools". In: *Lecture Notes in Computer Science*. Springer-Verlag, 2001, pp. 483–484.

[91] Sophos. *Troj/Eqdrug-G*. URL: https://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/Troj~Eqdrug-G.aspx.

[92] Robert Falcone, David Fuertes, Josh Grunzweig, and Kyle Wilhoit. *The Gorgon Group: Slithering Between Nation State and Cybercrime*. 2018. URL: https://unit42.paloaltonetworks.com/unit42-gorgon-group-slithering-nation-state-cybercrime/.

[93] Tom Hegel. *Burning Umbrella: An Intelligence Report on the Winnti Umbrella and Associated State-Sponsored Attackers*. 2018. URL: https://401trg.com/burning-umbrella/.

[94] H. Tsukimoto. "Extracting rules from trained neural networks". In: *IEEE Transactions on Neural Networks* 11.2 (Mar. 2000), pp. 377–389. ISSN: 1045-9227. DOI: 10.1109/72.839008.

[95] Jan Ruben Zilke, Eneldo Loza Mencía, and Frederik Janssen. "DeepRED – Rule Extraction from Deep Neural Networks". In: *Discovery Science*. Springer, 2016. ISBN: 978-3-319-46307-0.

[96] Nitesh V. Chawla, Nathalie Japkowicz, and Aleksander Kotcz. "Editorial: Special Issue on Learning from Imbalanced Data Sets". In: *SIGKDD Explor. Newsl.* 6.1 (June 2004), pp. 1–6. ISSN: 1931-0145. DOI: 10.1145/1007730.1007733.

[97] H. He and E. A. Garcia. "Learning from Imbalanced Data". In: *IEEE Transactions on Knowledge and Data Engineering* 21.9 (Sept. 2009), pp. 1263–1284. ISSN: 1041-4347. DOI: 10.1109/TKDE.2008.239.

[98] Royi Ronen et al. "Microsoft Malware Classification Challenge". In: *arXiv preprint* (2018). arXiv: 1802.10135.

Appendix A

Additional Results of the Neural Network-based Approach

The appendix consists of the recall, precision and AUC results of all experiments conducted in Chapter 6.

A.1 Country-Level Authorship Attribution with Unseen APT Groups

Dataset	Imbalanced	Undersampling	Oversampling
Extracted Cuckoo	0.583 (σ: 0.020)	**0.649 (σ: 0.113)**	0.633 (σ: 0.050)
Filtered Cuckoo	0.342 (σ: 0.033)	**0.410 (σ: 0.136)**	0.352 (σ: 0.055)
VirusTotal	0.614 (σ: 0.067)	**0.660 (σ: 0.029)**	0.616 (σ: 0.083)
FilteredVMRay	0.681 (σ: 0.094)	**0.714 (σ: 0.114)**	0.649 (σ: 0.071)

TABLE A.1: **Recall** Results Evaluating **Ro18** on Scenario A

Dataset	Imbalanced	Undersampling	Oversampling
Extracted Cuckoo	0.570 (σ: 0.017)	**0.635 (σ: 0.104)**	0.624 (σ: 0.053)
Filtered Cuckoo	0.351 (σ: 0.025)	**0.425 (σ: 0.125)**	0.362 (σ: 0.052)
VirusTotal	0.660 (σ: 0.091)	**0.691 (σ: 0.026)**	0.651 (σ: 0.089)
FilteredVMRay	**0.728 (σ: 0.071)**	0.667 (σ: 0.166)	0.679 (σ: 0.045)

TABLE A.2: **Precision** Results Evaluating **Ro18** on Scenario A

Dataset	Imbalanced	Undersampling	Oversampling
Extracted Cuckoo	0.581 (σ: 0.029)	0.578 (σ: 0.149)	**0.690 (σ: 0.058)**
Filtered Cuckoo	**0.417 (σ: 0.059)**	0.361 (σ: 0.115)	0.386 (σ: 0.085)
VirusTotal	0.767 (σ: 0.041)	**0.813 (σ: 0.026)**	0.804 (σ: 0.026)
FilteredVMRay	**0.856 (σ: 0.039)**	0.790 (σ: 0.116)	0.813 (σ: 0.027)

TABLE A.3: **AUC** Results Evaluating **Ro18** on Scenario A

A.2 Country-Level Authorship Attribution with Earlier Seen APT Groups

Dataset	Imbalanced	Undersampling	Oversampling
Extracted Cuckoo	0.861 (σ: 0.018)	0.782 (σ: 0.144)	**0.895 (σ: 0.005)**
Filtered Cuckoo	0.931 (σ: 0.012)	0.923 (σ: 0.004)	**0.932 (σ: 0.009)**
VirusTotal	0.984 (σ: 0.011)	0.978 (σ: 0.002)	**0.986 (σ: 0.006)**
FilteredVMRay	0.936 (σ: 0.013)	0.927 (σ: 0.014)	**0.939 (σ: 0.011)**

TABLE A.4: **Recall** Results Evaluating **Ro18** on Scenario B

Dataset	Imbalanced	Undersampling	Oversampling
Extracted Cuckoo	**0.924 (σ: 0.016)**	0.754 (σ: 0.102)	0.855 (σ: 0.005)
Filtered Cuckoo	**0.955 (σ: 0.010)**	0.890 (σ: 0.018)	0.916 (σ: 0.028)
VirusTotal	0.986 (σ: 0.007)	0.963 (σ: 0.009)	**0.987 (σ: 0.005)**
FilteredVMRay	0.941 (σ: 0.010)	0.895 (σ: 0.011)	**0.955 (σ: 0.014)**

TABLE A.5: **Precision** Results Evaluating **Ro18** on Scenario B

Dataset	Imbalanced	Undersampling	Oversampling
Extracted Cuckoo	0.979 (σ: 0.003)	0.913 (σ: 0.110)	**0.980 (σ: 0.002)**
Filtered Cuckoo	0.992 (σ: 0.003)	0.991 (σ: 0.001)	**0.994 (σ: 0.001)**
VirusTotal	0.998 (σ: 0.002)	0.998 (σ: 0.001)	**0.999 (σ: 0.000)**
FilteredVMRay	0.992 (σ: 0.002)	0.989 (σ: 0.002)	**0.994 (σ: 0.003)**

TABLE A.6: **AUC** Results Evaluating **Ro18** on Scenario B

A.3 APT Group-Level Authorship Attribution

Dataset	Imbalanced	Undersampling	Oversampling
Extracted Cuckoo	0.753 (σ: 0.018)	0.098 (σ: 0.026)	**0.767 (σ: 0.017)**
Filtered Cuckoo	0.881 (σ: 0.021)	0.366 (σ: 0.144)	**0.911 (σ: 0.022)**
VirusTotal	0.962 (σ: 0.026)	0.188 (σ: 0.073)	**0.968 (σ: 0.008)**
FilteredVMRay	0.873 (σ: 0.023)	0.136 (σ: 0.038)	**0.876 (σ: 0.009)**

TABLE A.7: **Recall** Results Evaluating **Ro18** on Scenario C

Dataset	Imbalanced	Undersampling	Oversampling
Extracted Cuckoo	0.771 (σ: 0.015)	0.049 (σ: 0.074)	**0.774 (σ: 0.015)**
Filtered Cuckoo	**0.927 (σ: 0.030)**	0.324 (σ: 0.132)	0.904 (σ: 0.011)
VirusTotal	0.972 (σ: 0.011)	0.169 (σ: 0.095)	**0.980 (σ: 0.008)**
FilteredVMRay	0.888 (σ: 0.023)	0.063 (σ: 0.040)	**0.911 (σ: 0.014)**

TABLE A.8: **Precision** Results Evaluating **Ro18** on Scenario C

Dataset	**Imbalanced**	**Undersampling**	**Oversampling**
Extracted Cuckoo	**0.971 (σ: 0.004)**	0.588 (σ: 0.027)	0.970 (σ: 0.003)
Filtered Cuckoo	0.990 (σ: 0.004)	0.820 (σ: 0.073)	**0.992 (σ: 0.003)**
VirusTotal	0.998 (σ: 0.002)	0.803 (σ: 0.038)	**0.999 (σ: 0.001)**
FilteredVMRay	**0.988 (σ: 0.005)**	0.715 (σ: 0.032)	0.987 (σ: 0.008)

TABLE A.9: **AUC** Results Evaluating **Ro18** on Scenario C

Appendix B

Additional Results of the RFC-based Approach

The appendix consists of the recall, precision and AUC results of all experiments conducted in Chapter 7.

B.1 Country-Level Authorship Attribution with Unseen APT Groups

Dataset	Imbalanced	Undersampling	Oversampling
Extracted Cuckoo	0.349 (σ: 0.022)	0.397 (σ: 0.126)	**0.452 (σ: 0.104)**
Filtered Cuckoo	0.411 (σ: 0.152)	**0.490 (σ: 0.101)**	0.351 (σ: 0.026)
VirusTotal	0.595 (σ: 0.047)	**0.668 (σ: 0.012)**	0.642 (σ: 0.033)
FilteredVMRay	0.683 (σ: 0.031)	0.672 (σ: 0.098)	**0.713 (σ: 0.057)**

TABLE B.1: **Recall** Results Evaluating **Am17** on Scenario A

Dataset	Imbalanced	Undersampling	Oversampling
Extracted Cuckoo	0.333 (σ: 0.005)	**0.405 (σ: 0.111)**	0.405 (σ: 0.112)
Filtered Cuckoo	0.407 (σ: 0.144)	**0.494 (σ: 0.089)**	0.345 (σ: 0.014)
VirusTotal	0.661 (σ: 0.055)	**0.692 (σ: 0.019)**	0.685 (σ: 0.032)
FilteredVMRay	0.657 (σ: 0.027)	0.662 (σ: 0.091)	**0.690 (σ: 0.059)**

TABLE B.2: **Precision** Results Evaluating **Am17** on Scenario A

Dataset	Imbalanced	Undersampling	Oversampling
Extracted Cuckoo	0.350 (σ: 0.080)	0.415 (σ: 0.143)	**0.470 (σ: 0.092)**
Filtered Cuckoo	0.488 (σ: 0.056)	**0.518 (σ: 0.056)**	0.444 (σ: 0.049)
VirusTotal	0.719 (σ: 0.040)	**0.723 (σ: 0.008)**	0.715 (σ: 0.020)
FilteredVMRay	0.784 (σ: 0.076)	**0.811 (σ: 0.079)**	0.789 (σ: 0.069)

TABLE B.3: **AUC** Results Evaluating **Am17** on Scenario A

B.2 Country-Level Authorship Attribution with Earlier Seen APT Groups

Dataset	Imbalanced	Undersampling	Oversampling
Extracted Cuckoo	0.930 (σ: 0.011)	0.921 (σ: 0.017)	**0.940 (σ: 0.010)**
Filtered Cuckoo	0.934 (σ: 0.003)	0.930 (σ: 0.006)	**0.944 (σ: 0.015)**
VirusTotal	0.961 (σ: 0.014)	0.964 (σ: 0.010)	**0.981 (σ: 0.007)**
FilteredVMRay	0.937 (σ: 0.014)	0.942 (σ: 0.008)	**0.958 (σ: 0.007)**

TABLE B.4: **Recall** Results Evaluating **Am17** on Scenario B

Dataset	Imbalanced	Undersampling	Oversampling
Extracted Cuckoo	**0.961 (σ: 0.007)**	0.894 (σ: 0.005)	0.921 (σ: 0.013)
Filtered Cuckoo	**0.965 (σ: 0.002)**	0.895 (σ: 0.028)	0.916 (σ: 0.017)
VirusTotal	0.982 (σ: 0.005)	0.949 (σ: 0.015)	**0.984 (σ: 0.006)**
FilteredVMRay	**0.974 (σ: 0.003)**	0.914 (σ: 0.014)	0.972 (σ: 0.008)

TABLE B.5: **Precision** Results Evaluating **Am17** on Scenario B

Dataset	Imbalanced	Undersampling	Oversampling
Extracted Cuckoo	**0.994 (σ: 0.002)**	0.988 (σ: 0.004)	**0.994 (σ: 0.002)**
Filtered Cuckoo	**0.996 (σ: 0.000)**	0.993 (σ: 0.003)	0.996 (σ: 0.001)
VirusTotal	**0.999 (σ: 0.000)**	0.997 (σ: 0.001)	**0.999 (σ: 0.000)**
FilteredVMRay	0.997 (σ: 0.001)	0.993 (σ: 0.001)	**0.998 (σ: 0.001)**

TABLE B.6: **AUC** Results Evaluating **Am17** on Scenario B

B.3 APT Group-Level Authorship Attribution

Dataset	Imbalanced	Undersampling	Oversampling	Am17 in [2]
Extracted Cuckoo	0.850 (σ: 0.006)	0.740 (σ: 0.026)	**0.878 (σ: 0.013)**	**0.93**
Filtered Cuckoo	0.888 (σ: 0.023)	0.839 (σ: 0.025)	**0.912 (σ: 0.012)**	-
VirusTotal	0.930 (σ: 0.012)	0.895 (σ: 0.019)	**0.941 (σ: 0.010)**	-
FilteredVMRay	0.882 (σ: 0.014)	0.826 (σ: 0.011)	**0.909 (σ: 0.011)**	-

TABLE B.7: **Recall** Results Evaluating **Am17** on Scenario C

Dataset	Imbalanced	Undersampling	Oversampling	Am17 in [2]
Extracted Cuckoo	**0.896 (σ: 0.024)**	0.714 (σ: 0.013)	0.871 (σ: 0.013)	**0.933**
Filtered Cuckoo	**0.964 (σ: 0.012)**	0.805 (σ: 0.031)	0.941 (σ: 0.006)	-
VirusTotal	0.978 (σ: 0.003)	0.882 (σ: 0.020)	**0.980 (σ: 0.003)**	-
FilteredVMRay	0.950 (σ: 0.026)	0.783 (σ: 0.011)	**0.950 (σ: 0.009)**	-

TABLE B.8: **Precision** Results Evaluating **Am17** on Scenario C

Dataset	**Imbalanced**	**Undersampling**	**Oversampling**	**Am17 in [2]**
Extracted Cuckoo	0.982 (σ: 0.009)	0.943 (σ: 0.009)	**0.984 (σ: 0.005)**	**0.99**
Filtered Cuckoo	**0.990 (σ: 0.006)**	0.980 (σ: 0.004)	0.989 (σ: 0.007)	-
VirusTotal	0.996 (σ: 0.004)	0.992 (σ: 0.002)	**0.999 (σ: 0.001)**	-
FilteredVMRay	0.992 (σ: 0.004)	0.968 (σ: 0.006)	**0.995 (σ: 0.004)**	-

TABLE B.9: **AUC** Results Evaluating **Am17** on Scenario C

Appendix C

Most Important Features in a Trained RFC

The 50 most important features that are extracted from a trained RFC as described in Section 9.1.1 are listed in Tables C.1 and C.2. Note that the directly identifiable links to malware actors or families are printed bold.

Extracted Cuckoo	Filtered Cuckoo
ntallocatevirtualmemory (0.030613)	2018 (0.006179)
ntcreatefile (0.029199)	11 (0.004540)
ldrgetprocedureaddress (0.022173)	3840773 (0.004335)
messageboxtimeoutw (0.021986)	2019 (0.004265)
ldrloaddll (0.021479)	5575a517844a4ed... (0.004042)
ntclose (0.021213)	27 (0.003981)
ntprotectvirtualmemory (0.017735)	01 (0.003869)
createactctxw (0.016108)	2277376 (0.003474)
seterrormode (0.014441)	e071e63a66e8311... (0.003270)
ldrgetdllhandle (0.014361)	e19c4b4b529be2e... (0.003099)
oleinitialize (0.014261)	c41c7c5cb09416b... (0.003036)
regsetvalueexa (0.014201)	3852382 (0.002945)
ntdelayexecution (0.013925)	2121728 (0.002595)
getsystemtimeasfiletime (0.013784)	20 (0.002485)
ntterminateprocess (0.013541)	0x63bc0000 (0.002437)
cocreateinstance (0.013190)	2273280 (0.002318)
ntwritefile (0.012980)	dos (0.002176)
ntopenfile (0.012786)	address (0.001819)
setunhandledexceptionfilter (0.012555)	16 (0.001740)
ldrunloaddll (0.012507)	2117632 (0.001719)
ntfreevirtualmemory (0.011596)	mode (0.001718)
ntopenkey (0.011011)	peid_packer (0.001585)
copyfilea (0.010918)	keaddsystemservicetable (0.001575)
getfileattributesw (0.010774)	0x63dc0000 (0.001524)
regclosekey (0.010430)	cannot (0.001468)
ntqueryvaluekey (0.010384)	deletefilea (0.001342)
couninitialize (0.010342)	virtual_size (0.001257)
regqueryvalueexa (0.010230)	rdata (0.001206)
ntresumethread (0.010204)	pe32 (0.001193)
findresourcea (0.009915)	virtual_address (0.001135)
findresourceexw (0.009821)	srzu (0.001125)
regcreatekeyexa (0.009418)	onq10t4 (0.001112)
regopenkeyexa (0.008826)	_eoyhhur (0.001110)
getsystemmetrics (0.008589)	0x1000e0dc (0.001105)
ntopenmutant (0.008406)	0x1000e15c (0.001073)
ntdeviceiocontrolfile (0.008319)	0x1000e114 (0.001054)
regqueryvalueexw (0.008285)	regsetvalueexa (0.001034)
createthread (0.008266)	12 (0.001012)
coinitializeex (0.008252)	seek_set (0.000990)
ntreadfile (0.008152)	com0 (0.000957)
regopenkeyexw (0.007910)	wrf (0.000947)
ntduplicateobject (0.007876)	executable (0.000946)
loadresource (0.007807)	packer (0.000929)
setfilepointer (0.007599)	18 (0.000890)
searchpathw (0.007453)	accesscheck (0.000883)
oleconvertolestreamtoistorage (0.007334)	tss (0.000873)
getsysteminfo (0.007243)	dll_installer (0.000865)
getadaptersinfo (0.007188)	internetopena (0.000863)
regcreatekeyexw (0.007121)	0x0000c5b5 (0.000858)
getcomputernamea (0.006975)	xj (0.000858)

TABLE C.1: 50 Most Important Features Extracted from the RFC Using Different Cuckoo Reports

VirusTotal	FilteredVMRay
2018 (0.005308)	2019 (0.009035)
kernelmode (0.004474)	22 (0.006692)
2e7383f131fc3b7... (0.003908)	armadillo (0.004487)
358f4d568bd00dc... (0.003417)	created (0.004420)
365b828f383f8f3... (0.003391)	vmray_check_for_packed_pe_file (0.004403)
4096 (0.003322)	_pe (0.004023)
62314fb44e1d832... (0.003294)	pe (0.003928)
virus1 (0.003250)	packed (0.003758)
winnti (0.003050)	report (0.003754)
24299 (0.002933)	packer (0.003570)
aa1a993bb980165... (0.002675)	_packed_pe_file (0.003399)
4fc9e77ec66d439... (0.002584)	malware (0.003248)
44f9e6e606d408e... (0.002533)	with (0.003177)
181030 (0.002449)	vti_rule_score (0.002986)
22397 (0.002368)	01 (0.002947)
executable (0.002352)	rule_name (0.002833)
words (0.002314)	winword (0.002754)
pe32 (0.002309)	ip_address (0.002708)
vir (0.002268)	_static (0.002674)
win32 (0.002211)	v1 (0.002665)
19155b (0.002149)	ruleset_name (0.002658)
28875 (0.002068)	matched (0.002579)
50613 (0.002062)	is (0.002440)
18301 (0.002022)	apts (0.002427)
2359b086a133960... (0.001968)	vti_classification (0.002407)
13564 (0.001945)	71 (0.002296)
108344 (0.001833)	exe (0.002273)
28876 (0.001799)	yara_match (0.002258)
downloader (0.001750)	description (0.002172)
20181029 (0.001733)	14 (0.002128)
totaledittime (0.001710)	_yara (0.002106)
miniduke (0.001692)	**cloud** (0.002105)
15400 (0.001665)	url_artifact (0.002095)
armadillo (0.001662)	php (0.002090)
20181030 (0.001648)	**hopper** (0.002078)
petype (0.001614)	ref_file (0.002074)
probably (0.001604)	32 (0.002050)
2cef86abb489ad... (0.001595)	office16 (0.001987)
dos (0.001542)	ruleset_id (0.001901)
1040 (0.001522)	rule_type (0.001849)
134 (0.001464)	yoqz6d (0.001846)
dll (0.001445)	rule_score (0.001842)
pages (0.001413)	ruleset (0.001830)
_local_unwind2 (0.001380)	process_dump (0.001808)
20180918 (0.001377)	library (0.001790)
_adjust_fdiv (0.001366)	category_desc (0.001788)
fanni (0.001359)	policy (0.001772)
000731c51 (0.001357)	**apt10_malware_sample_gen** (0.001745)
backdoor (0.001355)	18 (0.001736)
timestamp (0.001355)	hkey_local_machine (0.001709)

TABLE C.2: 50 Most Important Features Extracted from the RFC Using VirusTotal and filtered VMRay Reports

Appendix D

Extracted Decision Tree

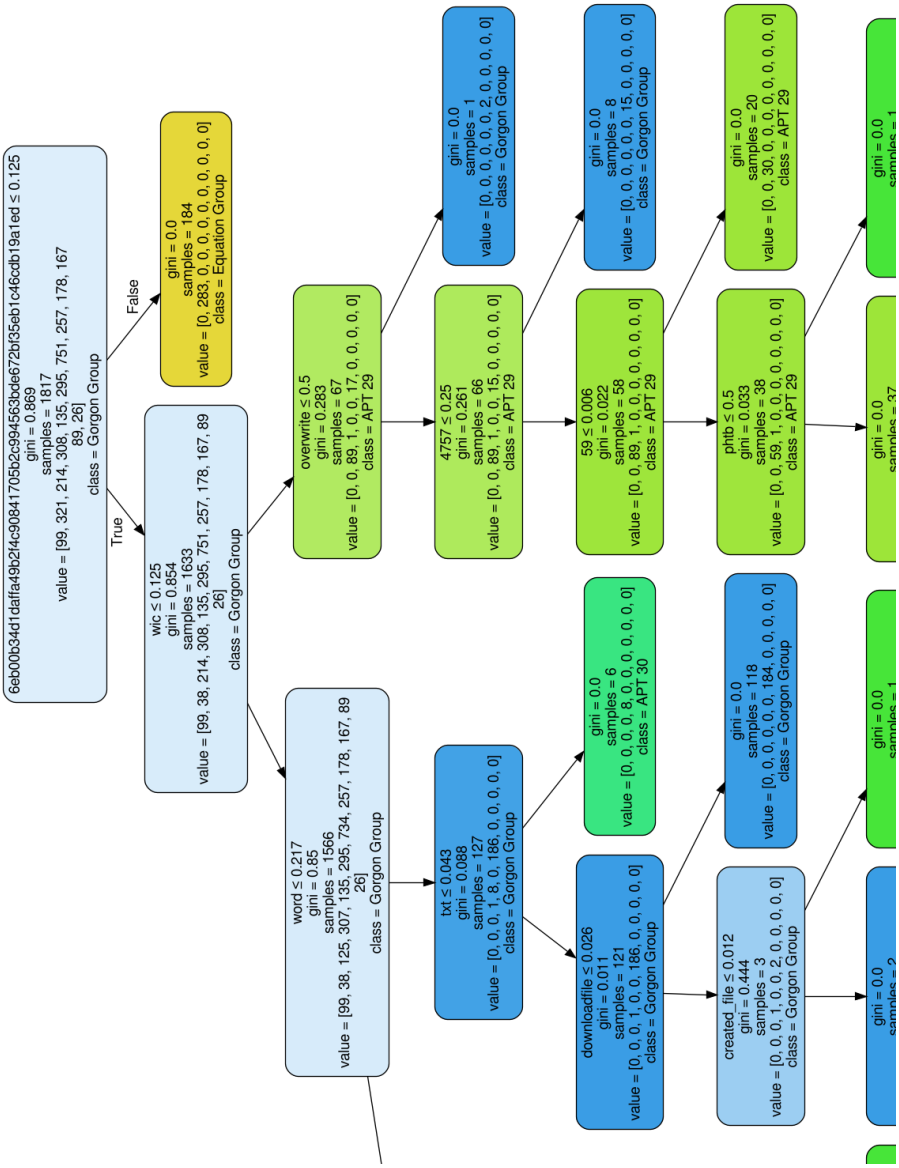

FIGURE D.1: Part of an extracted decision tree from a RFC trained on filtered VMRay reports