Weapons of Masked Destruction

Uncovering Trends in the Techniques of Global Advanced Persistent Threat Groups

A THESIS PRESENTED
BY
CATHERINE JOYCE STANTON
TO
THE DEPARTMENT OF STATISTICS
AND
THE DEPARTMENT OF GOVERNMENT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
BACHELOR OF ARTS (HONORS)
IN THE SUBJECTS OF
STATISTICS AND GOVERNMENT

HARVARD UNIVERSITY
CAMBRIDGE, MASSACHUSETTS

APRIL 2025

Thesis advisor: Jim Waldo Catherine Joyce Stanton

Weapons of Masked Destruction

Abstract

With the convenience and efficiency of a digitally dependent world come a slew of risks. In 2004 the United States Joint Chiefs of Staff acknowledged them by defining ”cyber” as the fifth domain of warfare. A fundamental challenge in the cyber domain is that offense is easy to conceal and attacks are difficult to attribute. Political scientists are still trying to understand how the cyber domain will impact international security and competition. This thesis seeks to contribute to the field by revealing trends in the types of malware used by global cyber threat actors and trends in the malware used over time. I analyze an open source dataset from the IEEE Dataport containing malware binaries conveyed as images and labeled by the actor that deployed them, or by the year they were used. I train four image classification and two clustering models on the Malimg Dataset to assign each sample a malware family. I identify threat actors that use the same malware families with similar frequencies and find parallels between the cyber domain and kinetic domains of warfare. States and malware families exist in a many-to-many relationship: one type of malware may be used by multiple states, and each state uses multiple types of malware for different missions or against different targets.

Contents

Listing of figures

This thesis is dedicated to two incredible women in stem.
Catherine (Cathy) Colgan Stanton and Joyce Yozzo Roy.

Acknowledgments

My gratitude for everyone who made this thesis possible is greater in magnitude and runs in more directions than any vector mentioned herein. Thank you to my advisor, Professor Jim Waldo. I left each of our meetings more excited about this project and full of questions and hypotheses for future ones. Thank you for prioritizing me as a whole student and whole person, and for making this my most intellectually rewarding experience yet.

Thank you to Professors Bruce Schneier and Alex Young for providing domain-specific guidance in cybersecurity and statistics. Thank you to the Technology Group at the Federal Reserve Bank of New York, and particularly ISNY, for giving me hands-on experience addressing emerging questions in cybersecurity, and for motivating my choice to pursue a thesis on it.

Thank you to my teachers in the White Plains City School District who prepared me well for college and lifelong learning. A special thanks to Ms. Pahk, Ms. Danz, and Mr. Spiconardi for inspiring my love of math, science, and social science. This thesis is the product of all three.

To the Hasty Pudding Theatricals and everyone else with whom I've had the privilege to make music and art at Harvard, thank you for making me feel at home. To Ruth, Sam, Kiesse, Lexi, Nour, and

Nico, thank you for being the best Proctor and co-PAFs to work with in making first-year students feel at home. To my own PAF Charles, thank you for helping me find my footing on campus and teaching me what it means to remain value-driven.

Sarah, Sedina, Selorna, and Milen: thank you for being my rocks and for knowing me better than I know myself. Counterintuitively, this work would not have been possible without doorknob confessions, cathartic eights, or impromptu cabinet battles. Thank you to everyone I see regularly in Winthrop House, and especially to Matt DosSantos DiSorbo for providing the most pragmatic advice in statistics and life. Thank you to the family I see for dinner in Eliot every Sunday. You are the most lively, creative, and caring community I could ever imagine.

Thank you to the brigade of Stanton, Roy, Fairbank, DelBalzo, DePaola, Brown, Yozzo, Soler, and Morgner cousins, aunts, uncles, and grandparents for your endless support. Thank you to Melanie, Camila, Viktoriya, Julianna, and Maya for being my first calls when I'm home.

Finally, thank you to Beth and Michael Stanton for encouraging me to explore and for standing by every time I dive into a niche interest or hobby. Thank you for celebrating every triumph with me, and for being a phone call away during every failure. Thank you to Patrick Stanton. There are few things I enjoy more than our conversations. Thank you for bringing curiosity, thoughtfulness, and more love than any older sister has the right to expect. And of course, thank you for inspiring me to run more than just code during this process.

In Intelligence, as in other callings, estimating is what you do when you do not know.

Sherman Kent

1 Introduction

1.1 Context

With the convenience and efficiency of a digitally dependent world come a slew of risks. When individuals store sensitive data (such as bank account numbers, health records, or passwords) in an online record keeper, they trust that their information will remain confidential, unaltered, and readily available. When they automate processes (such as driving, purifying water, or sending timely communications [4, 15]), they trust that the machines controlling those processes will function accurately and continuously. Any compromise of the systems could have consequences for human privacy and safety. Recognizing that these consequences pose a threat to national security, the United States Joint Chiefs of Staff defined

"cyber" as the fifth domain of warfare in 2004 [48]. A fundamental challenge in cyber warfare is that offense is easy to conceal and attacks are difficult to attribute. It is nearly impossible to hide mobilization for a ground attack, nuclear stockpiles, or mass weapons systems given the reach of advanced satellite imagery and the extent to which nations share knowledge through international institutions. For example, the United States' Defense Intelligence Agency spotted Russian troop movements on the border of Ukraine in September 2021 before Putin's invasion in February 2022. They did not detect anomalies in Russian cyber activity until January of 2022, and then again on the day of the ground invasion [43], even though cyber weapons proved instrumental to Russia's attack on Ukraine and the devastation that has ensued [23, 28].

Participation in the cyber domain requires only a laptop, so it is more accessible to a wider array of actors than any kinetic domain of warfare. In the wake of a cyber attack, the perpetrator is often unknown. It is the role of the intelligence community to use any and all data about code leveraged in the attack, the timing of the attack, or the systems targeted to determine liability. Threat actors in the cyber domain can be categorized as cybercriminals or Advanced Persistent Threats (APTs). Cybercriminals usually deploy malicious code, or "malware," on a large set of non-specific targets.[1] They are not typically sanctioned by a state, though their national origin may be known or discovered. Advanced Persistent Threats are prolonged attacks in which a threat actor uses multiple vectors (e.g., malware, social engineering) to gain access to and exploit a critical system [2, 3, 38, 41, 42, 49]. APTs tend to be stealthy, well-organized, and

[1]Cybercriminals largely exist to turn a profit and do not discriminate in choosing targets. They often cast a wide net, rather than focusing on a particularly lucrative victim, such as a nation-state. They benefit from stealing and selling sensitive data, or from demanding large ransoms for files. In 2023, it is estimated that cybercriminals made over $1 billion through the use of ransomware alone [13].

use state-of-the-art techniques. They often receive the explicit support of a state and work to advance that state's national interests.

1.2 Goals

Political scientists are still trying to understand how the cyber domain will impact international security as a whole [16, 35, 40]. This thesis seeks to contribute to the field by uncovering trends in the types of malware used by global cyber threat actors and broader trends in the malware used over time. Some of the questions that help guide the analysis are as follows. Do cyber threat actors from allied nations tend to use similar/the same types of malware? Are any types of malware unique to actors from the same state? And is it possible to differentiate threat actors from each other based solely on the malware they use?

To answer these, I analyze an open source dataset containing individual pieces of malware labeled by the threat actor that deployed them, or by the year they were used. The malware in the dataset are conveyed as grayscale images in which each pixel is assigned a value between 0 and 255 based on the value of its corresponding byte in the malware binary.[2] In order to learn about the behavior of each malware sample, I determine the family of malware to which it belongs. Malware families are classes of malicious code with similar semantics, capable of similar attacks.[3] I train four image classification models[4] (a kNN classifier, a random forest model, a support vector machine, and an XGBoost classifier) as well as two clustering models

[2] A binary is an executable file of machine code in which all commands are encoded as 0's and 1's.

[3] One malware family may contain a specific set of "worms," which infect multiple devices on the same network, without human intervention [24]. A separate malware family could contain Trojans, which disguise as benign code to enter a system and then inject other forms of malware [1].

[4] The choice of models were informed by [8], [33], and [31].

(KMeans and agglomerative clustering) to determine malware families based on a training dataset (the Malimg dataset [32]). The results can help states, industries, and organizations that know they are targeted by particular cyber threat actors[5] decide how to allocate limited resources in order to best defend against the technical capabilities of these actors.

1.3 Roadmap

The thesis proceeds as follows. Chapter 2 reviews the current state of malware analysis and APT attribution, as well as the models employed in this work. Chapter 3 introduces the structure of the datasets used and discusses their limitations. Chapter 4 justifies the models and parameters applied to the data. Chapter 5 evaluates each model and interprets the results in aggregate. Chapter 6 concludes with a discussion of the implications and limitations of this work, as well as avenues for future research.

[5]States may determine their threats in the cyber domain based on knowing their adversaries and allies in general. Industries and organizations can look to annual reports released by CrowdStrike and Mandiant, as well as profiles of cyber threat actors such as the MITRE ATT&CK Matrix [29] and Rapid7 [39], which include information about actors' interests and previous targets.

2

Literature Review

This chapter is divided into four sections. The first motivates malware analysis and discusses the pros and cons of existing techniques used for its end. The second presents a deeper dive into image recognition for malware analysis and reviews the models employed in this thesis. The third presents a history of APT attribution and describes how malware analysis is used towards its end. The chapter concludes with a discussion of gaps in the literature.

2.1 Existing Techniques for Malware Analysis

Malware refers to a piece of malicious software that exploits the functionality of an operating system or application, allowing those who administered it to access protected accounts, steal/alter data, or

damage individual devices [5]. Malware is categorized into families based on the way in which it goes about damaging a system or application. Malware from multiple malware families may be used in a single cyber attack. For example, malware from one family may be used to enter a system in disguise (this type of malware is known as a Trojan) and malware from another may be used to withhold files on that system from their owners until a sum of money is paid (this type of malware is called ransomware). The semantics used in writing different types of malware vary greatly, meaning that the binary code files of different types of malware vary greatly. It's also possible to write the same type of malware using different semantics (i.e., two Trojans may be implemented using different commands). But when two samples of malware use similar semantics to achieve similar ends, they can be categorized into a single family [44].

Traditional methods of analyzing malware samples for their behavior and/or family can be categorized as static or dynamic. Static analysis is a signature-based process to discover a malware binary's behavior without running it. Static analysis techniques include reading disassembled code line-by-line, examining on-disk resources used, and reading printed output strings [22, 45]. Static analysis is thorough and not computationally-intensive, but it is human-intensive and slow to perform by hand. Its accuracy also suffers when presented with encrypted/obfuscated binaries [38].

Dynamic analysis involves running a malware binary in a protected sandbox environment to characterize its behavior. It makes the analysis of an individual binary more efficient, but is inefficient to perform at scale because of the resources required to administer several protected sandbox environments. Dynamic analysis usually isn't hampered by obfuscation techniques, but there are a few threats (known as Advanced Volatile Threats, or AVTs) that can detect when they are running in a sandbox environment and mask their behavior

accordingly [22].

In 2011 malware analysts began using image processing and computer vision to make malware analysis more portable. Given the rate at which new malware is developed and deployed.[1], the ability to analyze malware samples en masse, as opposed to sample-by-sample, is important. In order to convey malware samples as images, Nataraj and colleagues [32] reshaped malware binaries into matrices of bytes and converted each byte to a grayscale pixel shaded on a scale of 0 to 255 based on its value.

There are similar patterns between the pixels of malware images from samples in the same family, and these images have similar textures at large. Most of the images, from all families, feature swaths zero-valued pixels.[2] These are the result of an obfuscation technique called zero-padding [22] which tries to mask the behavior of a piece of malware by including null information. Machine learning models learn to ignore these sections because they tend to be small [14] and do not explain a significant amount of variance between images in the dataset. Nor do they change the global structure of the malware binaries [14]. Using malware images, Nataraj and colleagues [32] re-classified malware samples with known families with 98% accuracy: they compiled the images with their family labels into the Malimg dataset[3] which contains 9,348 malware images.

[1] In 2020, Kaspersky labs identified 360,000 new malware files deployed each day, which marked a 5.2% increase in one year [20] Trojans accounted for the majority of this malware, followed by Backdoors. Trojans enter a system undetected and can then drop other forms of malware, steal data/passwords, or track user-activity on the systems they've infiltrated. Backdoors are a type of Trojan that allow attackers to remotely take control of an infected machine [].

[2] As is evident in 3 which includes examples of malware images.

[3] Used as the training dataset of this work.

2.2 Image Recognition for Malware Analysis

Even before the creation of malware images and the Malimg dataset, researchers applied machine learning to classify malware binaries into families. The features used in these models were drawn from the results of static and/or dynamic analysis. In 2014, Yuxin et al. [51] represented malware binaries as n-grams of their operational codes (opcodes), which are instructions that describe the behavior of a software file. Then they used a deep belief network (DBN) to identify patterns between them.[4] The set of n-grams from a malware binary's opcode and the set of features extracted from a malware image are similarly large. Yuxin and colleagues used document frequency and information gain to decide which features to keep in the data. while document frequency is not applicable to image analysis, information gain is a similar criterion as was used to reduce the dimensionality of malware image features. Deep belief networks are useful because of their ability to learn the features of unlabeled data.

Representing malware binaries as images has increased opportunities to apply machine learning to malware analysis, and widened the scope of researchers who can contribute. Since behavioral characteristics can be determined by pattern-matching malware images, cybersecurity-specific domain knowledge is no longer required to recognize which commands make a piece of software dangerous [10]. Several studies have used state of the art machine learning models for malware analysis. In 2018, Gibert et al. [14] used the first convolutional neural network (CNN) to classify malware images from the Malimg and Microsoft Malware Classification Challenge datasets, respectively. They achieved 98.46% accuracy on the Malimg dataset[5]

[4]Though the Malimg dataset was available at this point, I choose to include this example because the models it employed are similar to those used here.

[5]When using input images of size 256 pixels by 256 pixels and 10-fold cross-validation.

and 97.50% accuracy on the Microsoft Malware Classification Challenge Dataset.[6]

Subsequent analyses have turned to transfer learning instead of defining original CNNs to classify malware images. Transfer learning can create deep neural networks that effectively classify malware images by re-training the final layers of more generalized image recognition models (i.e., models that recognize animals, foods, landscapes, everyday items, etc.) to find patterns in malware binaries specifically. In 2020 Mitsuhashi and Shinagawa [30] trained a version of VGG19 and re-classified the Malimg dataset with 99.72% accuracy. They achieved the highest accuracy to date by freezing 80% of the convolutional layers in VGG19 (which was trained on ImageNet[7]) and re-training the final 20%. Importantly, they discarded the fully-connected layers of VGG19 as these are intended to assign input samples one of the 1,000 classes in ImageNet. The Malimg dataset only has 25 classes, all of which are specific to malware. Mitsuhashi and Shinagawa also addressed the fact that the Malimg dataset is unbalanced, meaning that there are different numbers of samples belonging to each malware family. To avoid training a model that simply memorized the features of the most prevalent class(es) and predicted it too often, they downsampled from the Malimg dataset. They compiled their training dataset from a subset of the images in the Malimg dataset (since the smallest malware family in the dataset had 80 samples attributed to it, they compiled 80 sampes from each family in the Malimg dataset and used it for training).

In 2021, Edie and Jasim [10] used transfer learning on the Xception model to classify the Malimg dataset. They also removed the

[6]Also using 10-fold cross-validation, but input images of size 128 pixels by 128 pixels.

[7]ImageNet is an open-source collection of 1.48 million images with 1,000 class labels, intended for training general machine learning models. Its classes range from the animals to foods to lanscapes and everyday items, mentioned above.

fully-connected layers from the model and experimented with both a kNN classifier and a support vector machine (SVM) to perform the final classification of images, after features were extracted from the convolutional layers. With the SVM, they achieves an accuracy of 99.20% on the Malimg dataset.

In 2023, Pachhala and colleagues [36] augmented the VGG16 model to create a specialized deep neural network for predicting malware families. VGG16 was created by the same group as VGG19 (the Visual Geometry Group of Oxford University). VGG16 uses 16 convolutional layers as opposed to 19 and was found to have a better performance on the original task of classifying ImageNet [46]. Pachhala et al. tested their model on the Malimg dataset and achieved over 98% accuracy after training the model for 160 epochs. They also explain the importance of data pre-processing and augmentation well. To remove noise from the malware images, and thus make binaries part of the same family even more recognizable as such, they resized images to 32, 64, 128, and 256 pixels squared, respectively, before inputting them to the VGG16 model.[8] Instead of downsampling from the Malimg dataset, as Mitsuhashi [30] did, Pachhala dealt with the class imbalance by generating synthetic samples of malware binaries in the smallest classes of the Malimg dataset and adding them to their training dataset. They generated these images through "rotation, shearing, zooming, flipping them horizontally or vertically, and adjusting the degrees of brightness."

Instead of training a Deep Neural Network to classify malware images, this thesis compares *classification and clustering* techniques. The choice of models is informed by the work of Narayanan and colleagues in 2016 [31], Deepa and colleagues in 2022 [8], and Divakarla et al., 2024 [9]. Narayanan used a kNN classifier, support

[8]VGG16 notably requires inputs of size 224 by 224 pixels, so images must have been resized again before being presented to the model.

vector machine, and Artificial Neural Network (ANN) to classify the Microsoft Malware Classification Challenge Dataset. They converted samples in the dataset to images manually, using the same approach that Nataraj used to create the Malimg dataset [32]. They did not use convolution for feature extraction of their images, but they did reduce the dimensionality of features using Principal Component Analysis. They found that a fewer number of components led to models that were less overfit in this context (using 12 principal components was found to be preferable to 52 principal components). Their results showed the kNN classifier to perform best [31].

In 2022, Deepa et al. used a Convolutional Neural Network (CNN) to extract features of the Malimg dataset and then applied a random forest classifier, SVM, XGBoost model, and Deep Neural Network (DNN) to the features to classify samples. The accuracies of each are provided in table 2.2.1. The CNN used for feature extraction was the VGG16 model. Deepa's work underscores the importance of using convolution for the feature extraction in image classification tasks. Convolutional layers apply a small filter to each pixel in an image, and the pixels surrounding it. The filter multiplies all of these pixels by a constant and adds them together for a numerical summary of the variation in pixels in each of the image's regions. Most CNNs also incorporate pooling, which aggregates the results of the small filters over slightly larger regions in order to gauge global features of images. In earlier studies of malware images [31, 32], a GIST algorithm for feature extraction was used instead of convolution. In a GIST algorithm, an image is broken up into blocks. A Gabor filter is passed over each to summarize the content of the block in a single pixel. Then, the pixels are averaged by region (of a fixed size, determined by the researchers) to obtain the feature(s) of the initial image [50]. The problem with GIST feature extraction relative to convolution is that it aggregates features/pixels early rather than applying a constant

Image Classification Algorithm	Percent Accuracy
SVM	98.51%
Random Forest	97.60%
XGBoost	99.39%
DNN	98.51%

Table 2.2.1: Classification model accuracies on the Malimg dataset presented by Deepa et al. [8].

multiplier to each region of the original image first. It reflects global features of images, but in malware analysis, it can miss attackers' attempts at obfuscating code through small-scale variations.

In 2024, Divakarla et al. [9] presented an XGBoost model for classifying malware images trained on the Malimg and Malevis datasets. Since the Malevis dataset contains some benign samples, their XGBoost model is able to distinguish malicious code from harmless code. Divakarla's work does not use convolution to extract features from malware images and it achieves a 97.17% accuracy, both in assigning malware families to malicious samples and in identifying benign code as such.

In this thesis, I use a kNN classifier, support vector machine, random forest classifier, and XGBoost model. The following subsections provide a general overview of each.

2.2.1 kNN Classification

Developed by Fix and Hodges in 1951, a k-Nearest Neighbor (kNN) classifier assigns classes to data points based on the true class of the majority of some k points in its vicinity [12]. The k parameter can influence model performance so it is often tuned using cross-validation. The distance metric to determine which points are truly near one another is also flexible, but Euclidean distance is

common and used in this thesis. kNN Classification has been used for previous instances of malware analysis. In 2019 Noor et al. [33] used it to classify APT reports, which contain information about IP addresses, commands, malware families, and file hashes used by an APT. The researchers one-hot encoded these features before classification. This practice actually detrimental to a kNN classifier because it means that two points with completely different features and thus classes may be equidistant from a third point, confusing the model as to which class the third point belongs. It could mean that two very similar points are equidistant from a completely different point. So it is not surprising that kNN classification only yielded an accuracy of 39.69% in Noor's work. It is reasonable to assume that the kNN classifier would work better on a dataset of images, with features extracted from VGG16 since each feature takes on more values than just 0 or 1, and that is consistent with the result of Narayanan and colleagues in 2016 [31].

2.2.2 Support Vector Machine

The support vector machine was introduced in 1995 [6]. An SVM non-linearly transforms input vectors into a space with more features. The purpose is to represent data in such a way that there will exist a linear boundary between vectors of different classes. The SVM finds this boundary by determining the "optimal hyperplane" between vectors in the high-dimensional feature space. That is, it finds the plane at which the distances between vectors in different classes are *maximized*. The closest vectors to the hyperplane are known as the support vectors, and they are sufficient for finding the optimal hyperplane. This means that the SVM can be theoretically trained at a high accuracy using a relatively small sample of the entire training data set.

2.2.3 Random Forest

Random Forest is a versatile ensemble learning technique. It involves training some number (B) of decision trees on B bootstraps of the original data. Each level of each tree considers a random subset of features from the data and selects the optimal feature on which to split the data (i.e., the feature that will result in classes where the most points belong to the same *true* class). The set features considered for splitting at each level of the tree can be distinct from those considered at higher levels. The final classification is determined by calculating the probability that each observation belongs to a given class, based on the classes it was assigned by each tree in the random forest. Increasing the number of trees in the forest can, up to a certain threshold, improve the model's accuracy without overfitting since the model is built on randomness (bootstrapping the data incorporates randomness and the features considered for splitting at each level of the tree are selected randomly).

A strength of random forest models in general is their interpretability; determining which features are most effective at splitting data is intuitive. A key metric for determining these features is the Mean Decrease in Impurity (MDI), which is defined for each node of the tree as such:

$$\Delta I_n = (\frac{n}{N})[Gini_n - \sum_{m \in Child(n)} (\frac{m}{n}) Gini_m] \tag{2.1}$$

$Gini_m$ refers to the Gini Index for each assigned class and is a measure of purity. The Gini Index is the product of the proportion of points in class k in a region times the proportion of points that are *not* truly in class k but are in that region, summed over all k classes in the dataset. If data is perfectly split, and no points are classified incorrectly, the Gini index would be 0. So for each node in each tree

of the random forest, the MDI is the difference between the impurity at some node and the sum of the impurities of all nodes thereafter, weighted by the proportion of observations that were classified by that node ($\frac{n}{N}$). It measures the extent to which the feature split on at that node contributed to homogeneous classes in later levels of the tree, or classes containing observations from a variety of classes. Even if the node resulted in one pure class and one very mixed class, the MDI will reflect the impurity of the mixed class by summing all of the Gini indices for classes that appear below the given node. Nodes with higher MDI values correspond to features that *best* discriminate between the different classes of data.

2.2.4 XGBoost

XGBoost is an open source library that performs gradient boosting. Like random forest models, gradient boosting is built on decision trees. It takes several trees that split data on weak, non-descriptive criteria, and combines them into a stronger, more expressive decision tree. An initial tree splits data on a trend it sees in the training set. The residuals of this split are calculated for each observation. Then a subsequent tree is added to split data based on the values of those residuals. This tree is usually assigned a smaller weight than the initial, which makes sense since it is not directly based on the data itself. The process is repeated until a stopping condition is met. XGBoost gives higher weights to trees that are more closely informed by the data and lower weights to those that split on the basis of a prior tree's errors. As each new decision tree is added, class predictions serve to eliminate extremely high residual values for any data points. XGBoost reduces the risk of overfitting by prioritizing information closer to the actual data over information based on the errors that one tree made in categorizing that data. It is also friendly

to interpretation because it assigns different weights to the different features it splits on, allowing researchers to gauge how important each feature is in classifying the data.

2.3 APT Attribution

APT Attribution is the task of identifying the perpetrator of a cyber attacks and learning as much about their operations and tactics as possible [27, 41, 49]. It allows victims to offensively hamper APT operations, fortify critical systems against them in the future, and contribute to the public domain of knowledge about an APT. APT attribution is holistic, relying on technical knowledge about malware binaries and political knowledge about a threat actor's motives. In 2015, Rid and Buchanan described a three-layered approach to APT attribution, summarized in table 5.4.5[41].

Each layer of Rid and Buchanan's model depends on the next. Attempting to attribute an attack to an APT solely on the basis of the malware used (result of the technical layer) is impractical because multiple APT groups may reuse the same types of malware. Likewise, attributing an attack to an APT solely based on knowledge about political affairs risks assigning liability in the absence of technical artifacts that prove an intentional attack occurred. With each layer of the model, the *aperture*, or "scope of sources that can be brought to bear on a specific investigation" [41] widens. There is more context that can differentiate suspects from each other. In the wake of a cyber attack, two APT groups may be equally plausible perpetrators based on technical evidence (i.e., they might both use the type of malware deployed, and at similar rates [49]). However, widening the aperture and considering which of the groups is sponsored by a nation adversarial to the victim (as is done at the strategic level) could yield

Layer	Description
Technical/Tactical	Reverse engineering malware; examining specific raw data/commands to understand the damage of a certain cyber attack. Sometimes reveals the type of system attacked, but rarely reveals the perpetrator or motivation for the attack.
Operational	Analyzing the commands in an attack for large-scale behavior. Capable of differentiating types of malware or recognizing similarities between malware used in separate attacks. Capable of revealing the motive against a particular *system*, but rarely reveals a political motive.
Strategic	Evaluating the state of international affairs and geopolitics to understand who is responsible for a cyber attack and why they chose a certain attack type or target. Often uses metadata about an attack, such as the time of day it was active vs. latent, language of printed strings, etc.

Table 2.3.1: Three-layer APT attribution model presented by Rid and Buchanan in 2015 [41].

a better estimate as to which APT group is responsible.

APT attribution is typically conducted by state-sponsored intelligence agencies (e.g., National Security Agency—NSA, or Government Communications Headquarters—GCHQ) because they have the widest apertures. They employ specialized technical, strategic and linguistic experts, they possess mass amounts of information at classified levels, and they have the computing power to generate big, accurate datasets about cyber attack mechanisms using dynamic analysis. Still private companies dedicated to APT attribution and exposure have proven incredibly important. One example is Kaspersky Labs whose 2014 report on the APT group *Careto* caused the group to halt operations within one hour [18, 41].[9] Mandiant is another example. When it exposed APT1 (associated with China's People's Liberation Army (PLA)) in 2013, the group halted all activity for 40 days and resumed an abnormally low level of activity for the following 160 days [27, 41].

In terms of Rid and Buchanan's model, this thesis focuses on the operational level. The analysis used to create the dataset(s) of malware images would be considered part of the technical layer. By discussing the malware usage patterns between APT groups, I touch on the strategic layer but remain cognizant of Rid and Buchanan's point that "strategic analysis is non-technical by definition," [41] and that there are limitations in trying to make technical findings overtly political. The purpose of this thesis is to illuminate patterns of malware use within and between APT groups. Practically, this could help inform the defensive efforts of an entity that knows it is targeted by a specific threat actor, or state that knows it is targeted by another state, but does not know the attack techniques typically (or recently) used by that group or state.

[9]Kaspersky Labs also found Careto activity to resurface in 2024, but the ten years of inactivity underscore the efficacy of their initial exposure [21].

In 2023, Xiao et al. [49] proposed an automated multi-modal neural network for APT attribution (APT-MMF). By using three distinct types of features about a piece of malware,[10] they achieved a micro-F1 score of 83.21%. Xiao did not utilize image representations of malware, instead relying on features extracted from static and/or dynamic analysis. Though accurate, the main drawback of their approach is that it undermines the rate at which APT groups pivot in response to shifting political contexts and motives. The inputs and training data for their model come from APT profiles, which evolve frequently. So the model would consistently need to be re-trained, which can be expensive. APT attribution is an ongoing effort; it is common to continuously discover new characteristics of an even after first analysis, and researchers that perform it consider this in their methods [27].

For a success story in APT attribution, one can look to the identification of APT29, or CozyBear, from Russia's Foreign Intelligence Service (SVR) as the actor behind the 2020 SolarWinds attack. During this attack, Russian hackers injected malware into a software update of SolarWinds' Orion platform, which is used by nearly 12 high-profile U.S. governmental agencies and about 100 Fortune 500 companies including Microsoft, Intel and Cisco [47]. The malware granted hackers administrator-level access to the systems on which it was deployed, meaning that hackers could steal and withhold data about a vast number of American citizens, as well as change the settings of computers it infected, potentially having effects on downstream systems. Several cybersecurity companies pooled data about previous attacks to recognize the type of malware used on the

[10] 1) One-hot encoded "types" of malware, such as family, category (e.g., ransomware, spyware), file type, programming language, and data types used. 2) Natural language features of the malware extracted using a BERT model. 3) Node2vec encoded topological relationship features describing the relations between pieces of malware.

Orion platform (Sunburst), identify the obfuscation techniques applied to it, and analyze the latent time between steps required to gain access to the Orion platform. They also drew comparisons between the types of companies targeted by the Sunburst malware and determined the attack to be waged against a state-sponsored group. These are all consistent with behavior of APT29, so the cybersecurity companies were able to confidently attribute the attack to APT29 [38]. Insights from this thesis could be brought to bear on future APT attribution cases by contributing to the body of knowledge about the types of malware typically used by prominent APT groups.

2.4 Gaps in the Literature

The existing literature in malware image analysis revolves around achieving higher accuracies when reclassifying samples for which malware families are known, such as those in the Malimg dataset. Most works employ transfer learning to train malware-specific Deep Neural Networks for this task. This thesis presents a departure from that: in addition to defining a classification scheme for the Malimg dataset, I apply it to a new set of malware images for which the malware families are unknown but the threat actors that used them *are* known.[11] The goal is to shed light on the types of malware most frequently used by a variety of global cyber threat actors. Hopefully it will also help entities who know that they are targeted by specific actors choose the technical controls put in place to deter those actors.

[11] A tangible contribution of this thesis is a labeling scheme for this dataset. My labeling scheme may be expanded upon or adopted by the curators of this dataset, widening the scope of models and research projects for which it may be used in the future.

3

The Data

The analysis presented here revolves around a dataset uploaded to the Institute of Electrical and Electronics Engineers (IEEE) Dataport in May 2024, titled *Advanced Persistent Threat (APT) Classified Dataset*.[1] The IEEE Dataport is an open source for downloading data that have been used for research in several domains, including cybersecurity. The APT dataset contains two sets of malware images: the first set are labeled by the threat actor that deployed them and the second are labeled by the year they were deployed. Within each set, malware images are further sorted into "groups" based on Ahash subclassification. The Ahash algorithm is given a key and returns a hash. In this case, the key would be a malware image and the hash would be a subgroup. The groups returned from Ahash are not

[1]Herein referred to as the APT dataset.

unique: several samples with different keys wind up in the same group. They are purely intended for organization and do not convey further information about the malware samples contained within them. I chose to disregard these groups since I was interested in studying malware at the granularity of APT groups and years.

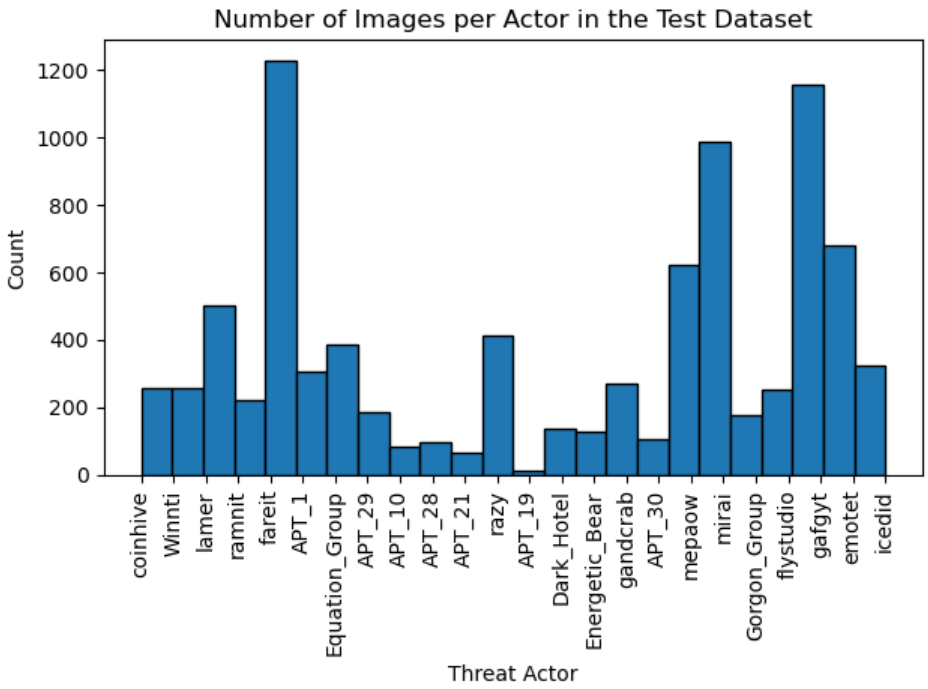

Figure 3.0.1: Number of Malware Images Included in the APT Dataset, by APT Group and Malware Strain.

The first subset of the APT dataset contains 8,861 malware images. They are sorted into 24 classes based on the threat actor that deployed them. Figure 3.0.1 summarizes the number of images belonging to each class. Twelve of the threat actors included in the dataset are prominent APT groups and the other twelve are malware strains. Table 3.0.1 lists each APT group and its country of origin.

APT Name	Number of Images	Country of Origin
APT_1	308	China (PLA)
APT_10	82	China (PRC)
APT_19	12	China (PRC)
APT_21	67	China (PRC)
APT_28	95	Russia (GRU)
APT_29	184	Russia (SVR)
APT_30	106	China (PRC)
Dark_Hotel	139	South Korea (ROK)
Energetic_Bear	127	Russia (Russian Federation)
Equation_Group	387	United States (USA)
Gorgon_Group	176	Pakistan
Winnti	259	China (PRC)

Table 3.0.1: APT groups in the Test Dataset and their Countries of Origin [29].

Malware strains refer to campaigns of malicious code used against a system. Though they may be used as part of an APT's attack arsenal, or common to several APT groups, malware strains tend to be short-lived and not explicitly sponsored by a state. Like APT groups, malware strains can leverage code from a variety of different malware families,[2] which is likely why they are included alongside APTs in the dataset and why they can analyzed using the same methods.

The second subset of the APT dataset contains 17,441 images labeled by the year in which they were deployed (2011 through 2023). This subset is disjoint from the first. There is no indication as to which actors deployed the binaries labeled by year, and no indication as to the year(s) that the binaries labeled by APT group were used (though it might be reasonable to assume that they were also

[2]Emotet is an example of a malware strain included in this dataset. Emotet is a Trojan, meaning that it disguises as benign to gain entry to a system. Once in, Emotet has been known to drop malware from other families, such as Azorult, TrickBot, IcedID, Qbot, Ryuk, and Bitpaymer. These forms of malware can collect credentials, hold files ransom, record keyboard strokes, and more [11].

collected between 2011 and 2023). Figure 3.0.2 summarizes the number of malware images included from each year.

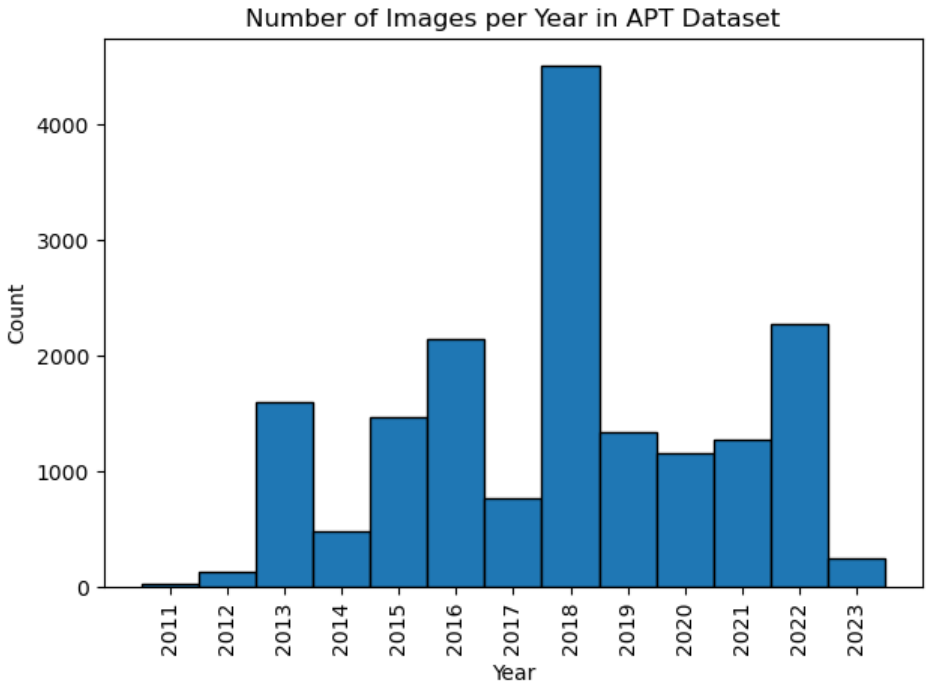

Figure 3.0.2: Number of Malware Images Included in the Dataset by Year.

In the first subset of data, there are more samples from malware strains than there are from APT groups. The malware strains with the most samples are fareit, gafgyt, and mirai. The APT groups with the most samples are the Equation Group, APT1, and Winnti. In the second subset of the data, most samples come from 2018. In both cases, there is a class imbalance. If I were training a model on this data, the class imbalance would pose a challenge (addressed in the next subsection which discusses Training Data), but since the APT dataset is used only in prediction, the fact that some classes contain more samples than others does not introduce bias to the models. In

fact, it may present an opportunity to learn about the relative activity levels of different threat actors or of peaks in malware use over time. If the sample is representative (i.e., if malware is included in the data with the same frequency that it is actually deployed), then the fact that there are more samples from one actor would indicate that that actor is more active than an actor from which there are fewer samples in the dataset. It can be difficult to determine if a sample is representative in the cyber domain, however, given that stealth is a goal. The best cyber attacks are those that evade detection, allowing actors to achieve their goals without reprehension. This creates a censored data problem; it cannot be determined if samples are excluded from the dataset, or what characteristics may lead samples to be excluded. It also impedes the ability to determine if a dataset is representative of the malware landscape by making it difficult to quantify the total number of malware samples or families in existence.

The APT dataset makes it possible to identify similarities in the malware used by different cyber threat actors and the malware used over time. But it lacks information about the nature of that malware. Analyzing the APT dataset as-is could reveal that APT1 and APT30, for instance, use similar malware binaries. But it could not determine whether those binaries served as ransomware, trojans, backdoors, worms, or something else entirely. My analysis seeks to fill in this gap by assigning a malware family to each sample in the APT dataset. My approach is to train a series of image classifiers on a dataset in which malware families are known (the Malimg dataset, described in the next subsection) and apply them to predict the malware families present in the APT dataset.

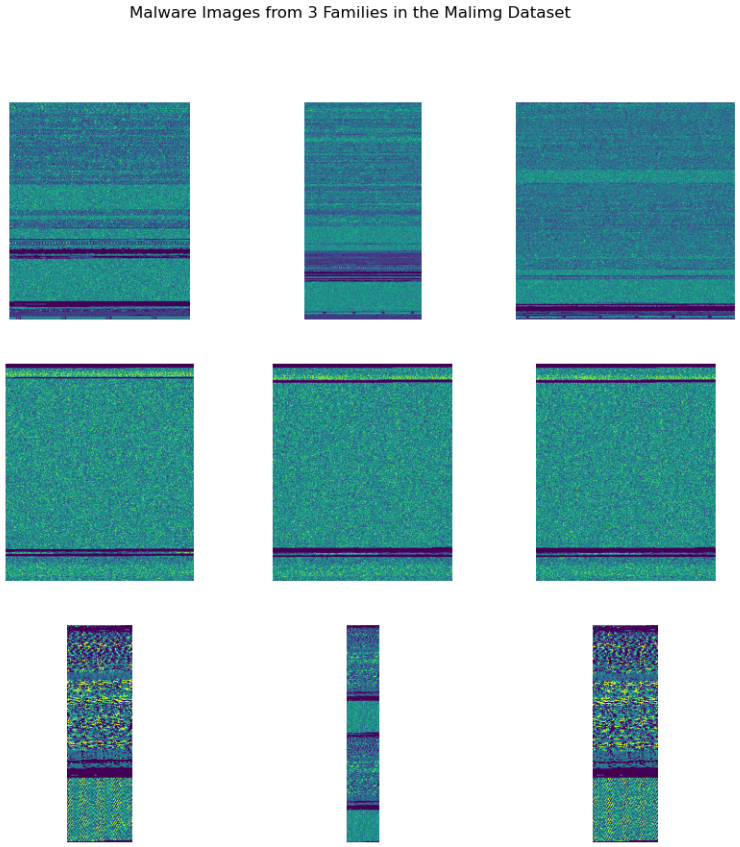

Figure 3.1.1: Sample Images from the Malimg Dataset.

3.1 Training Data

The Malimg dataset, introduced by Nataraj [32] in 2011, contains 9,345 grayscale malware images labeled by their 25 families. An example of samples from the Malimg dataset is provided in Figure 3.1.1. Images with similar textures and contours correspond to malware in the same family.

Table 5.4.6 provides an overview of the malware families included in

the Malimg dataset and their behaviors. It can be seen that multiple families of malware exist to carry out the same malicious task(s). This table referenced again in Chapter 5 to connect malware families used by different threat actors to the actual attacks they perform on systems or applications they've infiltrated.

Malware Family	**Description**
Adialer.C	Dialer
Agent.FYI	Trojan-Dropper
Allaple.A	Worm
Allaple.L	Worm
Autorun.K	Worm
C2LOP.P	Trojan
C2LOP.gen!g	Trojan
Dialplatform.B	Dialer
Dontovo.A	Trojan-Downloader
Fakerean	Roge/Ransomware
InstantAccess	Dialer
Lolyda.AT	Trojan-Theft
Malex.gen!J	Trojan
Obfuscator.AD	Encrypts malware
Skintrim.N	Trojan
Swizzor.gen!E	Trojan-Downloader
Swizzor.gen!I	Torjan-Downloader
VB.AT	Various
Wintrim.X	Trojan-Downloader
Yuner.A	Worm

Table 3.1.1: Malware Families in the Malimg Dataset and Behavior According to Microsoft's Security Encyclopedia [26].

The raw images in the Malimg dataset have variable sizes. The largest image in the dataset is 512 by 903 pixels and from the VB.AT family. Figure 3.1.2 shows the distributions of sizes of images in the Malimg dataset by family. Most images have similar areas, but those from the VB.AT are consistently higher. It is worth noting that

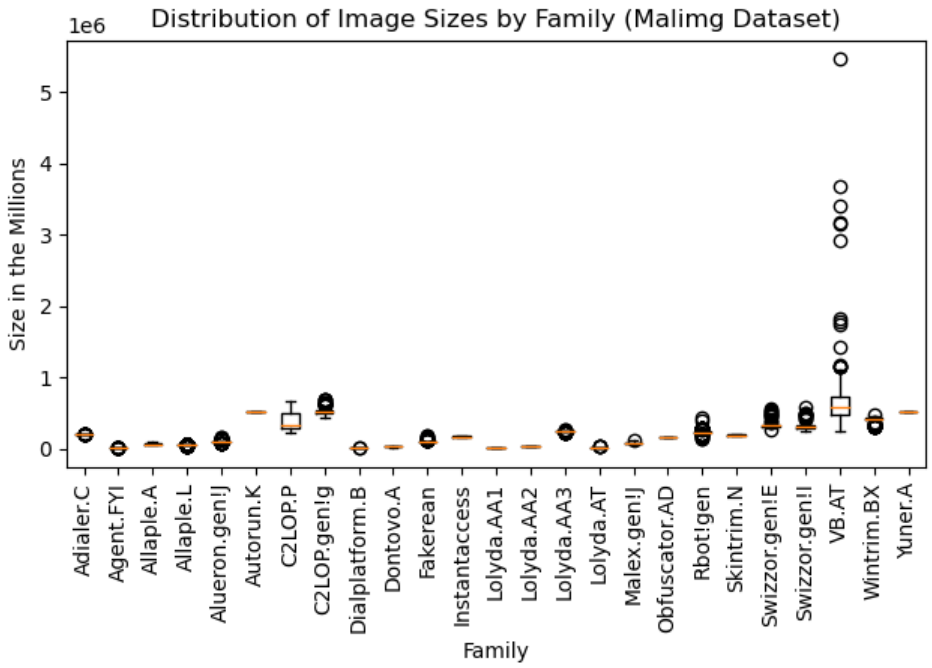

Figure 3.1.2: Original Sizes of Images in the Malimg Dataset, by Malware Family.

images with similar areas are not guaranteed to have similar dimensions: as evident in Figure 3.1.1, some run wider and others narrower.

The Malimg dataset is imbalanced: there are more samples attributed to some malware families than others. Figure 3.1.3 summarizes the number of images per family in the Malimg dataset. In classification, there is a risk that models learn the features of the class that occurs most frequently in their training data, and then over-predict that class when presented with new data. To mitigate this, I used a "balancing" parameter whenever possible. For the random forest and support vector machine models, this meant that all

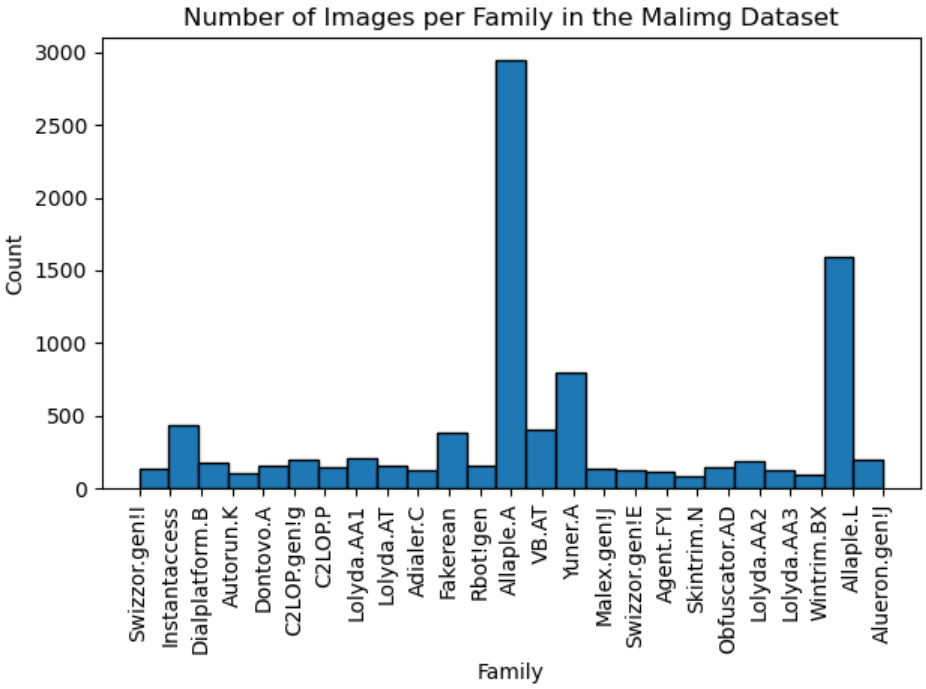

Figure 3.1.3: Number of Images in the Malimg Dataset by Family.

images were weighted inversely to the number of other samples from their family included in the Malimg dataset. Sci-Kit Learn, the Python library used to build the models described in Chapter 4, weights the samples in a class j using the formula in Equation 3.1. $N_{samples}$ is the total number of samples in the dataset, $n_{classes}$ is the number of classes in the dataset, and N_{class_j} is the number of classes categorized as class j.

$$w_j = \frac{N_{samples}}{n_{classes} N_{class_j}} \tag{3.1}$$

Essentially, the average number of samples per class is divided by the number of samples in class j. So if class j contains more samples than the average class, each will be down-weighted when training the classifier.

4

Models & Methodology

The previous chapter provided an outline of the APT dataset, which is used to compare the techniques of APT groups and malware strains with various national origins, as well as learn about changes in commonly-used malware families used over time. The previous chapter also described the Malimg dataset, which is used to train the models that identify malware families from images of malware binaries. This chapter opens with a high-level overview of the process by which I assigned a malware family to each binary in the APT dataset. Then I discuss the parameters and training of the two clustering and four classification models used, evaluating them based on accuracy, precision, recall, and F1 scores.

4.1 High Level Process

I began by dividing the Malimg dataset into a training and validation set using a random 80% training to 20% validation split. This amounted to 7,486 malware binaries in the training set and 1,860 binaries in the validation set.

Next I performed image pre-processing. In order to use the VGG16 model for feature extraction, each image needed to be of size 224 by 224 pixels. I used the `target_size` parameter in Keras' `flow_from_directory` function to resize images. For images narrower than 224 pixels, this command shifted pixels from lower in the image upwards until it could fill in rows of width 224. For images larger than 224 by 224 pixels, the command resulted in cropped images. Although some loss of information is inherent in this approach, it is preferable to performing data augmentation such as zooming in or out on images. That would seemingly preserve more visible features of each binary, but it would fundamentally change the values of and relationships between pixels. Since this is how all context about the underlying binaries is conveyed, I avoided altering it in order to preserve the integrity of the data. Previous works have resized images in the same way without loss of classification accuracy [22].

After image pre-processing, I performed feature extraction on all of the samples in the Malimg dataset using the VGG16 model. VGG16 contains thirteen convolutional layers and three fully-connected layers. In order to perform feature extraction without classification, I removed the 3 fully-connected layers and passed the data through the 13 convolutional layers.

After feature extraction, the result was a tabular dataset with 4,096 features per image. A classification model trained on thousands of features is prone to multicollinearity and overfitting. Multicollinearity leads to instability (in the event that one predictor is removed or

unavailable) and hurts interpretability. Principal Component Analysis (PCA) provides a simple and effective way to reduce dimensionality. PCA represents data through linear combinations of its original features. Each possible linear combination of the features is ranked in order of how much of the data's variance it explains. The linear combination that explains the most variance is the first principal component, and so forth. To systematically select a number of components to keep, I examined the percentage of variance explained by each component, and stopped including principal components when each stopped adding a substantial amount of variance explained. The percent of variance explained is calculated as a ratio of a given component's eigenvalue to the sum of all components' eigenvalues. Letting x_i represent the i^{th} principal component and $\mathbf{C}$ represent the covariance matrix of all features in the dataset, the eigenvalue of the i^{th} principal component is the λ that satisfies Equation 4.1. The principal components with the highest eigenvalues have nonzero coefficients where the covariance between two features is greatest, meaning that they capture the most variance in the feature space and can reliably represent the variance provided by all features.

$$\mathbf{C}\vec{x_i} = \lambda\vec{x_i} \tag{4.1}$$

Figure 4.1.1 provides a scree plot visualizing the amount of variance explained by each of the first 40 principal components, in terms of their eigenvalues. Figure 4.1.2 shows the cumulative amount of variance in the dataset explained by the first 40 principal components. These informed the choice to use four principal components: after the fourth principal component, the eigenvalues stabilized, meaning that additional principal components would likely lead to collinearity and model overfitting. The cumulative variance explained by four

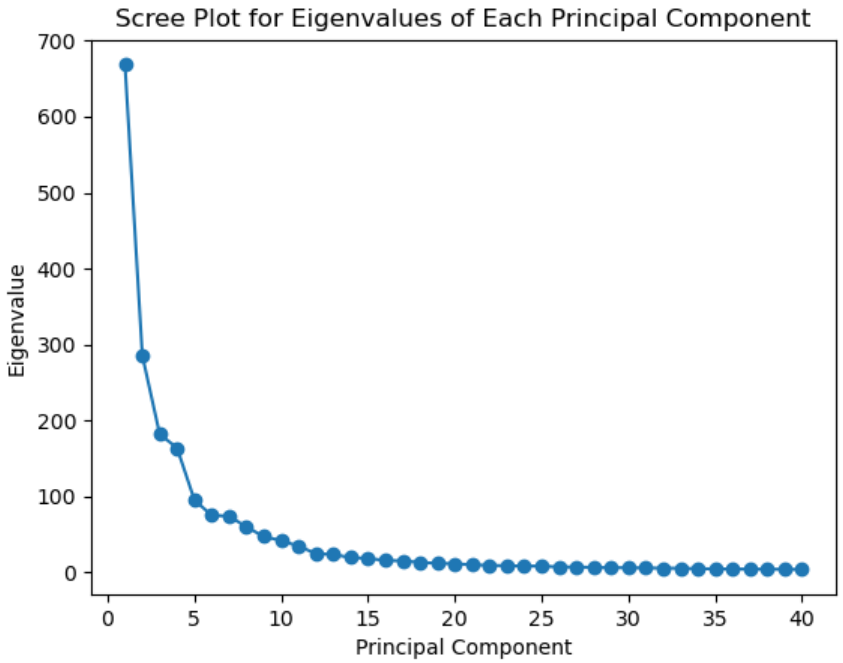

Figure 4.1.1: Eigenvalues of the First 40 Principal Components of the Malimg Dataset.

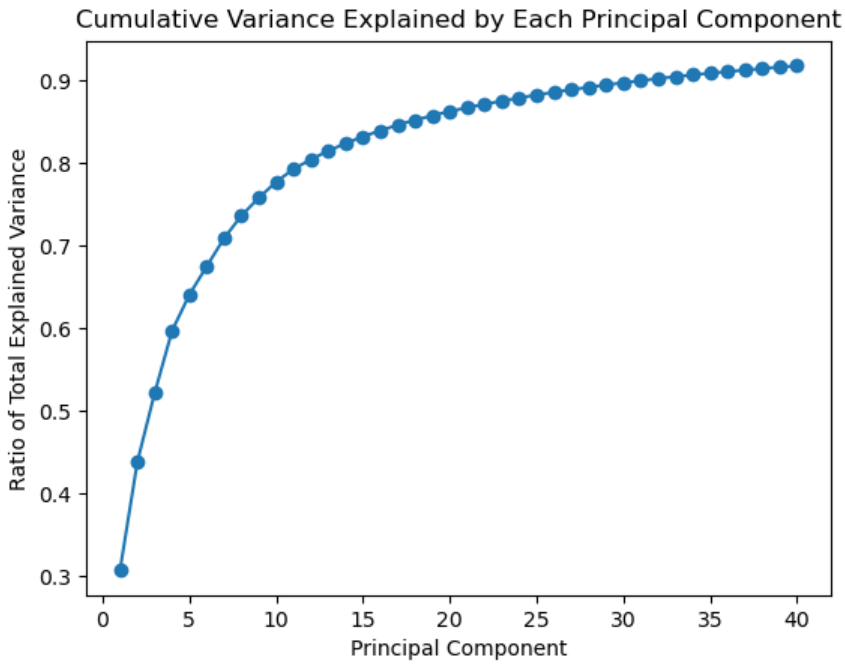

Figure 4.1.2: Cumulative Variance in the Malimg Dataset Explained by the First 40 Principal Components.

principal components is 59.75%.

Next I fit each of the models on the Malimg Training Dataset. To evaluate the quality of each model's predictions, I computed its accuracy, precision, recall, and F1 score on both the training and validation sets. I also computed a confusion matrix showing the classes to which it assigned malware binaries from the same family[1] for both the training and validation sets. The closer to a diagonal confusion matrix, the better the classifier in terms of precision and recall. To aggregate these classifiers' predictions into a single, final prediction, I took the mode.[2]

[1]The full confusion matrices for all models can be found in Appendix B.

[2]On the training dataset, all four classifiers yielded the same prediction in 91.05% of cases and three classifiers yielded the same predictio in 97.82% of cases.

I also performed open-set prediction using each classification model.[3] Traditional classifiers assign each sample in a test dataset a class that was in their training data by finding the probabilities that that sample belongs to each class and selecting the class that yields the maximum probability. This assumes that all of the classes present in the test dataset are included in the training data. In looking at the APT dataset, I could not be certain of this: the APT dataset could represent more or fewer malware families than the Malimg dataset since it was created years after the Malimg dataset, and by a different researcher.[4] Open-set prediction allows the classifier to assign a class of "unknown" to any sample if its highest probability of belonging to any class is below a certain threshold (0.75 is used here). The results of open-set prediction serve as another model diagnostic: on the training and validation data sets, low proportions of "unknown" classes indicate precise models. Unknown classes in the test data are more interesting (and less alarming) because they could represent new families of malware being developed or deployed in the time between the creation of datasets, or families that evaded detection by one researcher but were accessible to the other. However, the unknown classes still present a censored data problem. There may be multiple families of malware present in the test dataset but absent from the training dataset, yet all would be grouped into the single "unknown" category. Open-set prediction does not indicate *how many* new malware families are present, how many samples belong to each of

[3]The cluster model results were not precise enough to pursue in the final analysis. After initial investigation, I opted not to pursue open-set prediction with them.

[4]A variety of factors, including the researchers' locations, could influence the malware binaries they have access to. Malware that travels over the internet, for example, is usually intercepted by network censors and honeypots, which capture packets sent in their vicinity. If a piece of malware traveling over the internet uses the most efficient path to its target, it will more likely be picked up by censors closer to that target. So a researcher's location (specifically their proximity to targets of certain APT groups) may influence the nature of malware in their sample [25].

those families, or how similar the new families are to existing ones.

After fitting each model, I loaded each subset of the APT dataset, applied each classifier, and took the mode of the predictions to assign malware families of all samples in the dataset. I used both traditional prediction and open-set prediction. For each threat actor in the first subset of data, I plotted the frequency with which they used each family of malware in a segmented bar chart. This provided a visual overview of similarities between APT groups and malware strains, which I verified by examining the breakdowns of each malware family used by each actor, as percentages. In the next chapter I identify and interpret trends between APT groups.

Using the second subset of the data, I plotted the distribution of malware families used each year. As mentioned in Chapter 3, there is no distinction between threat actors in this subset—it is purely intended to illustrate how the most popular malware techniques shift over time. From these results, I can track the development and attrition of new malware families, including how long a malware family typically remains popular. However, if some APT group or malware strain shows a tendency to use one particular family of malware, and that family was inactive until a certain date, I may be able to infer that the threat actor was also inactive through that date.

This high-level process is summarized in Figure 4.1.3.

4.2 Considering Clustering

Open-set prediction is one way to address the possibility that the APT dataset includes more malware families than the Malimg dataset. Clustering is another way to categorize multi-class data, and has the potential to provide more specificity. While classifiers learn to recognize the features associated with samples of pre-defined groups

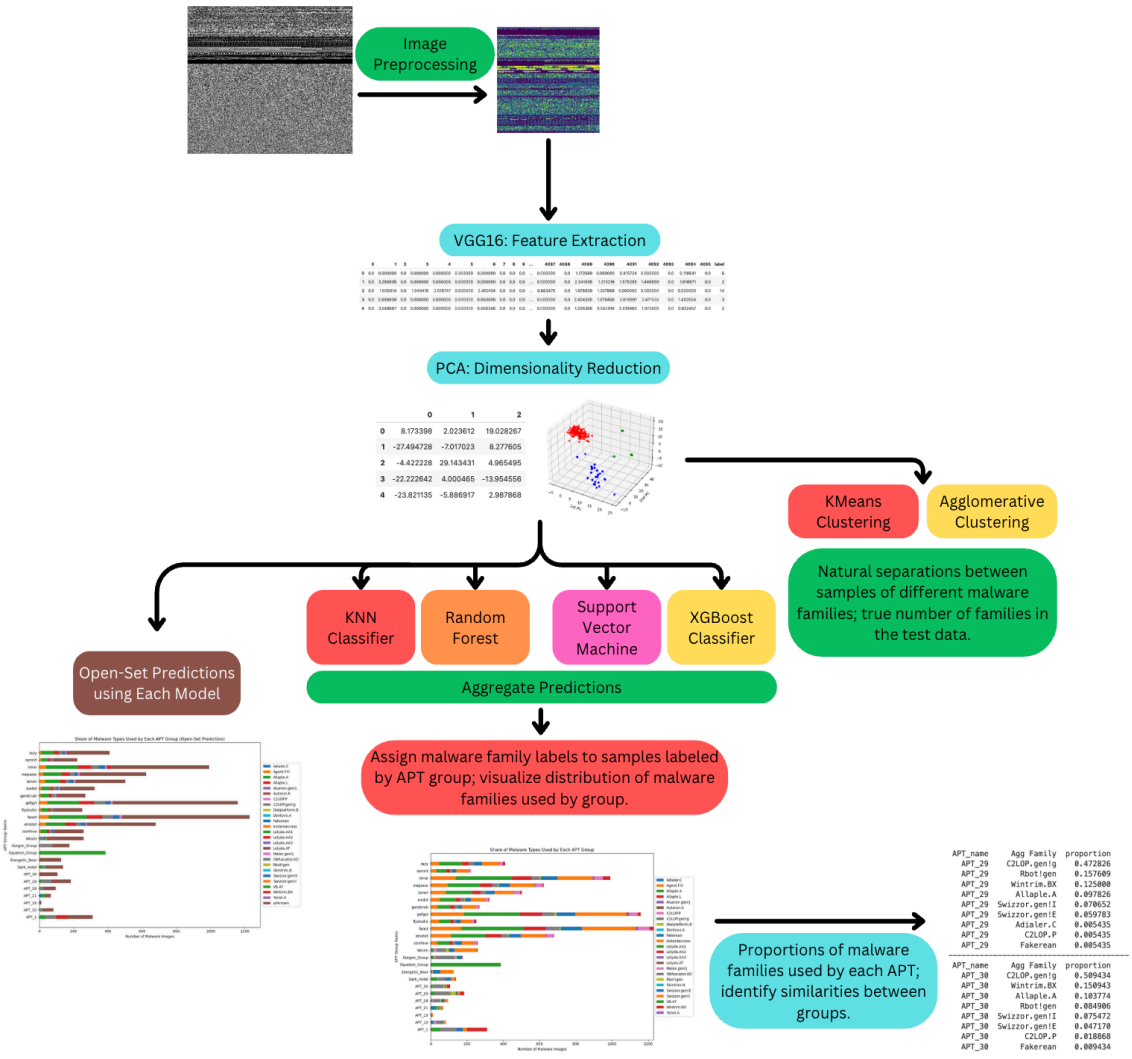

Figure 4.1.3: High Level Analysis of a Single Malware Binary.

in a correctly-labeled training dataset, clusterers form their own group definitions based on where they find natural breaks in the data they're provided. If malware samples in the training dataset exhibit smaller within-group distances than between-group distances (which can be gauged through an ANOVA F-test), then it may make sense to apply a clustering model to the test data for finding malware families because it means that malware families have distinguishable enough characteristics as to be recognized without pre-existing labels. Clustering does not yield the names of malware families in a dataset as a classifier does, but it provides a more accurate measure of how many total malware families are present in the dataset.[5]

The two cluster models employed here are KMeans clustering and agglomerative clustering. They are distinguished based on their stopping conditions: a KMeans clusterer stops splitting data once it has identified K distinct clusters in the dataset. An agglomerative clusterer begins with the assumption that each data point belongs to its own class, and then merges the classses with the smallest Euclidean distances between their samples.[6]

4.3 Model Specifications

This section describes the model specifications for the four classifiers and two clusterers used in this analysis. All models were built using Python's Sci-Kit Learn (Sklearn) library. The model results are explained in Chapter 5 and the parameter tuning processes are explained in Appendix A.

[5]The malware families of samples within a cluster may be gleaned indirectly: if the classification schemes assigned all samples in some cluster the same class, I could conclude that that cluster represented the given class.

[6]I used Ward's method for linking clusters, meaning that the agglomerative clusterer sought to minimize the within-group variance when merging clusters. It stopped merging clusters when the distance between them exceeded a pre-specified amount.

4.3.1 k-Nearest Neighbors Classification

The k-Nearest Neighbors (KNN) classifier is implemented using Sklearn's `KNeighborsClassifier` method. This classifier assigns each data point the majority class of some constant k points closest to it, in terms of Euclidean distance. I fit several kNN classifiers with k ranging from 1 to 30, and then compared their training and validation accuracy scores. I also performed five-fold cross validation on each of the models and looked at its validation accuracy scores to settle on a k value of 7 for the predictive model. The results of this analysis can be found in Appendix A.

kNN Classification should not, in theory, suffer from imbalanced classes because the dataset is sufficiently large. There are well over 7 instances of each malware family in the training dataset, so the model is not destined to reach a point whose 7 nearest neighbors are guaranteed to be in different classes (than each other, or than the point's true class).

4.3.2 Random Forest Classifier

Next I used Sklearn's `RandomForestClassifier` function to implement the random forest model. I chose to use 15 trees with maximum depth of 15 in the forest. At each split, the maximum number of features considered was the square root of the total number of features in the data.[7]

I decided on the number of trees to use and the maximum depth of those trees by comparing the training and validation errors between 340 random forest models with different numbers and trees with different depths (trees varied between 3 and 19 and depths varied between 10 and 29). I also cross-validated each of these random forest

[7]Recall that there are four features in the data after principal component analysis.

models on the entire Malimg dataset, training and validation sets combined, and found the number of trees and maximum depth where validation accuracy was high, but training accuracy did not yet reach 1.0, which would be evidence of overfitting. The model accuracy scores as a function of the number of trees in the forest and the depth of those trees are both included in Appendix A.

I chose to use balanced classes in the random forest model so that individual trees could use their depth to split data into accurate categories of approximately equal size, rather than fixating on an abnormally large class of malware and attempting to split it into inaccurate classes.

A drawback to using a random forest model in this setting is interpretability. Each tree in a random forest splits data into categories based on the values of a subset of its features. In this case the features are the first three principal components of the data, which are in turn linear combinations of the 4,096 features extracted from each image using VGG16. These features are numerical, and many of them equal zero. They cannot be easily attributed to visible characteristics in the images, making it difficult to explain how the random forest determines splits, and which characteristics of the images most distinguish malware binaries in different families from each other.

4.3.3 Support Vector Machine

I used Sklearn's `SVC` method with a linear kernel, balanced classes, and regularization parameter of 1 to build the Support Vector Machine (SVM). The linear kernel means that the SVM attempts to find a linear decision boundary between classes of points once they are transformed into a high-dimensional space.

Sklearn also includes a `LinearSVC` method to build Support Vector

Machines with linear kernels. The `LinearSVC` method works faster than the `SVC` method, but it uses a "one-versus-rest" approach to classification whereas `SVC` uses a "one-versus-one" approach. The "one-versus-one" approach is computationally more expensive, but worth it in this setting. "One-versus-one" classification compares $\binom{N}{2}$ classifiers where N is the number of classes in the data. It creates a classifier to delineate between every possible pair of classes. The "one-versus-rest" approach only deals with N classifiers, isolating each class and comparing it to all of the others (aggregated) in determining a decision boundary. Since some malware families are more similar than others in this dataset,[8] "one-versus-one" classification reduces the risk that a model conflates classes, or suffers from low recall. By directly comparing the samples in these two classes, the model can pick up on the nuances that distinguish them. With a one-versus-rest approach, it may fail to recognize a sample as a member of some class because of its similarity to some of the data included in the aggregate of all other classes.

4.3.4 Extreme Gradient Boosting

I implemented Extreme Gradient Boosting using the open source XGBoost library's Sci-Kit Learn interface. The advantage is that I had access to the same model evaluation functions as I did when building the other three classifiers, and I did not have to change the format of the training and validation data to a `DMatrix` type. I used

[8] Malware families account for the fact that malware with the same behavior and mechanisms of implementing that behavior might vary slightly in their semantics. That is, not every sample in a malware family is identical. Sometimes, the content of a malware binary diverges too far from the other samples in its family to still be considered a part of that family and a new family is instantiated with a similar name [32, 44]. For example, malware in the Allaple.L family evolved from malware in the Allaple.A family [37]. Samples in these two families are likely more similar to each other than samples in completely unrelated families, like Allaple.A and VB.AT, for example.

the histogram method for building trees. Instead of checking every sample to find an optimal split at each level of the tree, the model created bins of features and scanned them to determine splits. I also opted not to specify an early stopping condition as it hindered the percentage of observations classified as "unknown" in the training data. Otherwise, the maximum depth of each tree and the number of trees in the model were set to the same parameters as the random forest classifier.

4.3.5 KMeans Clustering

When fitting a KMeans cluster model on the training (Malimg) data, I used $K = 25$ since I knew that the Malimg dataset contained 25 classes. To evaluate the model, I created a confusion matrix between each sample's true family and the cluster it was assigned. To choose a value of K for the KMeans clusterer that I fit on the test dataset, I fit and examined the results of several KMeans clusterers that used K values between 10 and 30, since I did not know the true number of classes in the dataset. I plotted each model's silhouette scores for each cluster and considered the average silhouette score. For a cluster i, the silhouette score describes the difference in dissimilarity between points within cluster i and points in a neighboring cluster. The silhouette score is defined in Equation 4.2 where a_i is the average dissimilarity between points in the same cluster and b_i is the average dissimilarity between points in cluster i and its neighbor. Positive silhouette scores indicate better clustering than negative silhouette scores, as they mean that *dissimilarity* is greater between points in different clusters than points in the same cluster. Silhouette scores close to 1.0 indicate the best clustering because they indicate that the dissimilarity between points of the same cluster is extremely small, at least relative to the dissimilarity between points in separate clusters.

$$s_i = \frac{b_i - a_i}{max(a_i, b_i)} \tag{4.2}$$

Based on the silhouette scores, I chose to use 11 clusters in the KMeans cluster model fit on the APT dataset. The full set of silhouette scores plotted for each cluster in each KMeans model are included in Appendix A. Creating a confusion matrix for the test data did not make sense because I did not have access to the true families for the samples to compare with their predicted clusters.

4.3.6 Agglomerative Clustering

For the agglomerative clustering model, I opted not to specify a number of clusters but instead supplied a maximum distance threshold. While the distance between adjacent clusters was within this threshold, the model would merge the clusters according to Ward's method. When two clusters had a distance that exceeded the threshold, the model would not combine them. In the training case, I used the average distance between the centroids of each malware family as the distance threshold. In the testing case, I experimented with a variety of values between 135 and 195 and settled on a distance threshold of 155, which amounted to 20 clusters. However, I observed that this resulted in a lower silhouette score than the clustering established with 11 clusters in the KMeans model, so I also fit an agglomerative clustering model in which I specified that 11 clusters should be created, and the silhouette scores did in fact improve. The silhouette plots for the agglomerative clustering models using various distance thresholds, and the plot for the model using 11 clusters, are included in Appendix A.

5
Results

This chapter presents the results of assigning malware families to images in the APT dataset and begins to discuss their implications for understanding the international cyber threat landscape. I start by evaluating the performance of each model used to assign malware families to malware images, including a comparison of classification methods to clustering models. Then I explain how I assigned one malware family to each sample. I break down the distributions of malware families used by each threat actor in the data and interpret patterns between them. I also look at the distributions of malware families used by year in the APT dataset and interpret what they reveal about the top malware threats over time.

5.1 Model Evaluation

I begin by evaluating the performance of the four image classifiers used to assign malware families to individual malware images. For each model I report the training and validation set accuracy scores, as well as the macro-averaged training and validation precision, recall, and F1 scores[1] in Table 5.1.1. The accuracy score is simply the proportion of points in each dataset for which the model assigned a correct label. **Precision** measures the purity of each class, describing the proportion of observations assigned that class that truly belong to that class. The mean precision is defined in Equation 5.1 where T_i represents the number of samples correctly assigned class i and F_i represents the number of samples incorrectly assigned class i. The sum in this equation runs from $i = 0$ to 24 because there are 25 classes (malware families) in the training dataset.

$$Precision = \frac{1}{24} \sum_{i=0}^{24} \frac{T_i}{T_i + F_i} \tag{5.1}$$

Recall measures a model's ability to recognize the full spectrum of observations in its class, without missing any. Recall is defined in Equation 5.2 where T_j represents the number of samples in family j that were correctly assigned class j and T'_j represents the number of samples in family j that were mistakenly categorized elsewhere.

$$Recall = \frac{1}{24} \sum_{j=0}^{24} \frac{T_j}{T_j + T'_j} \tag{5.2}$$

The **F1 score** combines precision and recall according to Equation 5.3. It is the reciprocal of the mean of precision and recall (otherwise known as the harmonic mean of the two quantities). It is bound by 0

[1]The macro-averaged precision is the mean precision taken over all assigned classes and macro-averaged recall is the mean over all true families.

and 1.0, with values closer to 1.0 being preferable. Since precision and recall are each perfect when they equal 1.0, their arithmetic mean and reciprocal would be 1.0 when precision and recall are both perfect. If they each dip slightly below 1.0, their harmonic mean would still be higher than if one of the metrics dipped well below 1.0 and the other remained high. Thus, the F1 score balances the following natural tradeoff between precision and recall.[2]

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{5.3}$$

I also include the proportion of training data that each model classified as "unknown" when performing open-set prediction. For both the training and validation sets, low proportions of "unknown" observations are favorable because it means that the model effectively learned the features of its training data and can distinguish between classes.

In terms of the training metrics in Table 5.1.1, the XGBoost and random forest models were best. The kNN classifier yielded the best results on the validation set however. The XGBoost and random forest models show evidence of overfitting because their training metrics are all very close to 1.0 while their validation scores decrease substantially. Part of the decline in the XGBoost model's validation scores is due to its consistent misclassification of one malware family, Yuner.A, on the validation set.[3] XGBoost confounded Yuner.A with C2LOP.gen!g. This may be explained by the fact that there are

[2]To imagine the tradeoff between precision and recall, consider a model that assigns every sample the same family. It would have perfect recall for that family but would be very imprecise, with a denominator inflated by a high F_i value. Of course, in multiclass classification, the macro-averaged recall would *not* be perfect for such a model.

[3]A full confusion matrix for this model's predictions on the validation set is included in Appendix B.

Metric	kNN	Random Forest	SVM	XGBoost
Training Accuracy	0.9709	0.9965	0.9462	0.9933
Validation Accuracy	0.9548	0.9468	0.9355	0.8323
Training Precision	0.9261	0.9920	0.9231	0.9989
Training Recall	0.9362	0.9932	0.8890	0.9839
Training F1 Score	0.9295	0.9926	0.9015	0.9907
Validation Precision	0.8810	0.8669	0.8771	0.7546
Validation Recall	0.8920	0.8871	0.8689	0.7482
Validation F1 Score	0.8814	0.8717	0.8660	0.7243
Unknown Train	0.0572	0.0545	0.1035	0.0235
Unknown Validation	0.0586	0.1086	0.1280	0.1962

Table 5.1.1: Model Evaluation Metrics for 4 Image Classifiers on the Malimg Dataset.

relatively few samples of the C2LOP.gen!g family in the validation set (40).[4] But even when balancing classes, the prediction did not improve. To assess the XGBoost model's performance on all other classes, I fit one XGBoost model on data that excluded the Yuner.A family. The metrics are reported in Table B.4.1 of Appendix B and are much improved from those of the original model.

The support vector machine seems to be the weakest model on the training data, which is somewhat at odds with previous results [8, 10], although its metrics are not drastically lower than those of the other classifiers.

For each model, I plotted a confusion matrix, showing how many times malware samples from the same family were assigned the same class. Diagonal matrices are desirable because they indicate that almost all malware samples in the same family were identified as such, and that few were assigned a family other than their true one. The

[4] In initial iterations, the XGBoost model did not confound Yuner.A at all. This is explained in depth in Appendix B.

full set of confusion matrices for each individual model are included in Appendix C.

5.2 Cluster Model Results

Next I discuss the results of two clustering models used to categorize the Malimg dataset. Unlike classifiers, clustering models need not be explicitly fit on training data to later categorize test data because they learn natural groups present in that test data. The reason for fitting the clustering models on the Malimg dataset was to determine if clustering was a reasonable method for identifying malware families at all. That is, if the families in the Malimg dataset had similar characteristics enabled them to be split neatly.

5.2.1 *K*Means Clustering

I began by fitting a *K*Means cluster model with $K = 25$ clusters. Labels assigned by the clusterer do not correspond to indices of the malware families (as they do in a classifier) because cluster models assign samples to groups as they identify the features of those groups. This can occur in an arbitrary order relative to family indices. I plotted the spread of samples from each true family across clusters in the confusion matrix in Figure 5.2.1. The lack of a clear diagonal is not a problem for this model since indices do not correspond to malware family names. Instead, I look to see if samples from the same malware family are spread across multiple clusters, which would indicate a model with low recall. I see that samples from the Allaple.A family are sorted into about four different clusters and that samples from Allaple.L are split between two. Despite being semantically related, samples from Allaple.A and Allaple.L are never sorted into the same cluster. Related samples from Swizzor.gen!E and Sizzor.gen!I, on the other hand, are sorted into the same clusters

(clusters 1 and 13, in particular). In addition to the samples from Swizzor.gen!E and Sizzor.gen!I, cluster 1 also contains most of the samples from the Rbot!gen malware family, potentially indicating low precision for this model.

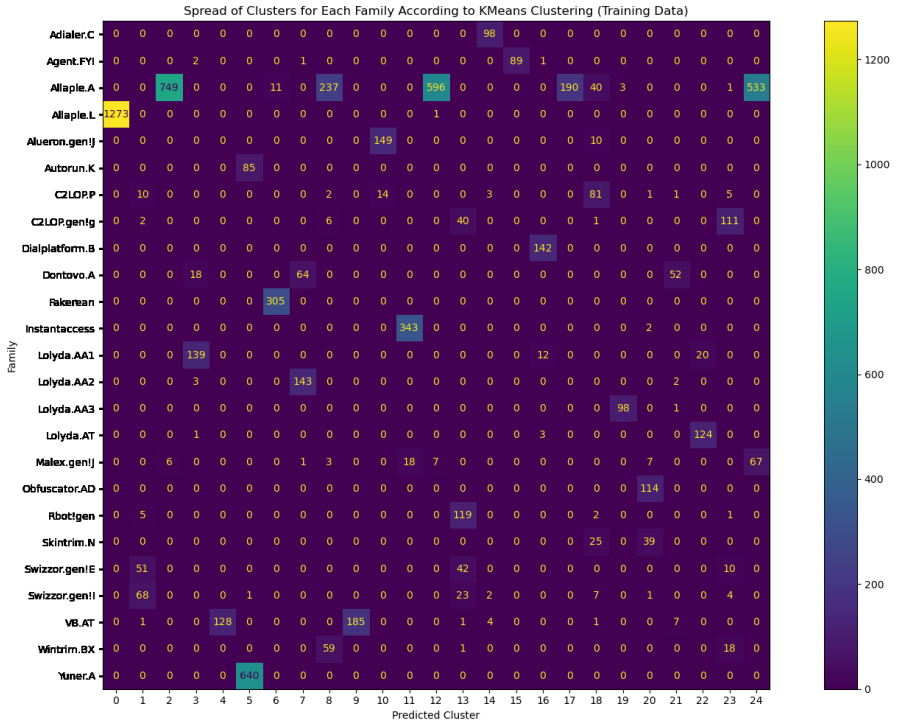

Figure 5.2.1: Malware Families and Assigned Clusters of the Training Dataset, According to KMeans Clustering.

5.2.2 Agglomerative Clustering

I also fit an agglomerative clustering model. For the training dataset, I chose to specify 25 clusters as opposed to setting a distance threshold above which the model would stop merging clusters. Figure 5.2.2 shows the spread of clusters assigned to samples from each

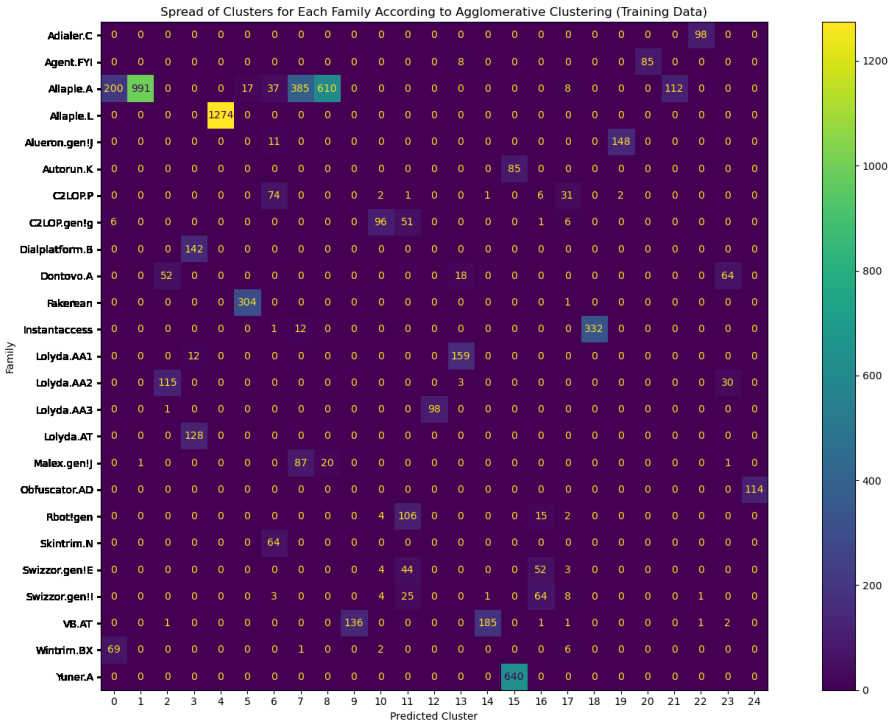

Figure 5.2.2: Malware Families and Assigned Clusters of the Training Dataset, According to Agglomerative Clustering.

malware family using this method.

The confusion matrix from the agglomerative clusterer shows that there are not many malware families whose samples are split between clusters. It remains the case that samples from the Allaple.A family are split between clusters, but the clusters they are split between tend not to contain samples from other malware families as well. This is a positive result, indicating that there is variance within the Allaple.A malware family, which makes sense given that Allaple.A is one of the families that eventually became so varied and evolved into another (Allaple.L). In addition, the two clusters of Swizzor.gen!E and Swizzor.gen!I samples still contain samples from each other's families with about equal frequencies, as was the case in the confusion matrix

Metric	KMeans Model	Agglomerative Model
Precision	0.7987	0.8329
Recall	0.8127	0.8243
F1 Score	0.8057	0.8286

Table 5.2.1: Estimated Precision, Recall, and F1 Score for the Two Clustering Models on the Malimg Dataset.

of KMeans clustering.

I estimated the precision of the two clustering models by finding the greatest number of samples in each cluster belonging to the same family and dividing it by the total number of samples in that cluster, and then taking the average over all clusters. To estimate recall, I found the maximum number of samples from each family that were clustered together, and divided by the total number of samples in that family, and averaged over all families in the dataset. Then I used these two estimates to estimate an F1 score. The metrics are included in Table 5.2.1. Each is substantially lower than the metrics of the classifiers, which is why I only included the classification models in my final prediction scheme, described in the next subsection.

5.3 Assigning Malware Families

Before aggregating the results of the four classifiers and assigning one malware family to each sample in the APT dataset, I applied each individual classifier to the samples in the APT dataset and plotted the distributions of malware families used by each threat actor, according to that classifier. The full results are included in Appendix C and the remainder of this section discusses the aggregate results.

I used the mode class predicted by the classifiers to assign one

malware family to each sample in the data.[5] This is because each of the classifiers had relatively high accuracy and its own merits for being a reasonable model (based on the design choices discussed in Chapter 4). There was no obvious way to drop one of the models entirely. Applied to the APT dataset, at least three of the four classifiers yielded the same prediction in 52.84% of the samples, and all four of the classifiers yielded the same prediction in 26.89% of the samples.[6]

Considering the results of four classifiers to determine a final prediction enhances the probability that the prediction is correct. Consider the case in which prediction is based on only one classifier. Let T_i represent the true malware family of sample i, let X_i represent the final predicted family of sample i, and let A_i be the family predicted by the single classifier. Conditional on the single classifier's prediction, the probability that the final prediction is the true class is given by:

$$P(X_i = T_i | A_i = T_i) = \frac{P(A_i = T_i | X_i = T_i) P(X_i = T_i)}{P(A_i = T_i)} \tag{5.4}$$

Now consider the case in which three classifiers are used. Let B_i represent the family predicted by the second classifier and C_i represent the family predicted by the third. Now X_i will be determined conditional on A_i, B_i, and C_i all predicting class T_i. That is:[7]

[5] I considered using a Super Learner to generate a prediction from the four models. This technique would have used a meta-function to learn how to optimally combine the results of the four classifiers for the most accurate predictions. When experimenting with a Super Learner that used Multi-Class Logistic Regression as the meta function, I found that the classification accuracy, 62%, was lower than that yielded by finding the mode of the four predictions determined by each classifier.

[6] In cases where there was no true mode, the model defaulted to predicting the family assigned by a) two out of the four models, if the other two yielded different predictions from one another or b) kNN classification

[7] Please note the assumption that each of the classifiers *independently* predicts the

$$P(X_i = T_i | A_i = B_i = C_i = T_i) = \frac{P(A_i = B_i = C_i = T_i | X_i = T_i) P(X_i = T_i)}{P(A_i = T_i) P(B_i = T_i) P(C_i = T_i)} \tag{5.5}$$

In Equations 5.4 and 5.5, the probabilities $P(A_i = T_i | X_i = T_i)$ and $P(A_i = B_i = C_i = T_i | X_i = T_i)$ are both equal to 1.0. In the single-model case, X_i will be equal to T_i in every case where the single classifier's prediction, A_i, is T_i. In the multi-model case, X_i will be equal to T_i whenever the three classifiers predict T_i. So the following are true:

$$P(X_i = T_i | A_i = T_i) = \frac{P(X_i = T_i)}{P(A_i = T_i)} \tag{5.6}$$

$$P(X_i = T_i | A_i = B_i = C_i = T_i) = \frac{P(X_i = T_i)}{P(A_i = T_i) P(B_i = T_i) P(C_i = T_i)} \tag{5.7}$$

Based on Equations 5.6 and 5.7, one can conclude that the probability of the final predicted class, X_i, being the true class for sample i will always be greater when three classifiers are used than when one classifier is used. Since $P(A_i = T_i)$, $P(B_i = T_i)$, $P(C_i = T_i)$ are all necessarily less than or equal to 1.0, their product will be less than or equal to $P(A_i = T_i)$. Thus, their reciprocal will be greater than the reciprocal of $P(A_i = T_i)$ alone, meaning:

$$\frac{P(X_i = T_i)}{P(A_i = T_i)} < \frac{P(X_i = T_i)}{P(A_i = T_i) P(B_i = T_i) P(C_i = T_i)}$$

$$\mathrm{P}(\mathrm{X}_i = T_i | A_i = T_i) < P(X_i = T_i | A_i = B_i = C_i = T_i) \qquad \blacksquare \tag{5.8}$$

class T_i. I consider this reasonable since none of the classifiers are given information about any of the other results before making their predictions.

Metric	Final Predictive Model
Training Accuracy	0.9866
Validation Accuracy	0.9543
Training Precision	0.9647
Training Recall	0.9735
Training F1 Score	0.9694
Validation Precision	0.8822
Validation Recall	0.8967
Validation F1 Score	0.8855
Unknown Train	0.0546
Unknown Validation	0.0742

Table 5.3.1: Model Evaluation Metrics for Aggregate Predictions of 4 Image Classifiers on the Malimg Dataset.

The predictions that considered the mode of four classifiers were not only better than those of each individual classifier in theory but also in practice. Table 5.3.1 shows the classification accuracy, precision, recall, and F1 scores of the final (mode) predictions, as well as the percentage of observations classified as "unknown" under open-set prediction, on the Malimg dataset.

5.4 Comparing Techniques Between Threat Actors

After developing final predictions for the malware families of samples in the APT dataset, I compared the families used by different threat actors. In this section I consider APT groups apart from malware strains because APT groups can indicate dynamics between states in the cyber domain whereas malware strains encompass threats leveraged by either states or cybercrime groups. However some observations about similarities between malware strains are

discussed in this section.

I started with a visual analysis of the share of malware families deployed by each APT group (Figure 5.4.1). By looking at the portion of each malware family in each bar, I develop intuition as to which APTs had similar patterns of malware use. For example, I observed that APT29 and APT30 used the same malware families with similar frequencies.

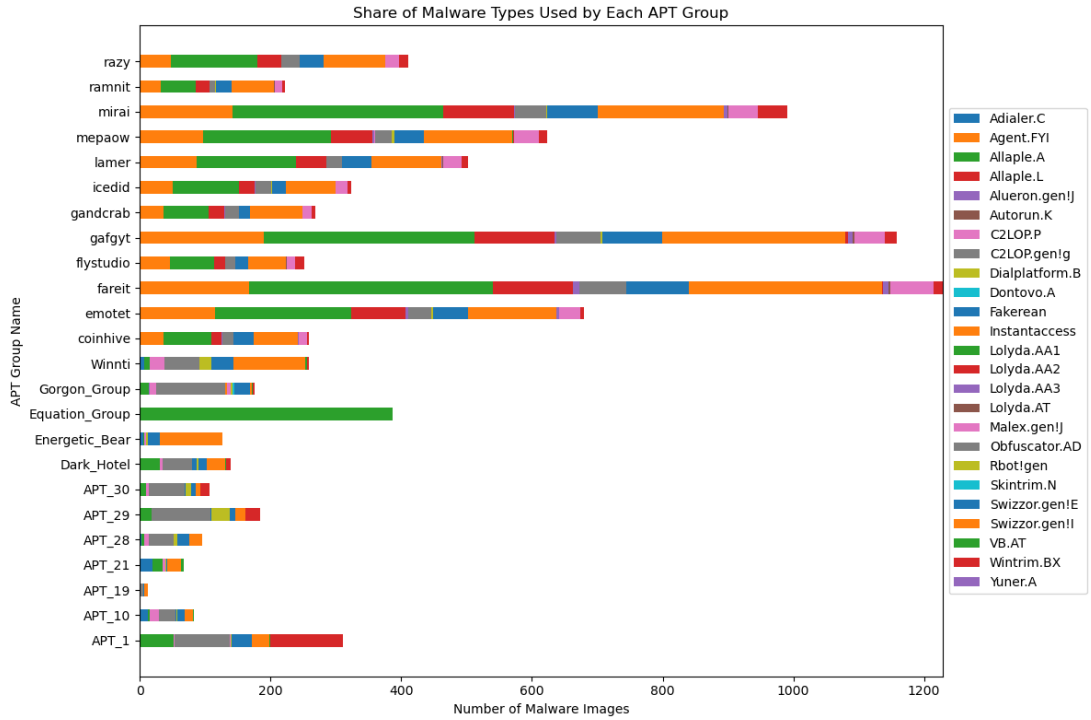

Figure 5.4.1: Frequency with which each APT Group in the Dataset uses each Malware Family.

The visual analysis allows me to focus my comparison on APT groups that use the same malware families as each other at all, and avoid comparing two groups if they do not use any of the same malware families. But visual analysis only goes so far. For each APT

Group, I computed the proportion of their total malware use attributed to each family to verify any visually-perceived similarities. Tables 5.4.1 and 5.4.2 show the frequencies with which APT29 and APT30 used various malware families. Their most frequently-used malware families are the same, but appear in slightly different orders. Both APT29 and APT30 rely on C2LOP.gen!g in more than half of their samples. They also display similar frequencies of using Swizzor.gen!E and Rbot!gen malware.

Malware Family	**Proportion of Usage**
C2LOP.gen!g	0.5870
Swizzor.gen!I	0.1304
C2LOP.P	0.1196
Allaple.A	0.0652
Swizzor.gen!E	0.0435
Rbot!gen	0.0326
Wintrim.BX	0.0218

Table 5.4.1: Frequency of Malware Families Used by APT 29.

Malware Family	**Proportion of Usage**
C2LOP.gen!g	0.5566
Wintrim.BX	0.1226
Swizzor.gen!I	0.0849
Swizzor.gen!E	0.0755
Allaple.A	0.0566038
C2LOP.P	0.0566
Rbot!gen	0.0283
Allaple.L	0.0189

Table 5.4.2: Frequency of Malware Families Used by APT 30.

As mentioned in Chapter 3, APT29 is affiliated with Russia and APT30 with China (PRC). Given the similar political motives of

these two nations, it is not surprising that they would support APT groups that employ similar techniques. The nations may have some similar targets or missions for which these techniques have proven reliable. What is more surprising is that a third group, DarkHotel, from South Korea, also had a similar profile to APT29 and APT30. The proportions of malware families employed by DarkHotel are given in Table 5.4.3.

Malware Family	**Proportion of Usage**
C2LOP.gen!g	0.3669
Allaple.A	0.1871
Swizzor.gen!I	0.1655
Swizzor.gen!E	0.1079
Wintrim.BX	0.0719
C2LOP.P	0.0647
Rbot!gen	0.0144
VB.AT	0.0144
Malex.gen!J	0.0072

Table 5.4.3: Frequency of Malware Families Used by DarkHotel.

Considering the results of APT29, APT30, and DarkHotel together, it seems clear that malware families are not used uniquely by individual states, or groups of allied states. Instead, states and malware families exist in a many-to-many relationship. This means that a given malware family is used by many states, and that a given state uses many malware families. Intuitively, this is consistent with other domains of warfare in which multiple states use the same types of weapons, but against different targets. In other domains states also possess multiple types of weapons (no state relies on a single weapon in any domain of warfare). This is the dynamic I find emerging among APT groups and malware families in the cyber domain.

To strengthen the point that the same state may employ multiple malware families for different ends, or against different targets, I

analyzed the malware families used by two APT groups with the same political affiliation: APT10 and APT21, both from the PRC. The profiles of these two groups are given in Tables 5.4.4 and 5.4.5.

Malware Family	**Proportion of Usage**
C2LOP.P	0.4146
C2LOP.gen!g	0.2927
Swizzor.gen!I	0.1098
Swizzor.gen!E	0.0976
Allaple.A	0.0732
VB.AT	0.0122

Table 5.4.4: Frequency of Malware Families Used by APT10.

Malware Family	**Proportion of Usage**
Swizzor.gen!I	0.3284
Allaple.A	0.2388
Adialer.C	0.1194
C2LOP.P	0.1194
VB.AT	0.0896
Swizzor.gen!E	0.0746
Rbot!gen	0.0299

Table 5.4.5: Frequency of Malware Families Used by APT21.

Both APT10 and APT21 use C2LOP.P, Swizzor.gen!I, Allaple.A, Swizzor.gen!E, and VB.AT malware. But they do so with vastly different frequencies. Furthermore, a malware family used by APT10 in nearly one-third of cases (C2LOP.gen!g) is missing from the set of malware families used by APT21. Even though APT10 and APT21 are both affiliated with the PRC, they likely have different ordinances or targets.

In Chapter 3 I provided a table outlining the behavioral characteristics of each malware family included in the Malimg dataset.

It is provided again in this section (Table 5.4.6) for reference. Looking at the Table 5.4.6, I can identify the behaviors most utilized by APT groups in the dataset. APT 29, for example, used a combination of Trojans and Worms. In the context of a state-sponsored operation, these malware families make sense. Trojans can stealthily infiltrate a network and then release further malware therein, and the Worms can infect numerous devices on that same network. This was the type of behavior exhibited by Stuxnet, a state-sponsored attack by the United States and Israel on Iranian nuclear centrifuges in 2010 [7, 19]. It also makes sense that there is an absence of dialers and ransomware in the malware families used by APT29 and APT30, as these are associated with cybercrime more than they are associated with prolonged cyber campaigns carried out by nation states. Indeed, the InstantAccess dialer appears among the malware families used as part of malware strains such as coinhive, emotet, and fareit in this dataset.[8]

5.5 Temporal Trends

In this section I discuss trends in the global malware landscape over time. The APT dataset includes samples of malware deployed each year between 2011 and 2023, but the actors who deployed these samples are unknown. By applying my classification method to these samples, I can discover the most common types of malware used each year.

The result could suffer from sampling bias in the case that certain families of malware are systematically included in the dataset more

[8] A note on similarities observed between malware strains: coinhive, emotet, fareit, gafgyt, and mirai seem to use the same types of malware in the same frequencies based on visual analysis. The full breakdown of malware families they employ is included in the Jupyter Notebook attached to this thesis' GitHub and available at https://github.com/cathystanton/senior-thesis.

Malware Family	Description
Adialer.C	Dialer
Agent.FYI	Trojan-Dropper
Allaple.A	Worm
Allaple.L	Worm
Autorun.K	Worm
C2LOP.P	Trojan
C2LOP.gen!g	Trojan
Dialplatform.B	Dialer
Dontovo.A	Trojan-Downloader
Fakerean	Roge/Ransomware
InstantAccess	Dialer
Lolyda.AT	Trojan-Theft
Malex.gen!J	Trojan
Obfuscator.AD	Encrypts malware
Skintrim.N	Trojan
Swizzor.gen!E	Trojan-Downloader
Swizzor.gen!I	Torjan-Downloader
VB.AT	Various
Wintrim.X	Trojan-Downloader
Yuner.A	Worm

Table 5.4.6: Malware Families in the Malimg Dataset and Behavior According to Microsoft's Security Encyclopedia [26].

than others. But I assume that an oversampled family would be oversampled across years, not in just one. Thus, the result can still be interpreted to explain the relative usage levels of that malware family over time.

As I did in the analysis of APT groups, I assigned each sample in this subset of the data a family based on the mode of predictions from the four classification models. The same malware family was predicted by all four models in 29.76% of samples. In 63.57% of the samples, the same malware family was predicted by at least three of the models. Then I aggregated the predictions by year and plotted the distribution of malware families deployed each year between 2011 and 2023.

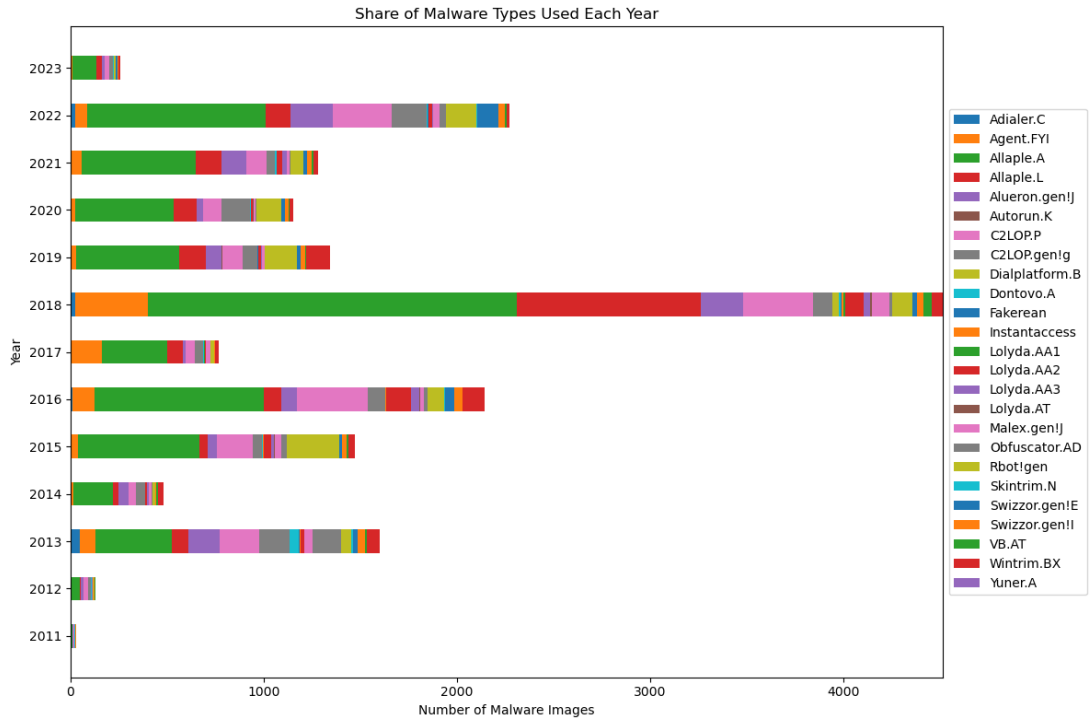

Figure 5.5.1: Distributions of Malware Used for Each Year Between 2011 and 2023.

To visualize the first result, I plotted the share of malware families detected each year in a segmented bar chart (Figure 5.5.1). Across years Allaple.A was the most-used family of malware. Allaple.A accounted for the most samples in the Malimg dataset, so it is possible that the classification scheme over-predicted this family, or that it is a very commonly-used and easy-to-sample type of malware and thus appeared the most times in the APT dataset too. In most years it was followed by Allaple.L, which is not surprising because Allaple.L is a variant of Allaple.A with similar characteristics. Both Allaple.A and Allaple.L behave as worms.

Next, I converted the counts of malware families used each year to

percentages. For each family, I plotted a line graph showing the percentage of total malware use that it represented in a given year. The result is in Figure 5.5.2 and helps showcase how the most commonly-used malware types change over time. Allaple.A persists in popularity, but in 2012, C2LOP.P peaked in activity. Rbot!gen peaked in activity in 2015, Agent.FYI in 2017, and Allaple.L in 2018.[9] Besides Allaple.A, it seems that malware families peak for relatively short amounts of time (one year). This could be the result of the threat actors associated with these families lessening their activity levels year-to-year, or of the development of detection and defense mechanisms against these types of malware. These two possibilities are not unrelated: detection of a threat actor's malware family of choice could cause the actor to reduce its activity for a time. Such was the case with Emotet in 2021: when its malware was wiped from affected systems in January, the group remained inactive until November [11].

The full results, including line graphs for the prevalence of each malware family according to each individual classifier, are included in Appendix C.

5.6 Results from Open-Set Prediction

Lastly, I perform open-set prediction on both subsets of the APT dataset to address the possibility that malware families in the APT dataset do not directly reflect those in the Malimg dataset. I also applied two clustering models to the APT dataset to address this possibility. I start with a discussion of the results from open-set prediction.

[9]Peaks are determined both by looking at the families labeled in the legend of Figure 5.5.2 and by looking at the percentages of total malware used by year, which are printed in the Jupyter Notebook attached to this thesis' GitHub.

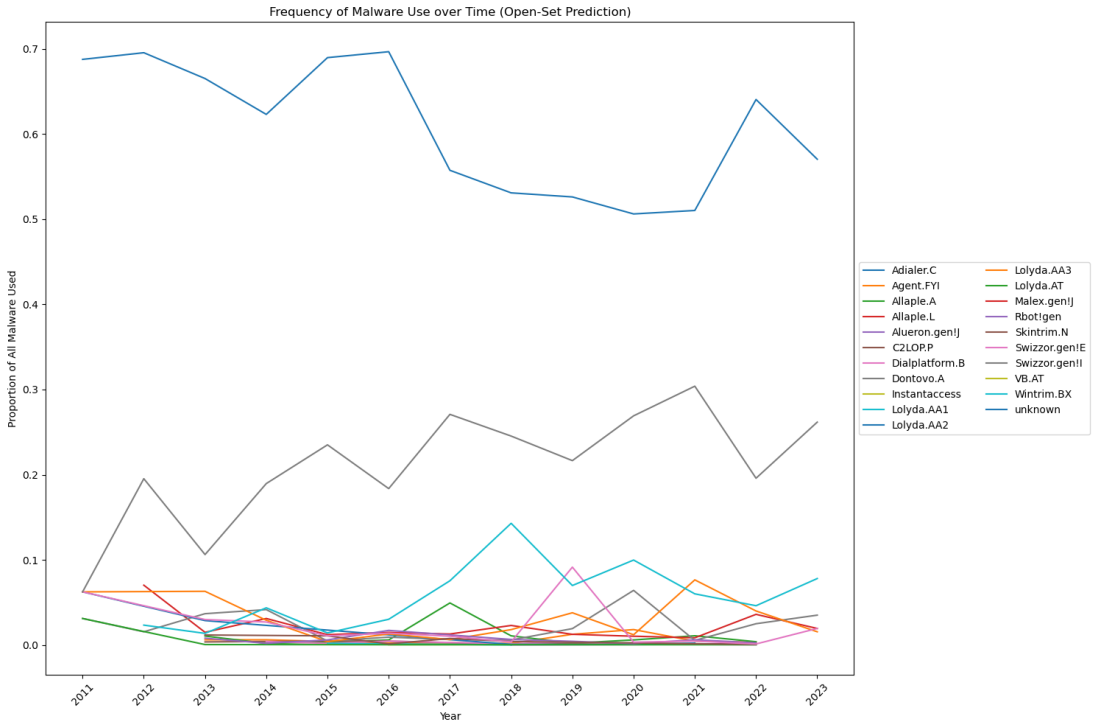

Figure 5.5.2: The Most Prevalent Malware Families Used Each Year between 2011 and 2023.

I repeated the analysis process for assigning malware families to APT groups to see if the malware samples included in the APT dataset could be described in terms of families from the Malimg dataset. Figure 5.6.1 shows the relative frequencies of each malware family used by each threat actor. Under open-set prediction, the malware families deployed by each actor were unknown about half of the time.[10] A simple way to reduce the number of classes categorized as "unknown" would be to reduce the threshold by which the models need to be sure that a sample belongs to a given class before assigning that class. This practice may not reduce the quality of prediction too

[10]The full results, including printed tables showing the percentages of each known malware family used by each APT group are provided as printed outputs in the Jupyter Notebook attached to this thesis' GitHub.

consequentially since, as explored earlier in the chapter, using four classifiers improved the probability of the final prediction being correct from each of the individual models. Other methods by which to reduce the number of classes categorized as "unknown" are discussed in Chapter 6.

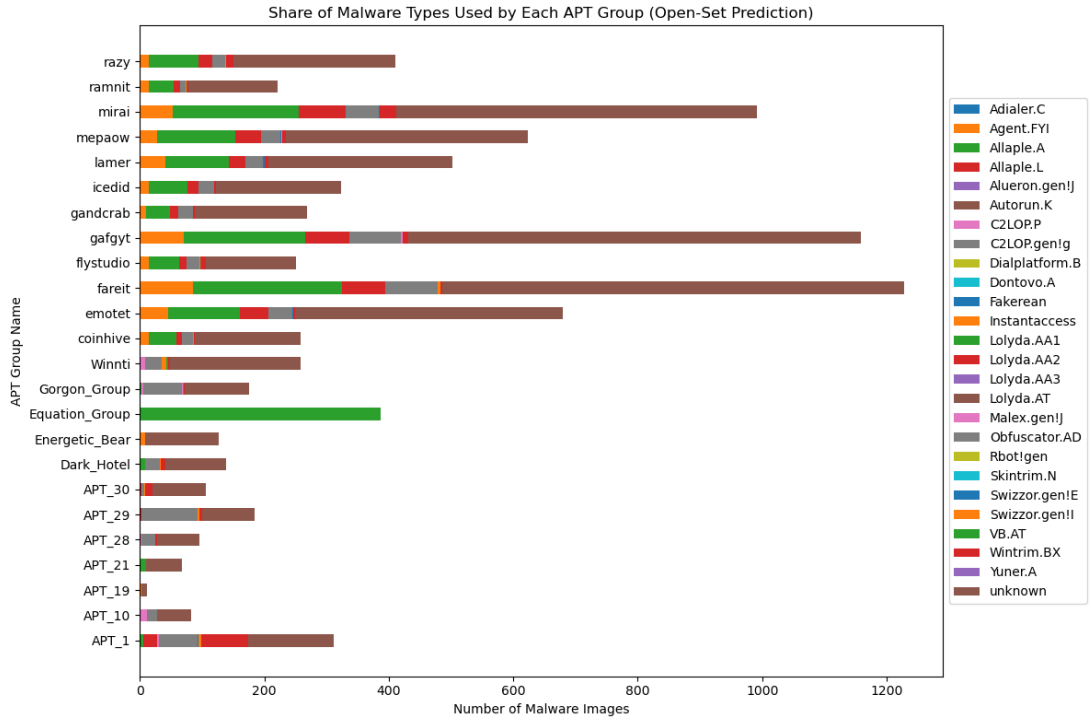

Figure 5.6.1: Frequencies of Malware Used for Each Year Between 2011 and 2023, According to Open-Set Prediction.

I also repeated the temporal analysis using open-set predictions. The analogous line chart providing the frequencies of each malware family used over time is found in Figure 5.6.2. Over time, the most common malware family detected was also unknown.[11]

[11] The full proportions of each malware type used each year are included as outputs in the Jupyter Notebook attached to this thesis' GitHub.

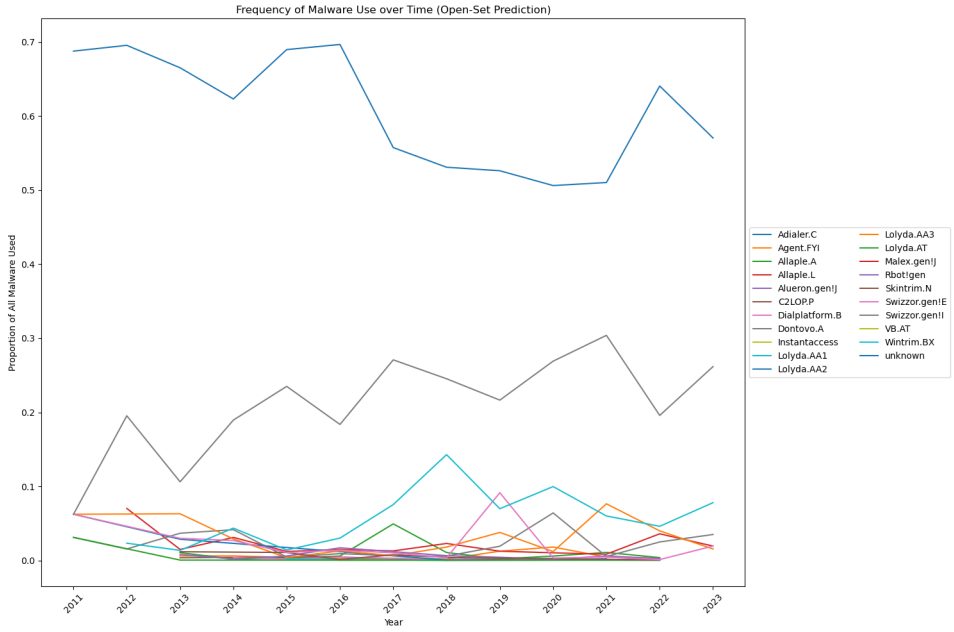

Figure 5.6.2: The Most Prevalent Malware Families Used Each Year between 2011 and 2023 based on Open-Set Prediction.

5.7 Clustering on the Test Dataset

Though clustering performed poorly on the training dataset relative to classification (in terms of precision, recall, and F1 score), I experimented with using it on the APT dataset. The purpose was to search for the true number of classes in the APT dataset and *not* to attribute families to individual samples.[12] The method by which I decided on the number of clusters to use, and the silhouette plots that informed the choice, are included in Appendix A but the distributions of malware families used by each APT group, according to each clustering, are included here. Figure 5.7.1 shows the share of clusters that each actor's malware samples were assigned as determined by a

[12]The classification scheme presented above and outlined in the previous chapter is still used for attribution.

KMeans model with 11 clusters. Figure 5.7.2 is the analogous plot based on agglomerative clustering, also with 11 clusters. The choice to use 11 clusters in the agglomerative clusterer was based on fitting six agglomerative clustering models using a different distance thresholds. Each distance threshold naturally corresponded to a number of clusters, which I extracted from each model and used in fitting the final agglomerative clusterer.

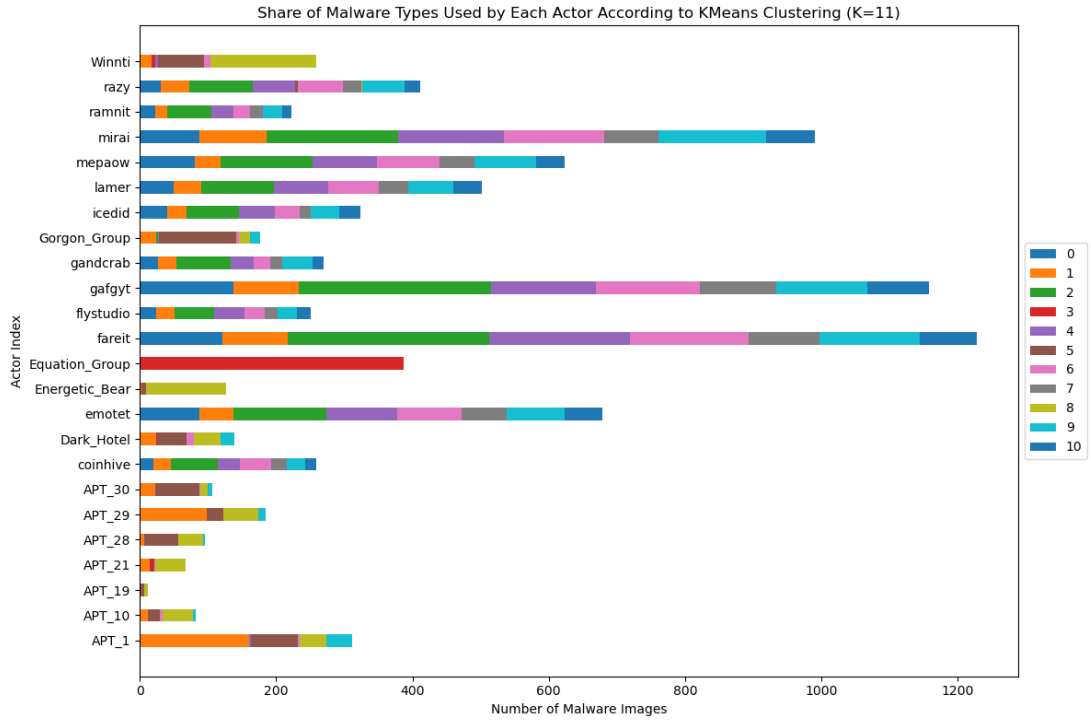

Figure 5.7.1: Distribution of Malware Types Used by Each APT Group, According to KMeans Clustering.

An encouraging result is that both clustering models determined that all malware samples from the Equation Group belonged to the same family. This was also the case in the classification models, which all categorized samples from the Equation Group as belonging to the

VB.AT malware family.[13] The two clustering models verified that the samples in the APT dataset from the Equation Group were in fact near each other in terms of Euclidean distance. Similarly, both clustering models grouped the majority of malware samples attributed to EnergeticBear in the same cluster, which is consistent with the results of most of the individual classifiers (excluding XGBoost). The results of the classification models also showed that the fareit, gafgyt, and mirai malware strains used the same malware families with similar frequencies, which is reflected in the cluster model results as well.

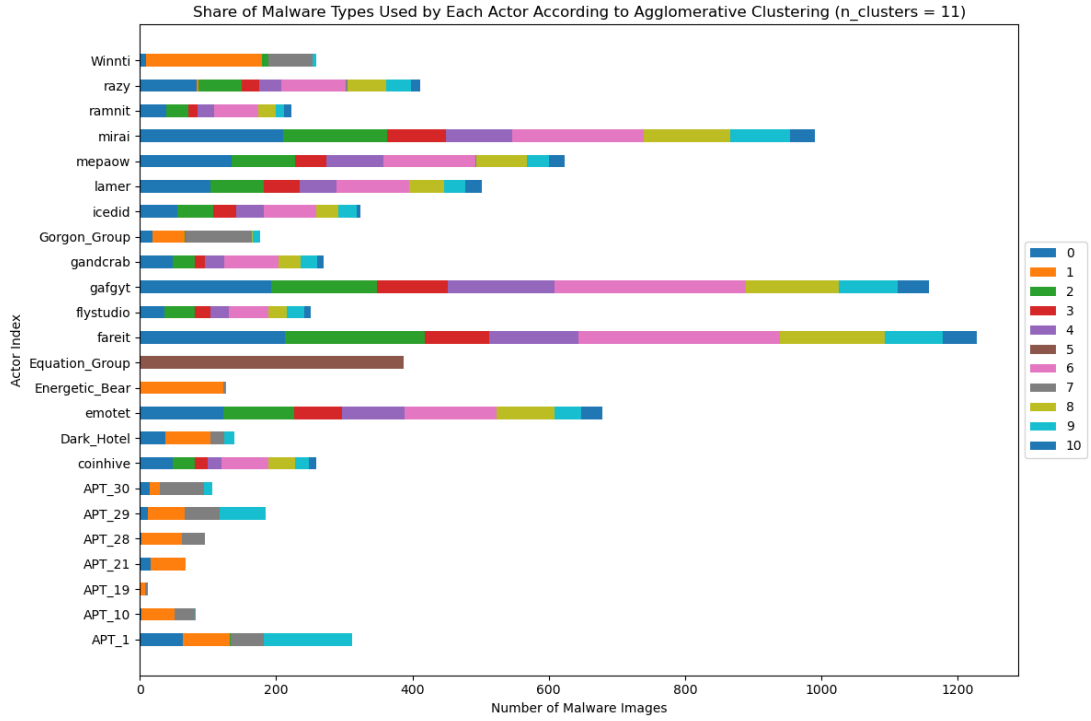

Figure 5.7.2: Distribution of Malware Types Used by Each APT Group, According to Agglomerative Clustering.

[13]VB.AT was also used by some other actors in the dataset, but with much lower frequencies than it was used by the Equation Group. The Equation Group was the only group for which VB.AT was the top malware family used, and the only group that used one family of malware.

6

Discussion & Conclusion

This thesis analyzed an open-source dataset on the IEEE Dataport which conveys malware binaries as images. In one subset of the data, binaries are attributed to the threat actor that used them (APT group or malware strain) and in the other they are labeled by the year in which they were used. By training a series of image classifiers (kNN classifier, random forest model, support vector machine, and XGBoost model) to assign malware families to these binaries, I was able to identify trends in the behaviors of different Advanced Persistent Threat actors, as well as changes in the most popular types of malware used over time. I initially wondered if it would be possible to map malware families to threat actors or nation states in a one-to-one relationship, but this seems infeasible based on the data analyzed. Threat actors with different political affiliations were found

to use the same malware families, although sometimes with different frequencies. Likewise, threat actors with the same political affiliations were found to use malware of different families. This establishes a parallel between the cyber domain and other domains of warfare: no competitive nation or threat actor relies on a single technique, and few techniques are unique to any particular actor.

The practice of mapping the malware families used by different threat actors, and the malware families used over time, can help states, sectors, companies, or other entities allocate limited resources [17] to protect their critical systems from the particular techniques used by their top adversaries, or top cyber adversaries in general. Even beyond organizations and nation states, detecting and preventing common cyber attacks on everyday technologies, including household IoT devices, promotes national security. In a digitally-dependent era, these technologies possess massive amounts of data about citizens, and are valuable to any state.

6.1 Limitations

The first limitation of this work is that only one training dataset (the Malimg dataset) was used. Previous literature leveraged the Microsoft Malware Classification Challenge Dataset [14, 31] and the Malevis Dataset [9] for classifying malware binaries into families based on grayscale byteplots. However, to use the Microsoft Malware Classification Challenge Dataset, binaries were manually converted into images before analysis, which was not a goal of this research. I did attempt to use the Malevis dataset, but my machine yielded security concerns when attempting to download it.[1] In a future work,

[1]The Malevis dataset originates from Hacettepe University in Turkey. More information may be found at: https://web.cs.hacettepe.edu.tr/~selman/malevis/.

I would download the Malevis dataset to a safe environment and append it to the Malimg dataset in order to fit models that can learn a wider array of malware families before predicting the families used by global threat actors.

Naturally, the second limitation of this work is the scope of malware families and threat actors considered. Each is considerably smaller than the full set of malware families and threat actors that currently exist or that have ever existed. But as a result of the censored data problem, discussed at length in Chapter 3, it is difficult to tell *how* much smaller they are, or if they even form a representative sample of all malware families and cyber threat actors. I hope that the techniques utilized can be replicated on larger datasets, and on datasets that represent new malware families and threat actors as they become available.

Another limitation of this analysis is the uncertainty about whether the samples in the APT dataset can be categorized into the same families as samples in the Malimg dataset. I attempted to remedy the uncertainty by performing open-set prediction and by fitting clustering models alongside classifiers. Open-set prediction allowed the image classifiers to assign a class of "unknown" to any samples for which the highest probability of their belonging to any of the classes from the training data was beneath a specified threshold (0.75 in this case). Clustering addresses the uncertainty because it defines classes based on features of samples in the test dataset themselves, as opposed to trying to form a relationship between those features and features of samples in a separate dataset. Though KMeans clustering requires a set number of clusters as a parameter, agglomerative clustering assumes that all observations begin in their own cluster and merges them in such a way that minimizes the variance in their new groups until the distance between groups to merge becomes

sufficiently large.[2] I found consistency between the clustering methods employed in this work: both suggested that the APT dataset could naturally be grouped into 11 clusters. Furthermore, both clusters recognized when all, or most, of some threat actor's malware samples should be grouped together (i.e., it recognized that all of the Equation Group's samples belonged in a family together, which was also the result of the classification models).

6.2 Future Work

An interesting extension of this work would be applying causal analysis to the prevalence of malware families over time. From a technical perspective, factors that could influence the rise or fall in popularity of a malware family could be changes in the operating systems of technologies it targeted, the development of better detection systems, or the creation of new, stealthier, more versatile malware families in its place. From a political perspective, shifting alliances, international wars, the implementation of digital firewalls, or international norm-setting[3] could all influence the types of malware deployed over time. Time series data about malware usage at a level more granular than the annual would need to be available for a causal analysis to be informative.

New representations of malware binaries, besides 2-dimensional pixel graphs, present another avenue for future research. Visualizing malware binaries in multiple dimensions may be interesting. Since it seems that malware binaries are not deployed alone and that multiple

[2]This references Ward's Method, which chooses clusters to link based on those that yield the smallest within-cluster variance. Ward's method was the linkage method used in this thesis, as mentioned in Chapter 4.

[3]At present there is very little consensus about what types of cyber attacks constitute "acts of war" and what forms of retaliation are acceptable. Standard repercussions for cyber attacks may influence the cost-benefit analyses of actors deciding what types of malware to use in some cyber offense [34].

malware families may be used in a single cyber attack, it may be useful to represent binaries as nodes in a network of binaries, and to perform APT attribution using that network. This would consider the relationships between binaries, rather than just the binaries themselves, in determining the perpetrator of a cyber attack. Under Rid and Buchanan's framework [41], it would constitute "widening the aperture" of information available about a cyber attack.

It will also be interesting to track the role of Artificial Intelligence (AI) in this field over time. In this work AI could have been leveraged to find similarities between APT groups once malware families were assigned. Classical statistics succeed in comparing two groups when their malware samples belong to the same families but in different frequencies. But when some of the classes attributed to groups differ, AI's ability to create and process complex representations of those groups is beneficial.

A

Model Parameters

Chapter 4 mentioned the design choices for each of the four classification models and two clustering models employed in this thesis. In this appendix, I provide and explain some of the diagnostics that led to those decisions.

A.1 kNN Classifier

A kNN classifier assigns each point the class that corresponds to the class of the majority of the k points closest to it. To determine the value of k I fit several kNN classifiers, varying k between 1 and 30. Then I compared their training and validation accuracy scores. I also cross-validated each using five folds of the entire Malimg dataset (meaning both the training and validation data). The training and

validation accuracies as a function of k are plotted in Figure A.1.1 and the cross validated accuracy is plotted as a function of k in Figure A.1.2. Between $k = 5$ and $k = 15$, the cross-validated accuracy is highest. Any values in that range would be a reasonable choice for k. I chose to proceed with $k = 7$ as this is where the validation accuracy reached a local maximum and the rate at which the training accuracy decreased began to stabilize. The training accuracy will always be highest when k is small because the model assigns each point the class of the point immediately adjacent to it in the feature space. But this is computationally expensive and prone to overfitting, which is why I look for the point at which the training accuracy reached a lower, more stable value.

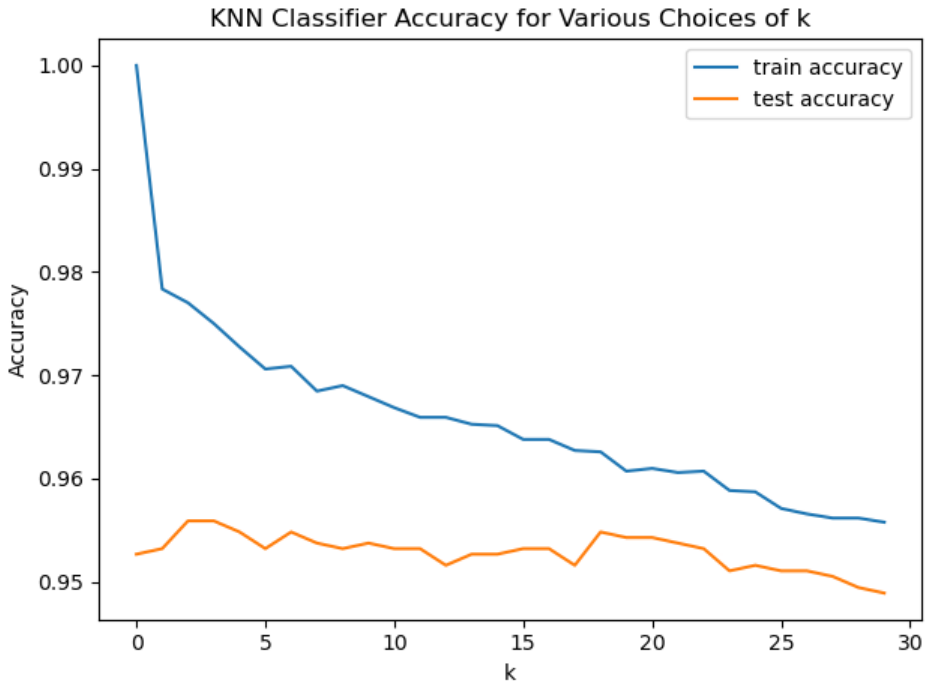

Figure A.1.1: Training and Validation Accuracy of the kNN Classifiers for $k \in [1, 30]$.

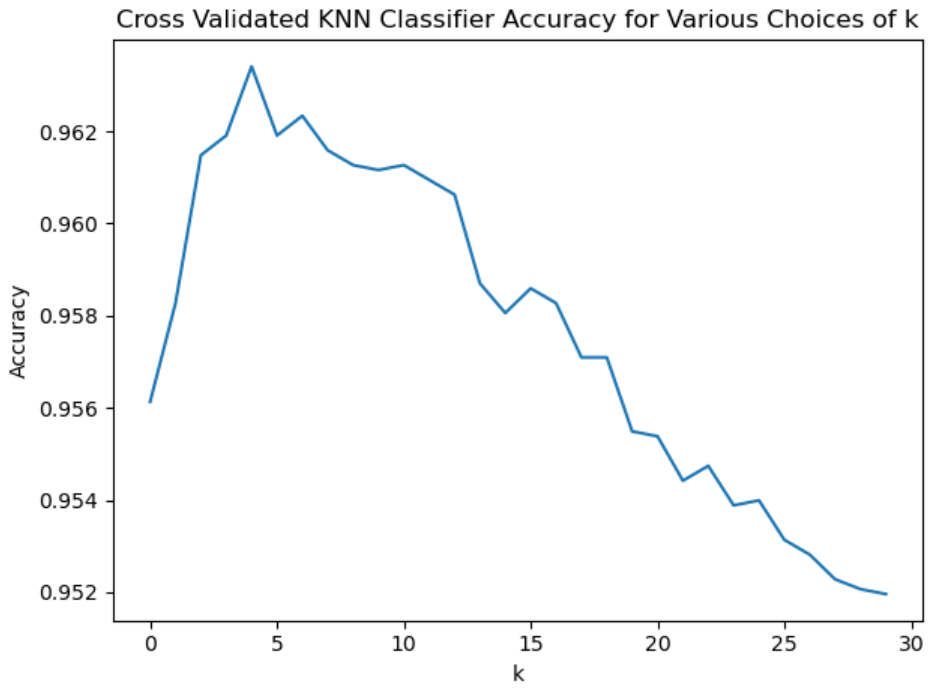

Figure A.1.2: Cross-Validated Accuracy Scores of the kNN Classifiers for $k \in [1, 30]$.

A.2 Random Forest Classifier

To decide on a number of trees in the random forest model, and the maximum depth of each tree, I fit 340 random forest models with trees between 3 and 19, and depths between 10 and 29. The choice to start with three trees was based on the fact that, after principal component analysis, there are only three features in the data for the trees to use in determining splits. It makes sense to have more than three trees in the random forest so that different combinations and orders of splits can be tested at different levels of each tree, but it does not make sense to use many more than 19 trees. In fact, I found that the training, validation, and cross-validated accuracies stopped improving well before 19 trees were included in the model. First, the

training accuracy as a function of both the number of trees and depth of each tree is shown in Figure A.2.1. The validation accuracy and cross-validated accuracy as a function of both parameters are included in Figures A.2.2 and A.2.3.

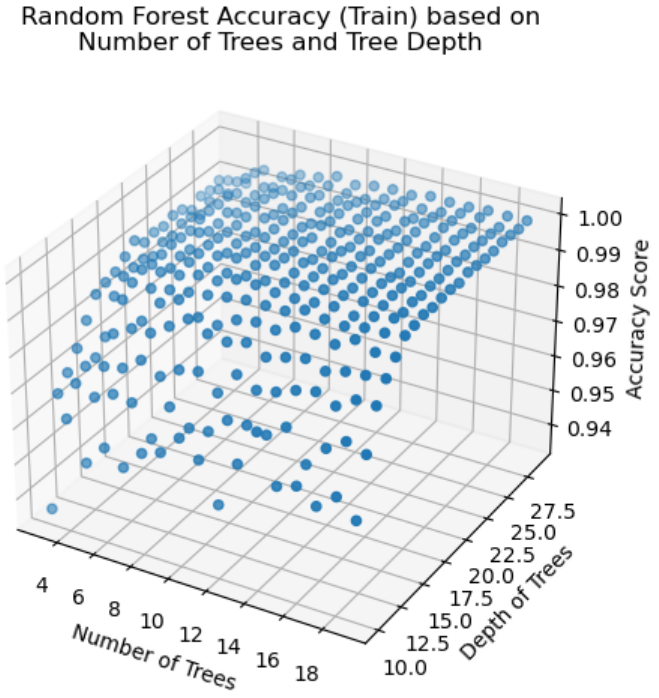

Figure A.2.1: Training Accuracy of Random Forest Classifier as Number of Trees Vary from 3-19 and Depths of Trees vary from 10-29.

It appears that accuracy reached a maximum when the number of trees was around 10. To check the result, I plotted the training accuracy as a function of the number of trees only in Figure A.2.4. Based on this plot, 10 would be a reasonable number of trees to include in the forest as that is where the accuracy stops increasing rapidly. Including more than 10 trees would risk an overfit model. To

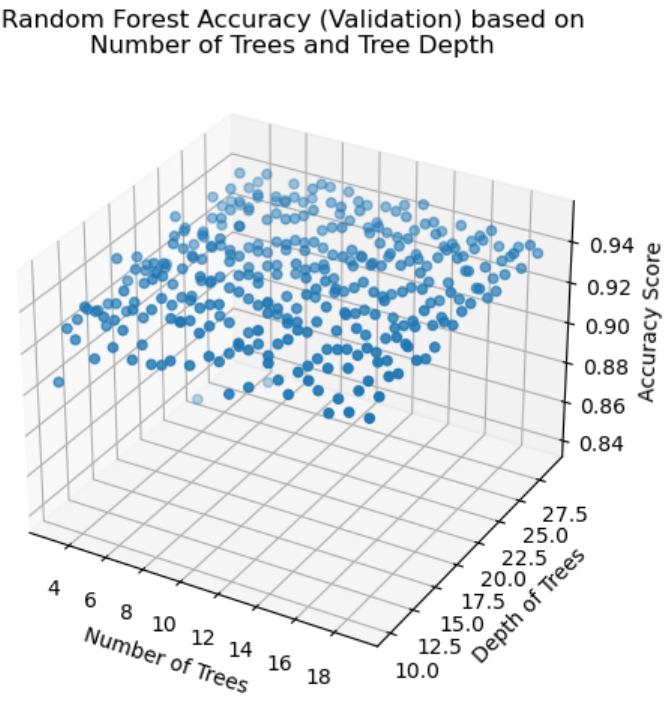

Figure A.2.2: Validation Accuracy of Random Forest Classifier as Number of Trees Vary from 3-19 and Depths of Trees vary from 10-29.

further justify the choice, I plot the validation and cross-validated accuracies of the models as a function of the number of trees in Figures A.2.5 and A.2.6 to show that they too stabilize at an optimal value when the number of trees in the random forest is 10.

4.1.1

To determine the depths of the trees, I plotted the training, validation, and cross-validated accuracy scores as a function of the depths of the trees only. These are included in Figures A.2.7, A.2.8,

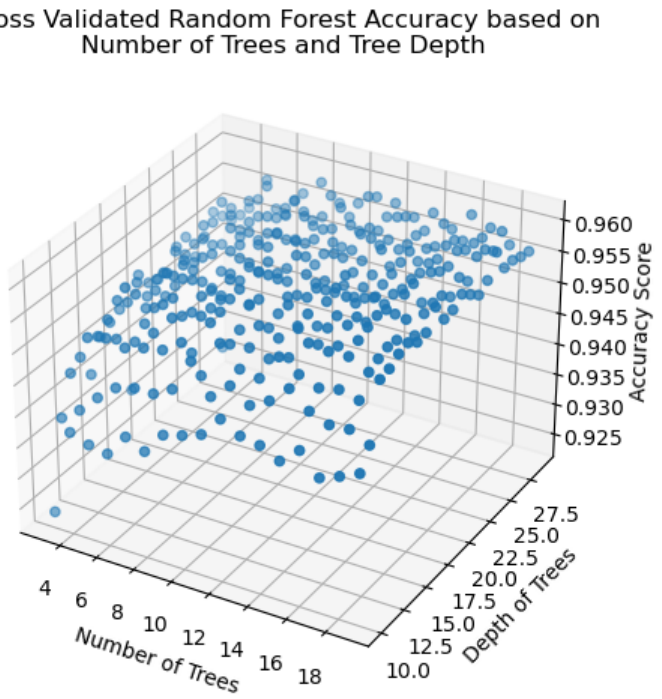

Figure A.2.3: Cross-Validated Accuracy of Random Forest Classifier as Number of Trees Vary from 3-19 and Depths of Trees vary from 10-29.

and A.2.9, respectively.

A.3 Support Vector Machine

In Chapter 4 I justified the choice of using a linear kernel and "one-versus-one" comparison in the support vector machine. I also tuned the regularization parameter, C, by fitting five SVM models with C values of 10^{-2}, 10^{-1}, 1 (Sklearn's default), 10, and 100. I found that $C = \frac{1}{10}$ yielded optimal models in terms of training, validation, and cross-validated accuracy (plotted in Figures A.3.1,

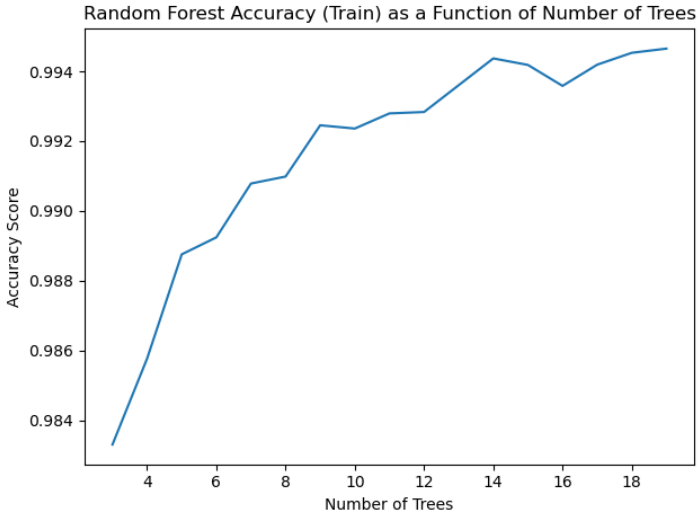

Figure A.2.4: Training Accuracy of Random Forest Classifier as Number of Trees Vary from 3-19.

A.3.2, and A.3.3).

The regularization parameter in a support vector machine controls overfitting. When C is large, the model penalizes misclassifications and demands more precise decision boundaries, which can lead to overfitting. When C is small, it allows for wider, more general classes. The tradeoff is that these classes risk encompassing some samples that do not actually belong within them.

A.4 *K*Means Clustering

The number of clusters is the defining parameter of a *K*Means cluster model, so deciding on a number in a setting in which the true number of classes was unknown was difficult. I fit a series of *K*Means cluster models, varying the value of K between 10 and 19. I found that silhouette scores were highest when K was on the lower end of

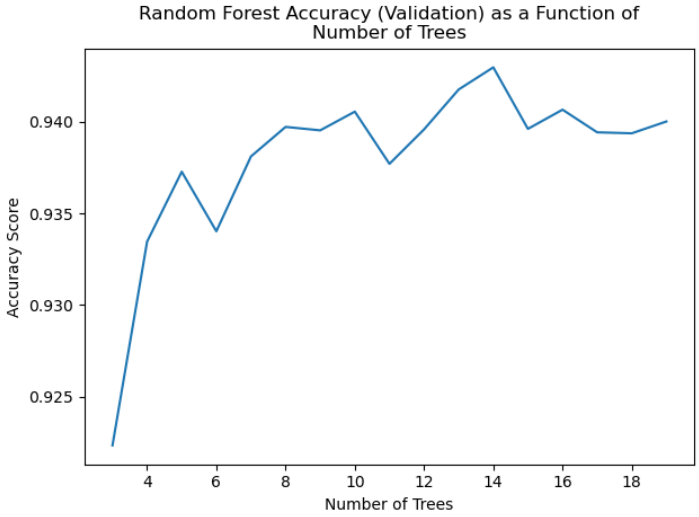

Figure A.2.5: Validation Accuracy of Random Forest Classifier as Number of Trees Vary from 3-19.

this spectrum. The silhouette scores decreased slowly for every increase in K clusters, so I did not worry about my range of 10 through 19 being orders of magnitude away from the true number of clusters in the data. The average silhouette score for a model with 10 clusters was slightly higher than that for a model with 11 clusters. But I also plotted the silhouette scores, by cluster, so that I could visualize which samples had the best in-cluster similarity (silhouette scores near 1.0) and which ones had the worst (negative silhouette scores). The plots are in Figure A.4.1. Overall, as the number of clusters increased, the number of clusters with negative silhouette scores increased, which is adverse. Based on the silhouette plots, I could tell that the model with 11 clusters had fewer clusters with negative silhouette scores, and more clusters with silhouette scores that approached 1, or fell above the mean. Some of the individual silhouette scores were of greater magnitude than individual silhouette scores in the model with 10 clusters. But I decided to use the model

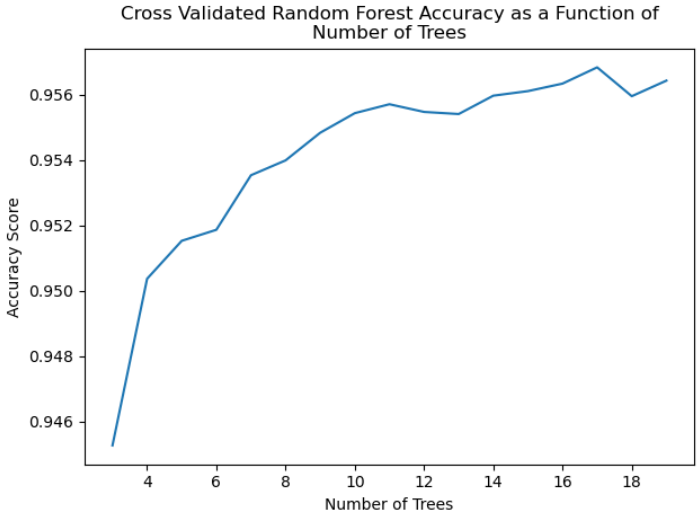

Figure A.2.6: Cross-Validated Accuracy of Random Forest Classifier as Number of Trees Vary from 3-19.

with more clusters well-defined, as opposed to fewer that were slightly more specific to a few samples.

A.5 Agglomerative Clustering

In fitting the agglomerative cluster model to the test dataset, I used a distance threshold. I tried a candidate distances between 290 and 340, in increments of 10, and plotted the silhouette scores of their models in order to choose an appropriate distance. After fitting each candidate model, I was able to see how many clusters its distance threshold yielded by using the `.n_clusters_` attribute. The silhouette plots are included in Figure A.5.1 and validate that the last distance threshold (340) yielded the purest clusters. There were, as in the case of KMeans clustering, 11 clusters total.

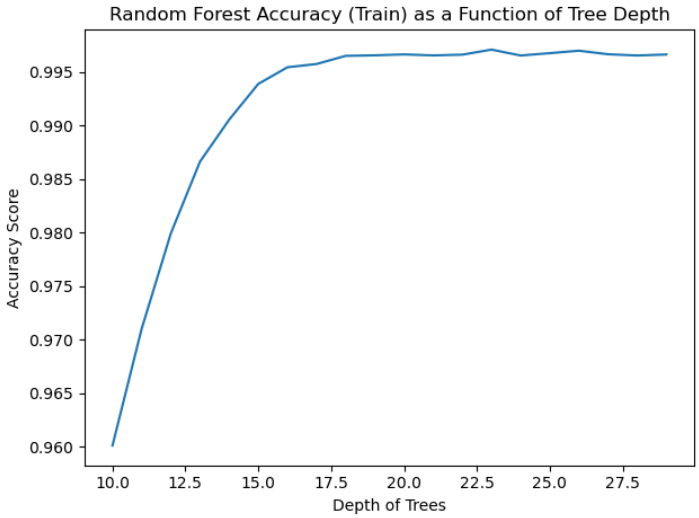

Figure A.2.7: Training Accuracy of Random Forest Classifier as Tree Depth Varies from 10 to 29.

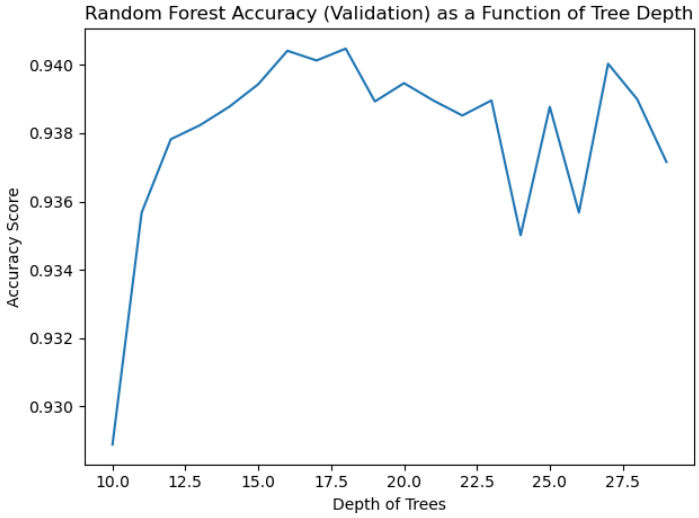

Figure A.2.8: Validation Accuracy of Random Forest Classifier as Tree Depth Varies from 10 to 29.

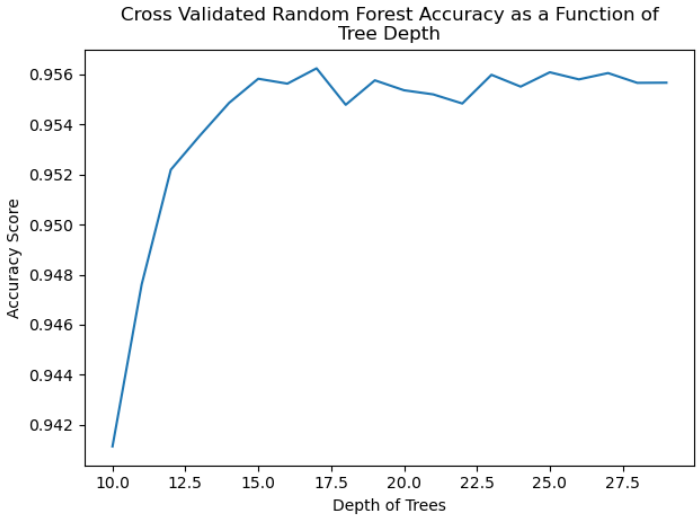

Figure A.2.9: Cross-Validated Accuracy of Random Forest Classifier as Tree Depth Varies from 10 to 29.

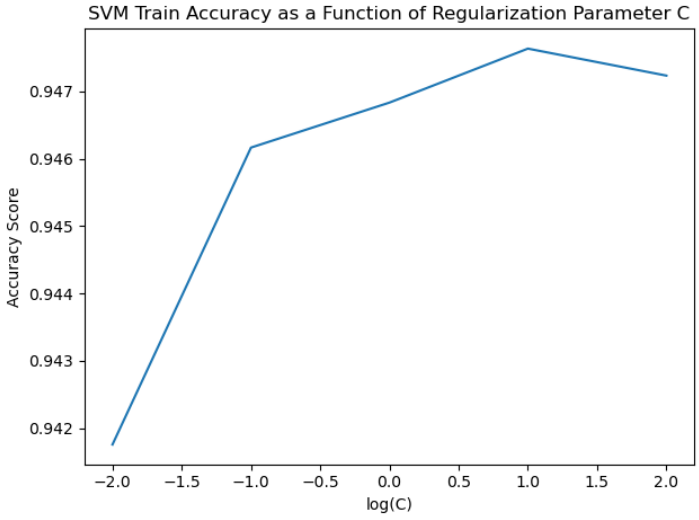

Figure A.3.1: Training Accuracy of Support Vector Machine as C Varies from 10^{-2} to 10^{2} Exponentially.

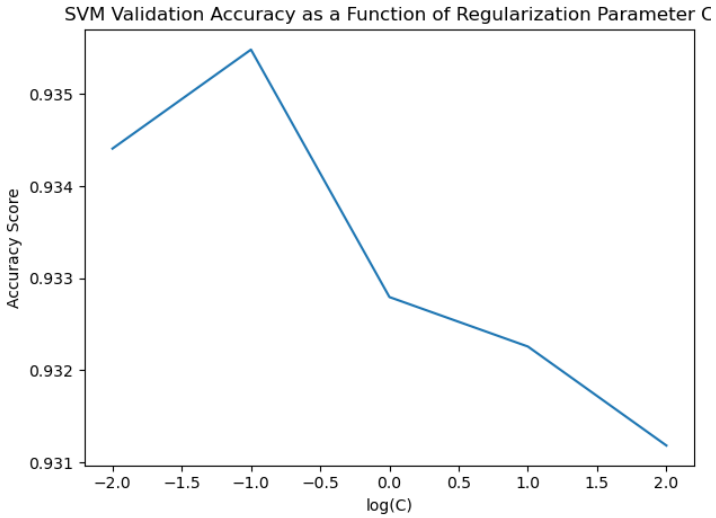

Figure A.3.2: Validation Accuracy of Support Vector Machine as C Varies from 10^{-2} to 10^{2} Exponentially.

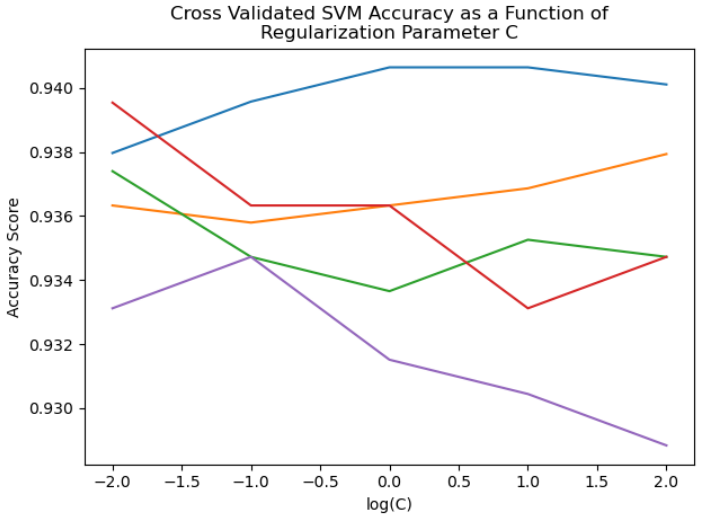

Figure A.3.3: Cross-Validated Accuracy of Support Vector Machine as C Varies from 10^{-2} to 10^{2} Exponentially.

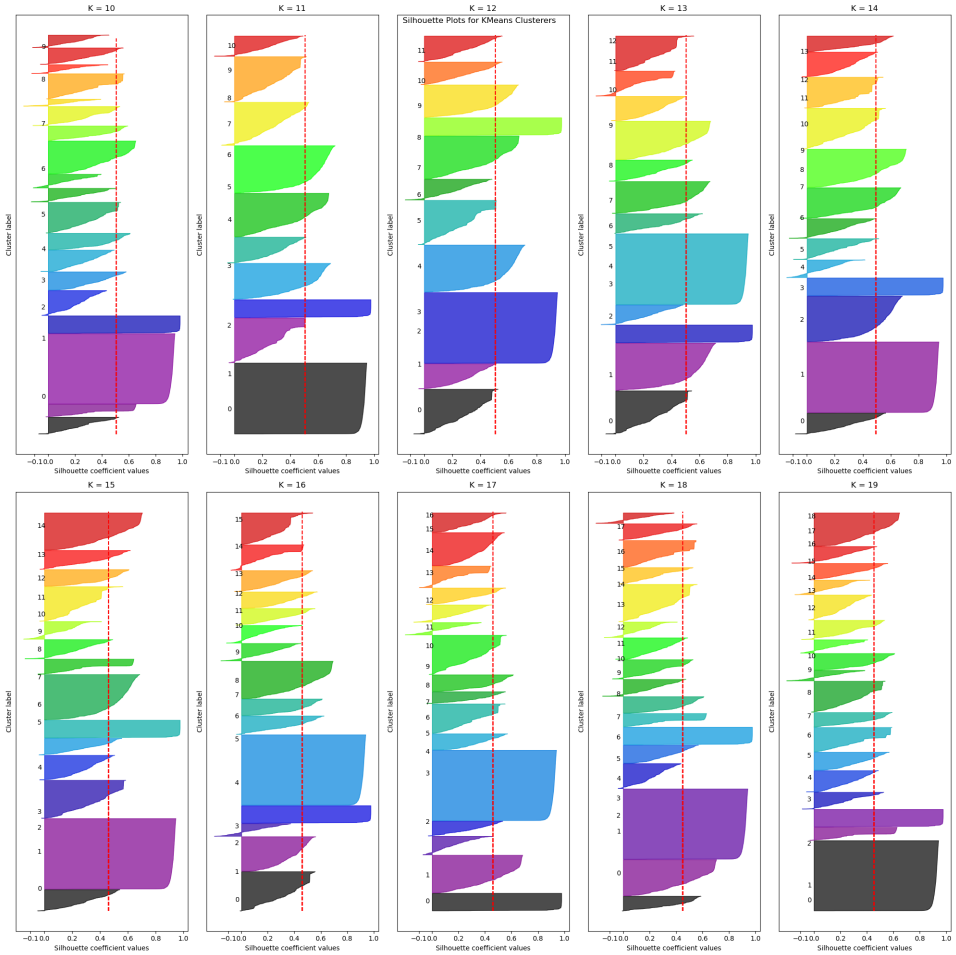

Figure A.4.1: Silhouette Scores by Cluster for KMeans Clusterers with $K = 10$ through $K = 19$.

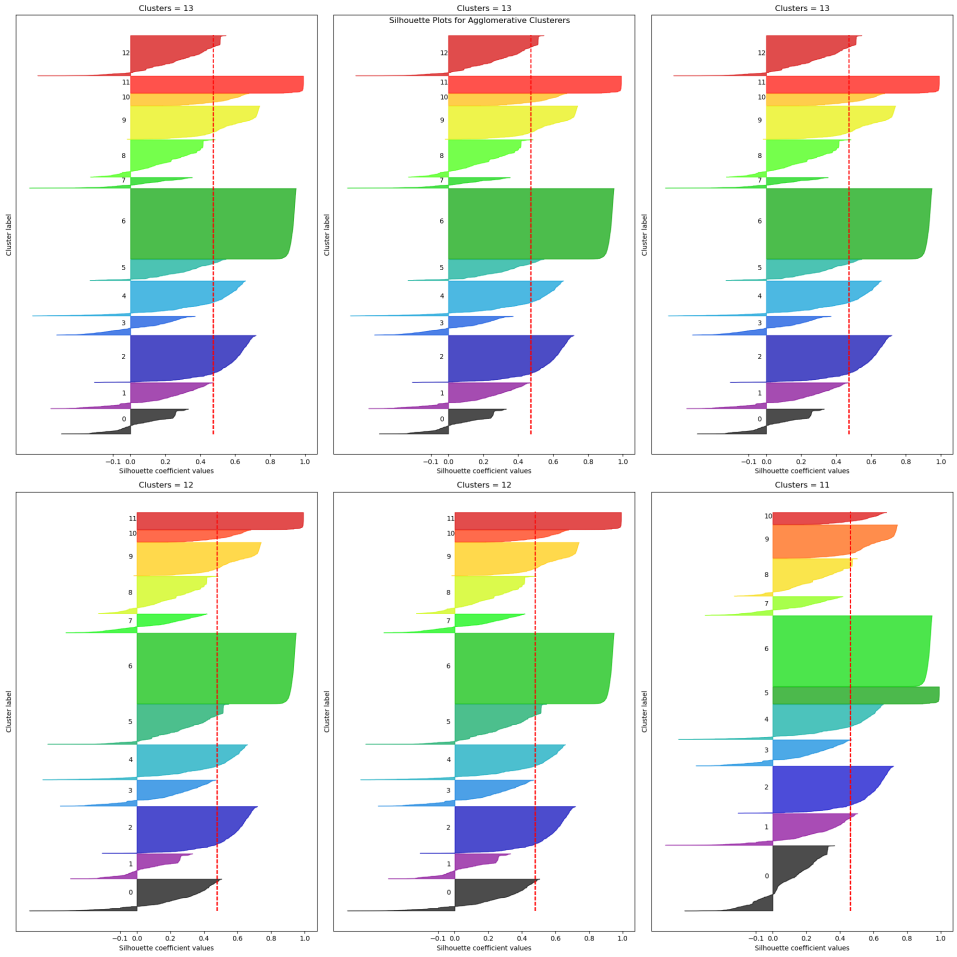

Figure A.5.1: Silhouette Scores by Cluster for Agglomerative Clusterers with Distance Thresholds between 290 and 340.

B

Confusion Matrices of Four Image Classifiers

The training and validation accuracy, precision, recall, and F1 scores for each model are included in Chapter 5. This appendix provides extended diagnostics, particularly the confusion matrices, for each individual classifier.

B.1 kNN Classifier

The confusion matrices for the malware families predicted using kNN Classification on the training and validation sets are included in Figures B.1.1 and B.1.2. Both matrices are similar in their classifications and they are quasi-diagonal, indicating accurate

predictions. The classes with which the model seems to have the most difficulty assigning predictions are Swizzor.gen!E and Swizzor.gen!I: the model classified about two thirds of the samples correctly and assigned the other one third to different classes—most commonly each other's. This could be the result of Swizzor.gen!E and Swizzor.gen!I having some similar features since they are derivative of each other.[1]

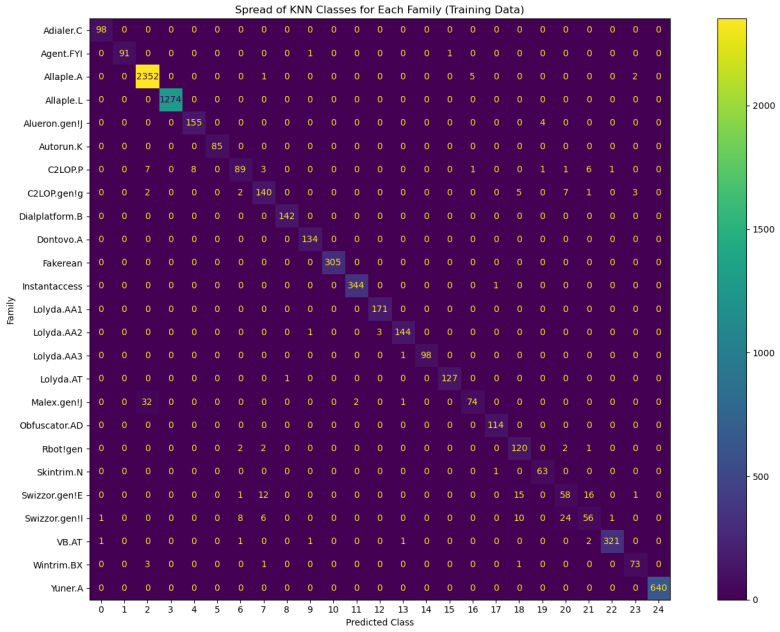

Figure B.1.1: Classes Assigned to Malimg Training Samples via a kNN Model.

[1]It is also possible that the preprocessing to resize images to 224 by 224 pixels removed some of the pixels that distinguished these classes from each other, particularly since they are derivative of each other and the changes between them are small.

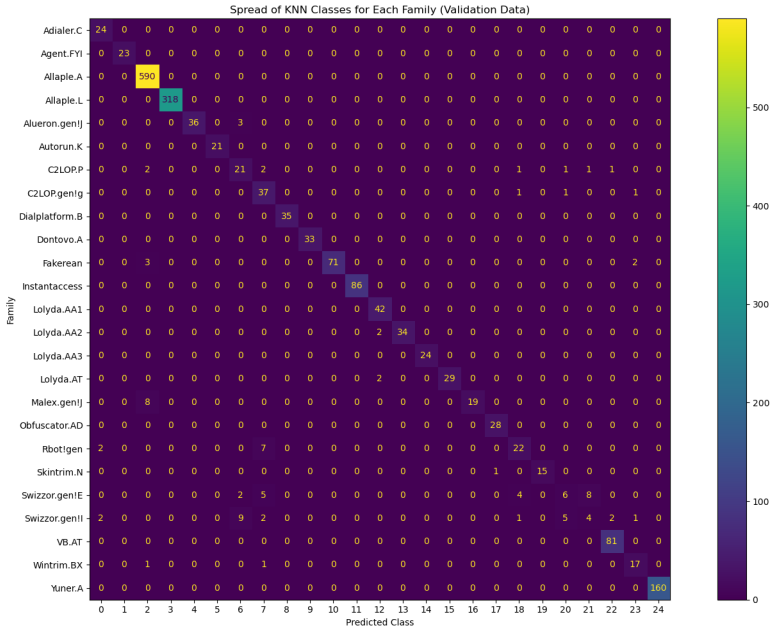

Figure B.1.2: Classes Assigned to Malimg Validation Samples via a kNN Model.

B.2 Random Forest Model

The confusion matrices for the malware families predicted using a random forest model on the training and validation sets are included in Figures B.2.1 and B.2.2. For most of the classes, the random forest assigned the same number of samples as the kNN classifier, however it places more samples in the correct classes (its confusion matrix has more elements along its diagonal than that of the kNN classifier), as evident by the higher training accuracy. Similar to the kNN classifier, the classes with which it has the most difficulty on the validation set are Swizzor.gen!E and Swizzor.gen!I.

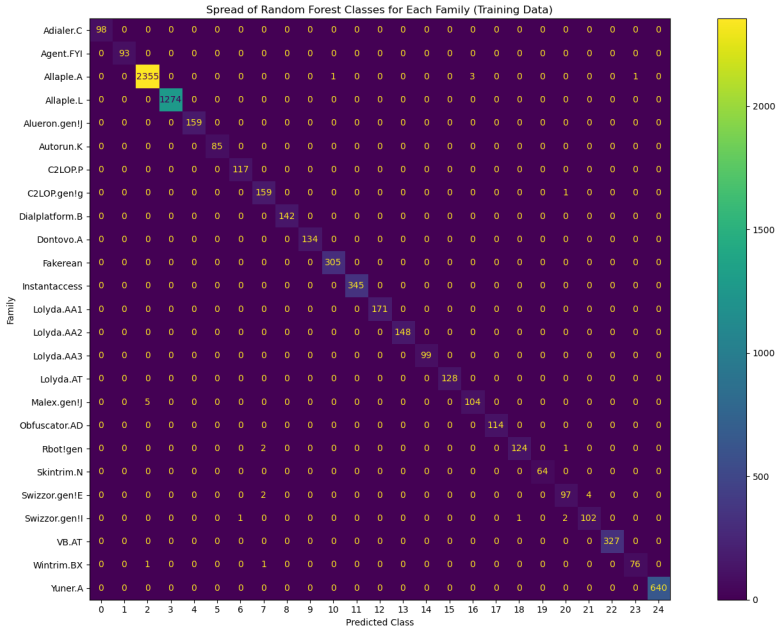

Figure B.2.1: Classes Assigned to Malimg Training Samples via Random Forest Classification.

B.3 Support Vector Machine

The confusion matrices for the malware families predicted using kNN classification on the training and validation sets are included in Figures B.3.1 and B.3.2. The SVM classifier suffered from slightly lower training and validation accuracy, precision, recall, and F1 scores than the other models included in this analysis. On the confusion matrices, this is visually apparent by its misclassification of samples in the Allaple.A family and its confusion between the Swizzor.gen!E and Swizzor.gen!I families (even though this confusion is also common between the random forest and kNN classifiers). The SVM classifier misclassified samples in the Allaple.A family on both its training and validation data, failing to identify over 286 samples in the training data and 48 samples in the validation data. the most common family

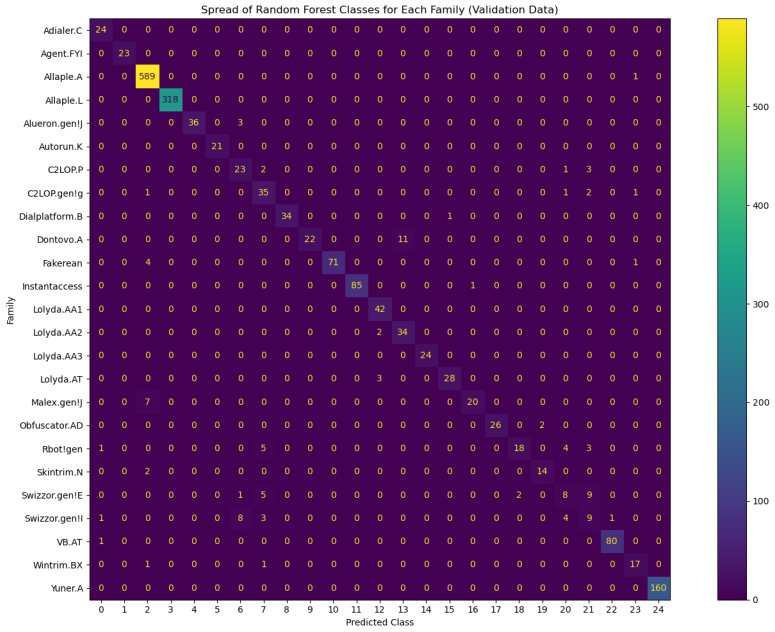

Figure B.2.2: Classes Assigned to Malimg Validation Samples via Random Forest Classification.

with which Allaple.A was confused was Malex.gen!J. In turn, 18 of the 119 Malex.gen!J samples in the training data and 6 of the 27 Malex.gen!J were misclassified as belonging to Allaple.A by the SVM classifier.

B.4 XGBoost Model

The confusion matrices for the malware families predicted using the XGBoost classifier on the training and validation sets are included in Figures B.4.1 and B.4.2. The XGBoost model achieved nearly perfect training accuracy, precision, recall, and F1 scores. Its validation accuracy suffered as a result of its confusing the Yuner.A, Autorun.K, C2LOP.gen!g, and Swizzor.gen!I classes. On the validation set, the

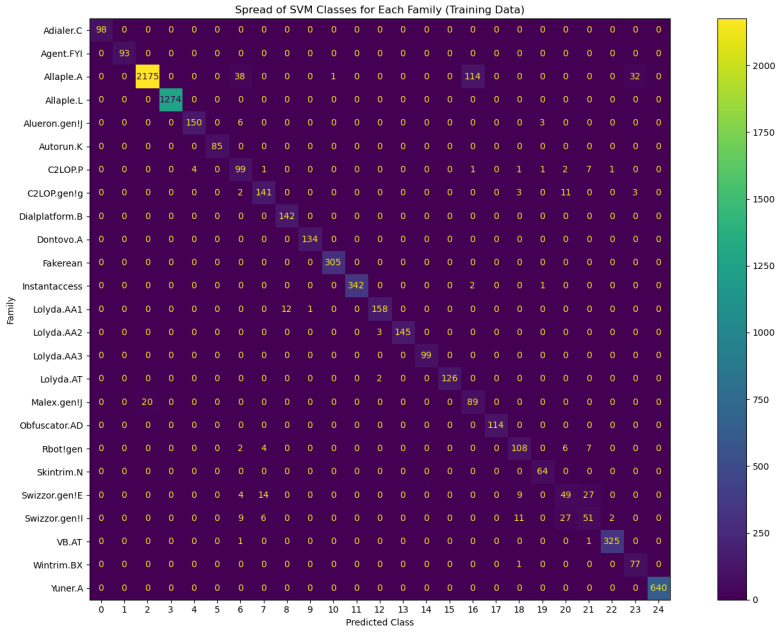

Figure B.3.1: Classes Assigned to Malimg Training Samples via SVM Classification.

XGBoost model classified all of the samples belonging to the Yuner.A family as belonging to the C2LOP.gen!g family. In turn, it classified all of the samples belonging to the Autorun.K family as belonging to Swizzor.gen!I. The model also showed the same slight discrepancies in differentiating the Swizzor.gen!E and Swizzor.gen!I which the other three classifiers showed, but these may be explained by the fact that binaries in these two families are semantically similar.

The misclassification of the Yuner.A family by the XGBoost model could be a product of randomness. In an initial run with no random seed, the model predicted all Yuner.A samples correctly. Adding a seed influenced the prediction, and the mistake did not disappear even after removing the seed and restoring all other conditions (e.g., restarting the kernel and using original model parameters). It could

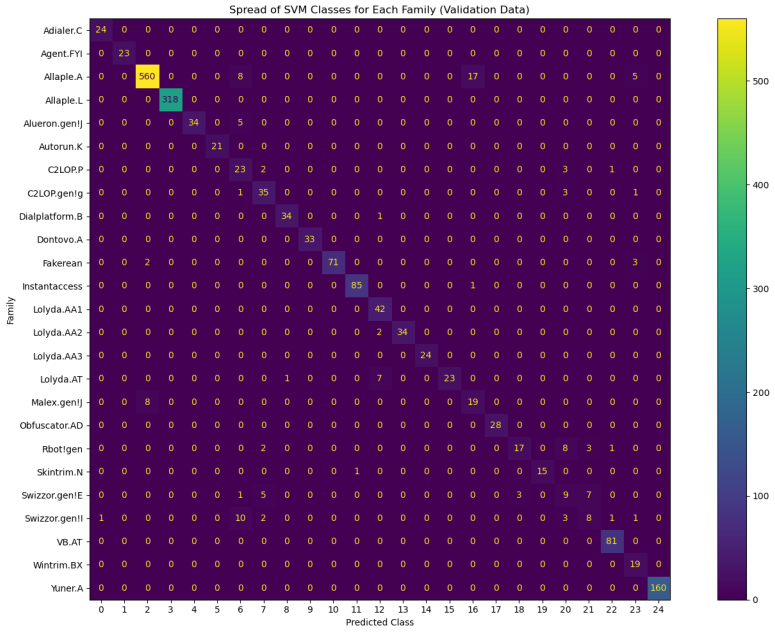

Figure B.3.2: Classes Assigned to Malimg Validation Samples via SVM Classification.

also be explained by the fact that C2LOP.gen!g is one of the smaller families in the dataset, containing fewer malware samples than the other families (40). This could mean that the model does not have enough context to learn how to differentiate its features from those of the Yuner.A class. However, the Yuner.A family has a substantial number of samples in the dataset, so its misclassification is more surprising.

I computed the training and validation accuracy, precision, recall, and F1 scores for a version of the XGBoost model trained on all data excluding those samples of the Yuner.A family, to verify that they improved. The results are included in Table B.4.1.

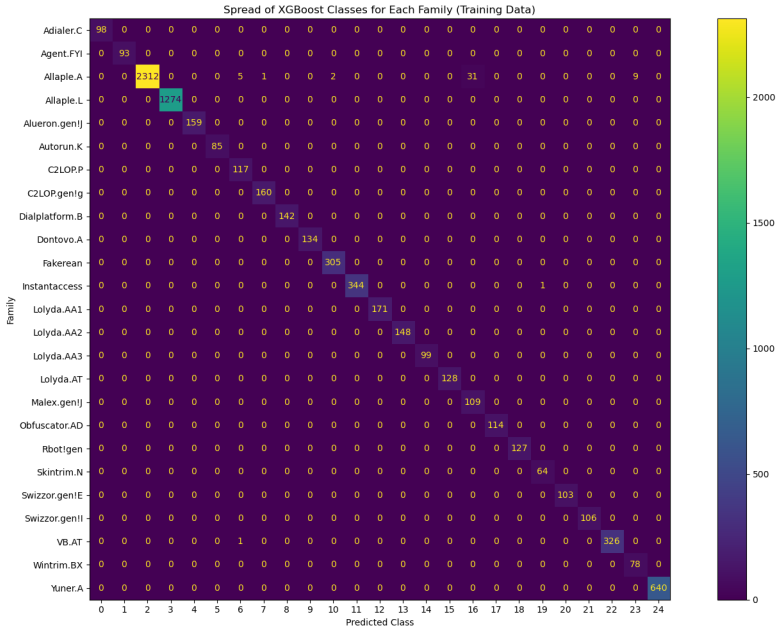

Figure B.4.1: Classes Assigned to Malimg Training Samples via XGBoost.

Metric	Training	Validation
Accuracy Score	1.0	0.9706
Precision	1.0	0.9304
Recall	1.0	0.9344
F1 Score	1.0	0.9311

Table B.4.1: Training and Validation Accuracy, Precision, Recall, and F1 Scores for a Version of the XGBoost Model Excluding the Yuner.A Family.

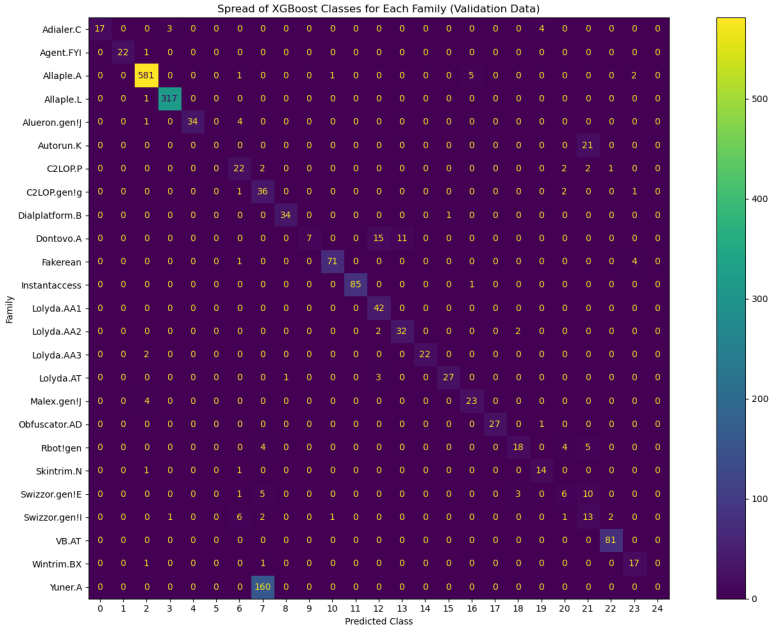

Figure B.4.2: Classes Assigned to Malimg Validation Samples via XGBoost.

C

Results of Individual Image Classifiers

In Chapter 5 I determined the malware family of each sample in the APT dataset by taking the mode prediction of four image classifiers, and interpreted the result. In Appendix B I expanded on the diagnostic results of each model. In this appendix I include the extended results—how each of the four classifiers categorized each sample in the dataset—and compare the model predictions to each other.

C.1 kNN Classifier

Figure C.1.1 summarizes the distribution of malware families used by each threat actor, according to the kNN classifier only. The kNN classifier shows a propensity to predict the Allaple.A family of malware, especially within each malware strain (as opposed to APT group). It notably does *not* assign the Instantaccess family to many samples, which was one of the most common predictions made by the random forest, support vector machine, and XGBoost models.

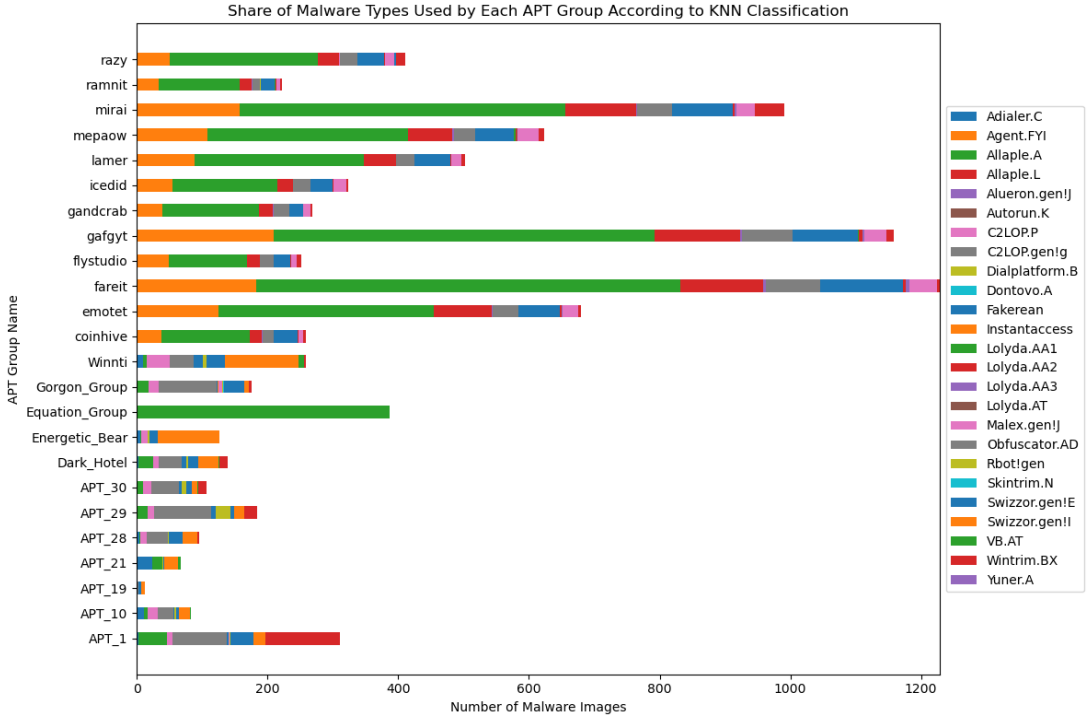

Figure C.1.1: Malware Families Used by Each Actor, according to kNN Classification.

Figure C.1.2 shows the fluctuations in malware families used over time, according to the kNN classifier only.

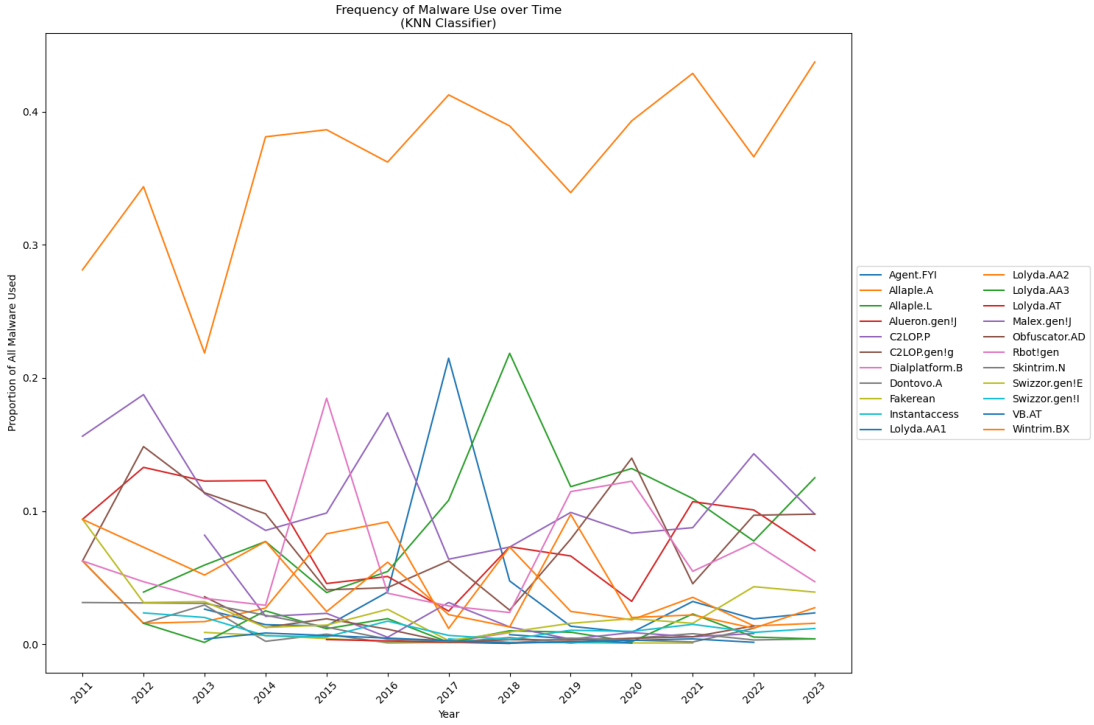

Figure C.1.2: Malware Families used over time, according to kNN Classification.

C.2 Random Forest Model

Figure C.2.1 summarizes the distribution of malware families used by each threat actor, according to the random forest model only.

Figure C.2.2 shows the fluctuations in malware families used over time, according to the random forest model only.

C.3 Support Vector Machine

Figure C.3.1 summarizes the distribution of malware families used by each threat actor, according to the support vector machine only.

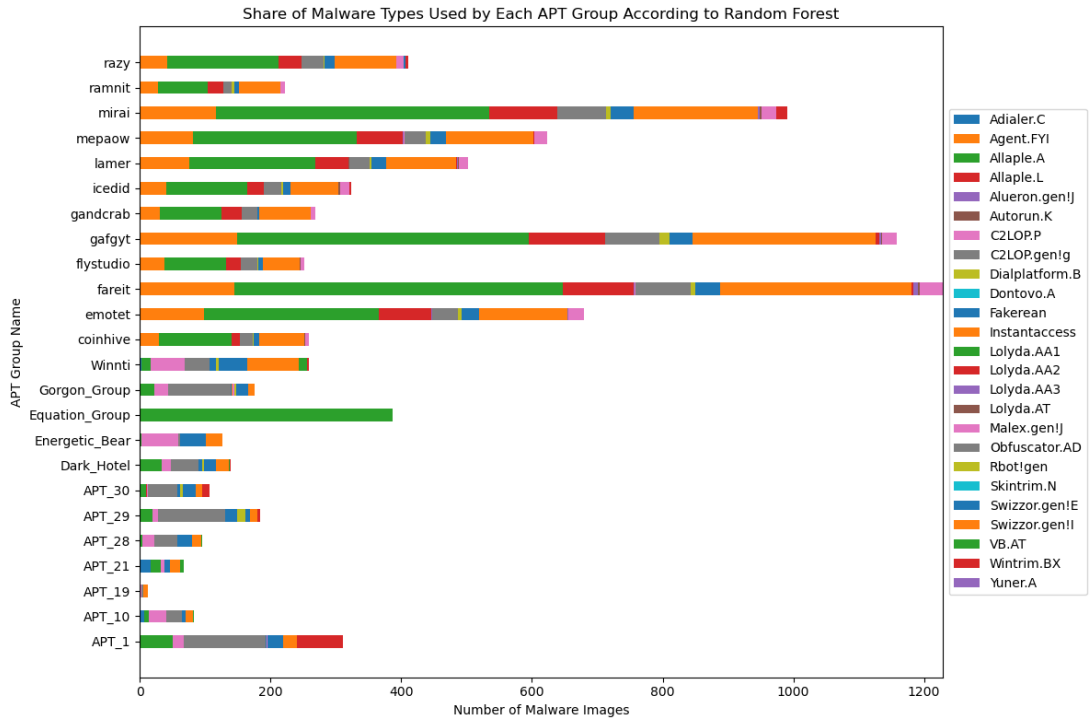

Figure C.2.1: Malware Families Used by Each Actor, according to Random Forest Classification.

The frequency with which the support vector machine predicted family "Allaple.A" is lower than that of the other models. Overall, its predictions appear more varied than those of the other models: more families are predicted, each is predicted a relatively few number of times.

Figure C.3.2 shows the fluctuations in malware families used over time, according to the SVM classifier only.

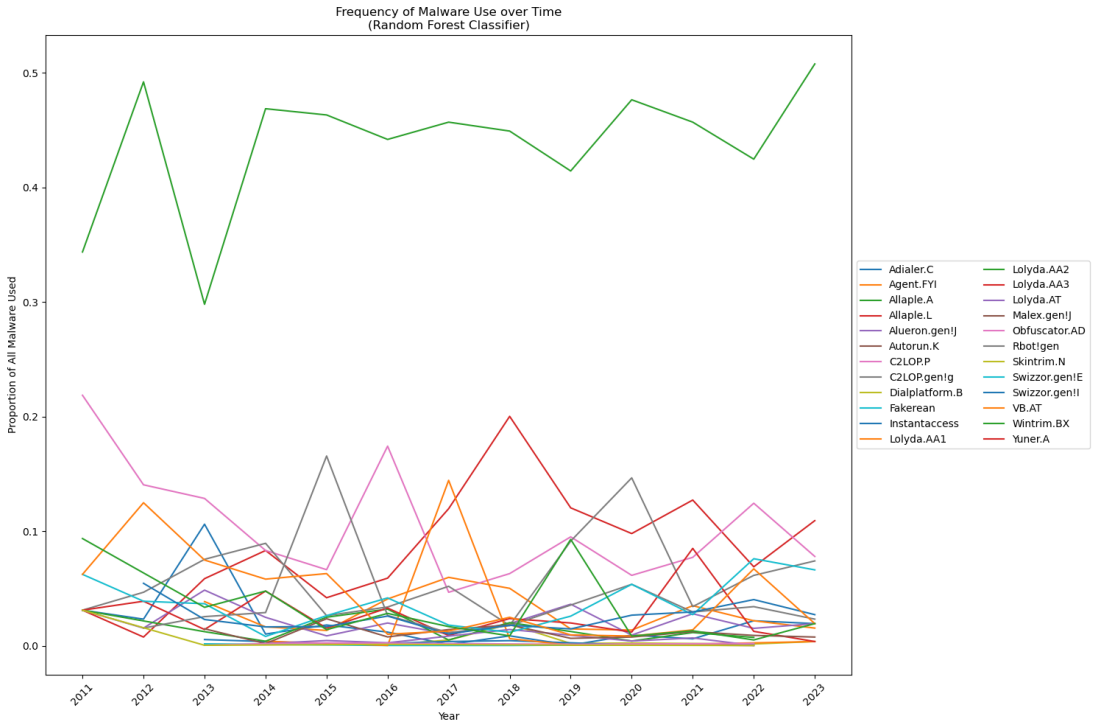

Figure C.2.2: Malware Families used over time, according to Random Forest Classification.

C.4 XGBoost Model

Figure C.4.1 summarizes the distribution of malware families used by each threat actor, according to the XGBoost model only. The XGBoost model's predictions are closest to that of the random forest model, which is unsurprising given that both methods are built on decision trees.

Figure C.4.2 shows the fluctuations in malware families used over time, according to the XGBoost classifier only.

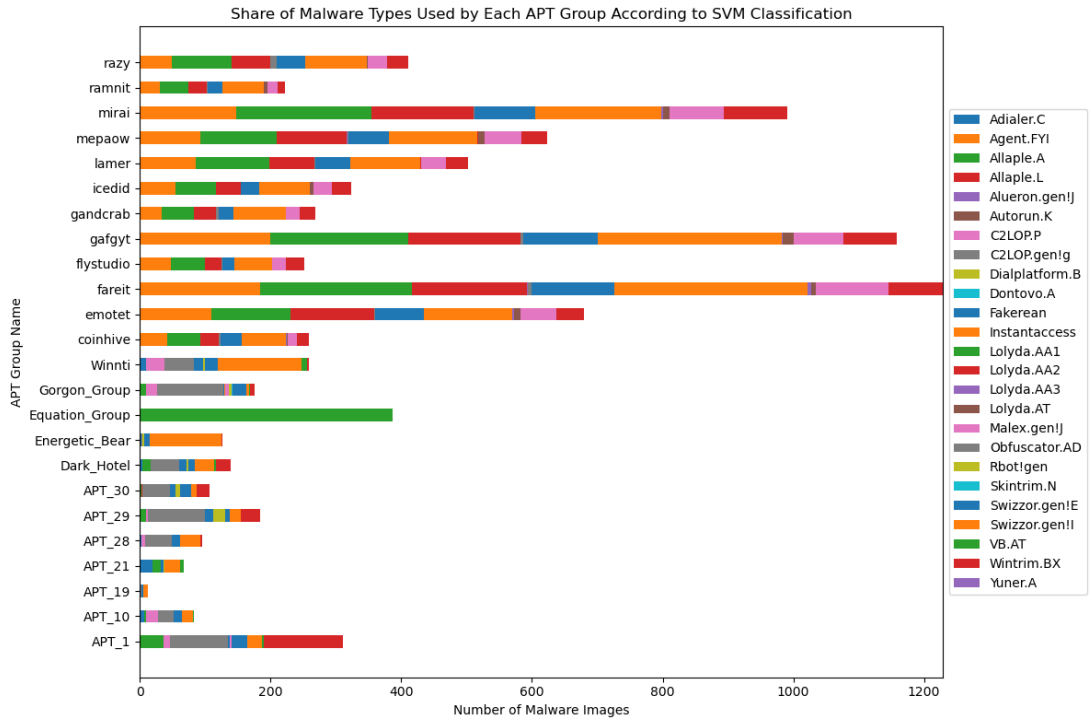

Figure C.3.1: Malware Families Used by Each Actor, According to Support Vector Machine Classification.

C.5 Model Comparison

As mentioned in Chapter 5 all four classifiers predicted the same malware family for 26.89% and three out of the four classifiers predicted the same malware family for 52.84% of the samples in the APT dataset. I will mention here that, when the models were fit using only three principal components of the Malimg dataset, these metrics improved to 34.07% and 81.72%, respectively, which suggests that even the use of four principal components may have resulted in overfitting to the training data, despite being justified by their cumulative variance explains. Comparing the bar charts and line charts in this appendix, it seems that the kNN model and random

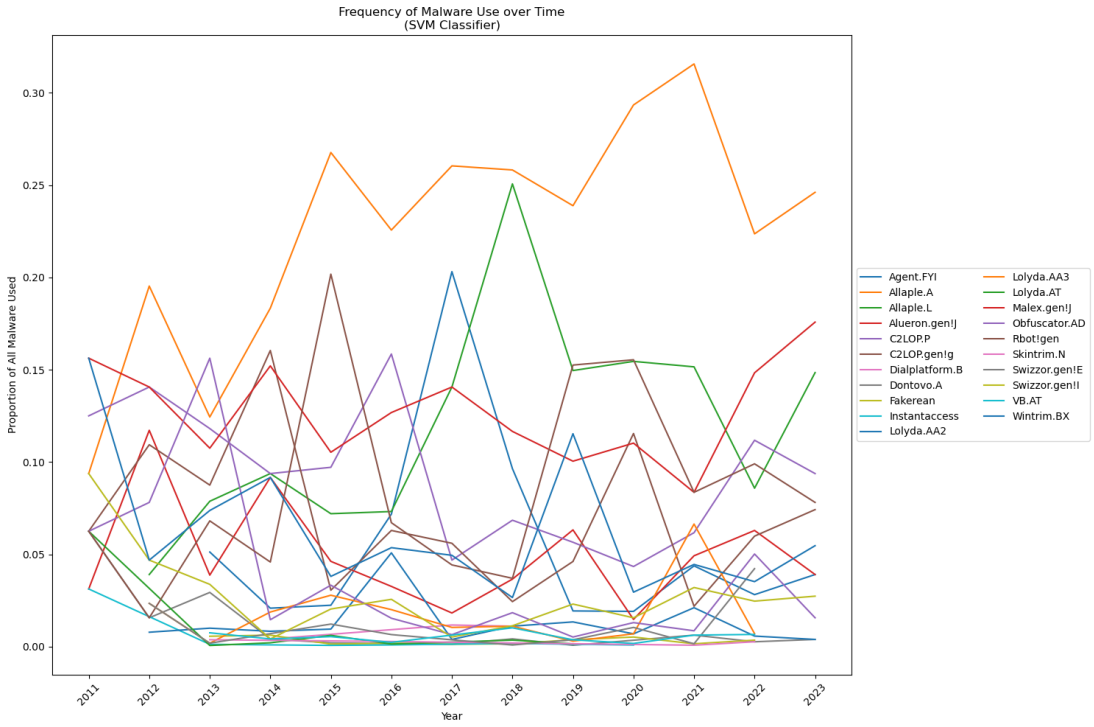

Figure C.3.2: Malware Families used over time, according to SVM Classification.

forest model were most similar in prediction and that the SVM had the most variability.

Cohen's Kappa Statistic is a metric for pairwise comparison between two classifiers on the same dataset. It is distinct from taking the empirical proportion of times that the two models predicted the same class in that it accounts for the fact that the two models may have yielded similar predictions purely based on chance. Cohen's Kappa is defined in Equation C.1 where p_a represents the observed agreement between the models (the proportion of times they predicted the same classes for the same samples) and p_e agreement of the models due to chance. p_e is defined in C.2 where cm^1 is the number of samples that rater 1 assigned to class 1 and rm^1 is the

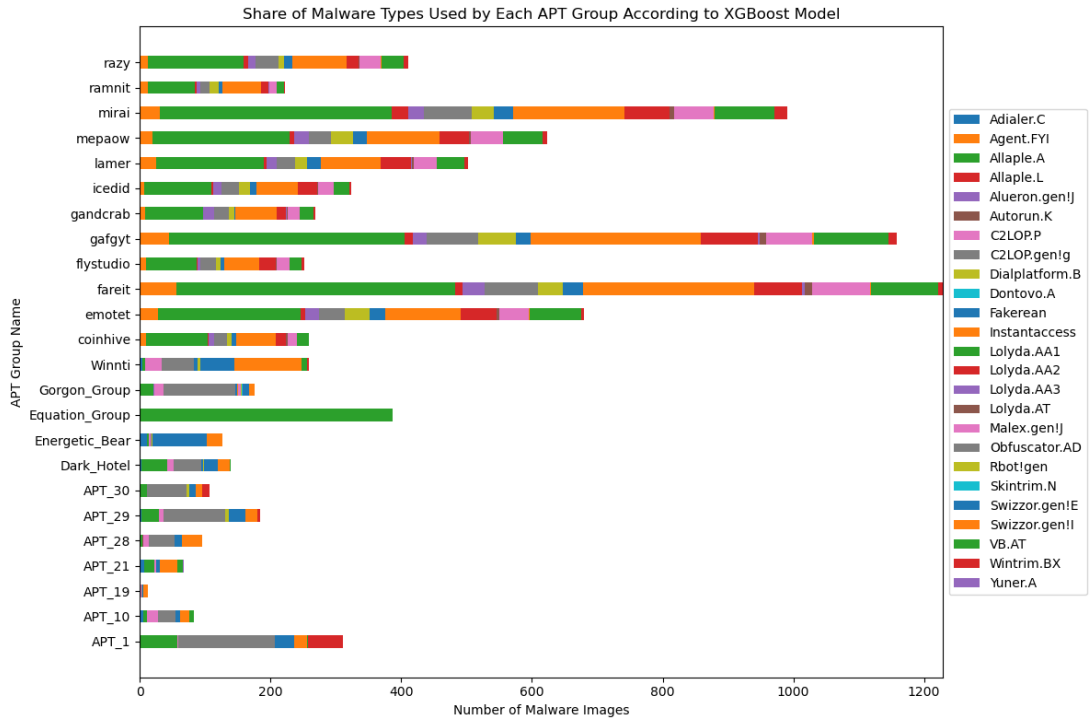

Figure C.4.1: Malware Families Used by Each Actor, according to XGBoost Classification.

number of samples that rater 2 assigned to class 1, and vice versa for cm^2 and rm^2.

$$\kappa = \frac{p_a - p_e}{1 - p_e} \tag{C.1}$$

$$p_e = \frac{cm^1 \cdot rm^1 + cm^2 \cdot rm^2}{n^2} \tag{C.2}$$

The values of Cohen's Kappa Statistic for each of the four classifiers used in this methodology are provided in table C.5.1. It seems that the random forest model and support vector machine are actually the two most similar models in terms of prediction.

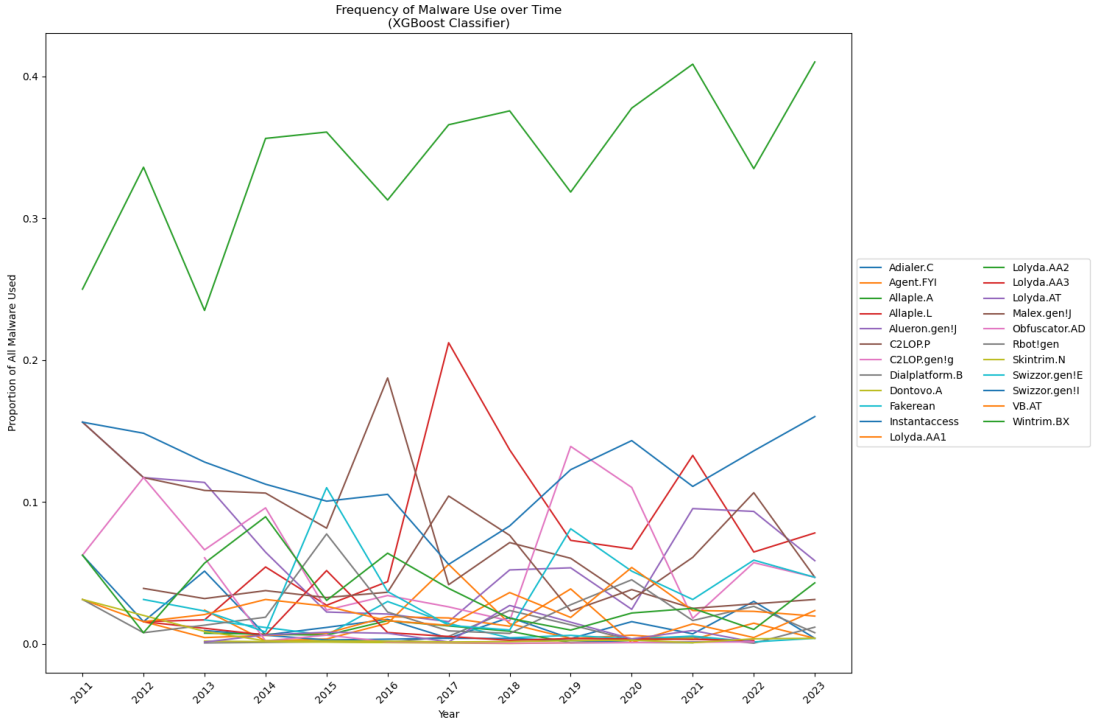

Figure C.4.2: Malware Families used over time, according to XGBoost Classification.

C.6 Takeaways

In the bars represented by APTs, there seems to be more consensus than in the bars represented by malware strains. This makes sense because the number of samples attributed to malware strains tends to be higher than that attributed to APTs. This is also consistent with the fact that APTs represent cyber actors narrowly focused on specific missions or targets, whereas malware strains encompass the wider landscape of cybercriminals.

A testament to the overall reliability of the models is that each model independently predicted the Equation Group to use VB.AT malware in 100% of its samples and that these samples were all

Models	κ
kNN & Random Forest	0.3839
kNN & SVM	0.4877
kNN & XGBoost	0.3530
Random Forest & SVM	0.5307
Random Forest & XGBoost	0.3957
SVM & XGBoost	0.3084

Table C.5.1: Cohen's Kappa Statistic, κ, For Pairwise Comparison of Classification Models.

clustered together by the KMeans and agglomerative clustering models. This could also reflect a specific characteristic of VB.AT malware that makes it easily distinguishable from samples in all other families.

References

[1] Baker, K. (2023). The 12 Most Common Types of Malware. *CrowdStrike*. Available at: https://www.crowdstrike.com/en-us/cybersecurity-101/malware/types-of-malware/.

[2] Charan, P., Anand, P., Chunduri, H., and Shukla, S. (2023). A Framework for Advanced Persistent Threat Attribution using Zachman Ontology. *European Interdisciplinary Cybersecurity Conference (EICC 2023)*, pages 34–41.

[3] Chen, P., Desmet, L., and Huygens, C. A Study on Advanced Persistent Threats. *15th IFIP International Conference on Communications and Multimedia Security (CMS)*,, pages 63–72.

[4] CISA. Critical Infrastructure Sectors. Available at: https://www.cisa.gov/topics/critical-infrastructure-security-and-resilience/critical-infrastructure-sectors.

[5] Cisco. What is Malware? Available at: https://www.cisco.com/site/us/en/learn/topics/security/what-is-malware.html#:~:text=Malware%2C%20short%20for%20malicious%20software,spyware%2C%20adware%2C%20and%20ransomware.

[6] Corets, C. and Vapnik, V. (1995). Support Vector Networks. *Machine Learning*, 20(3):273–297.

[7] Council on Foreign Relations (2010). Stuxnet. Available at: https://www.cfr.org/cyber-operations/stuxnet.

[8] Deepa, K., Adithyakumar, K., and Vinod, P. (2022). Malware Image Classification using VGG16. *2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS)*, pages 1–6.

[9] Divakarla, U., Chandrasekaran, K., Harish, S., Kanal, P., and Shalini, C. (2024). Malware Classification Using XGBoost and Genetic Algorithm for Hyperparameter Tuning. *2024 8th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS)*, pages 1–6.

[10] Edie, Z. Z. and Jasim, A. D. (2021). Malware Detection System Based on Deep Learning Technique. *Iraqi Journal of Information and Communications Technology*, 1(1):33–44.

[11] Emotet (2023). Emotet Malware: The Enduring and Persistent Threat to the Health Sector. Technical report, Department of Health and Human Services, Office of Information Security.

[12] Fix, E. and Hodges, J. (1951). Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review*, 57(3):238–247.

[13] Fortinet (2023). Ransomware Statistics And Ransomware Trends. Available at: https://www.fortinet.com/resources/cyberglossary/ransomware-statistics#:~:text=These%20settlement%20amounts%20do%20not%20include%20the,breach%2C%20which%20IBM%20says%20averages%20$4.54%20million.

[14] Gibert, D., Mateu, C., Planes, J., and Ramon, V. (2019). Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques*, 15:15–28.

[15] Government Accountability Office (2022). Internet-Connected Technologies Can Improve Services, but Face Risks of Cyberattacks. Available at: https://www.gao.gov/blog/internet-connected-technologies-can-improve-services-face-risks-cyberattacks#~:text=Some%20examples%20of%20IoT%20in,also%20part%20of%20the%20IoT.

[16] Hughes, D. and Colarik, A. M. (2016). Predicting the Proliferation of Cyber Weapons into Small States. *Joint Force Quarterly 83*, pages 19–26.

[17] ISC2 (2024). ISC2 Cybersecurity Workforce Study, 2024. Technical report, International Information System Security Certification Consortium.

[18] Kaspersky Labs (2014). Unveiling "Careto" - The Masked APT. Technical report, Kaspersky Labs.

[19] Kaspersky Labs (2015). Equation Group: Questions and Answers. Technical report, Kaspersky Labs.

[20] Kaspersky Labs (2020). The number of new malicious files detected every day increases by 5.2% to 360,000 in 2020. Available at: https://www.kaspersky.com/about/press-releases/the-number-of-new-malicious-files-detected-every-day-increases-by-52-to-3600

[21] Kaspersky Labs (2024). Careto APT resurfaced after 10 years with new malicious frameworks. Available at: https://www.kaspersky.com/about/press-releases/careto-apt-resurfaced-after-10-years-with-new-malicious-frameworks.

[22] Kiger, J., Ho, S., and Heydari, V. (2022). Malware Binary Image Classification Using Convolutional Neural Networks. *International Conference on Cyber Warfare and Security*, 17(1):469–.

[23] Kirsch, S. and Saunders, B. (2023). Addressing Russian and Chinese Cyber Threats.

[24] Lenaerts-Bergmans, B. (2023). Computer Worms Explained. *CrowdStrike Cybersecurity 101: The Fundamentals of Cybersecurity*. Available at: https://www.crowdstrike.com/en-us/cybersecurity-101/malware/computer-worm/.

[25] Madan, S., Sofat, S., and Bansal, D. (2022). Tools and Techniques for Collection and Analysis of Internet-of-Things malware: A systematic state-of-art review. *Journal of King Saud University - Computer and Information Sciences*, 10B(34):9867–9888.

[26] Malware Encyclopedia. Microsoft security intelligence. Available at: https://www.microsoft.com/en-us/wdsi/threats.

[27] Mandiant (2013). APT1: Exposing One of China's Cyber Espionage Units. Technical report, Mandiant.

[28] Maschmeyer, L. (2024). Cyber Conflict and Subversion in the Russia-Ukraine War. Available at: https://www.lawfaremedia.org/article/cyber-conflict-in-the-russia-ukraine-war.

[29] MITRE. MITRE ATT&CK Matrix. Available at: https://www.cybercom.mil/About/History/#:~:text=The%20Joint%20Chiefs%20of%20Staff,actors%20in%20this%20new%20domain.

[30] Mitsuhashi, R. and Shinagawa, T. (2020). High-Accuracy Malware Classification with a Malware-Optimized Deep Learning Model.

[31] Narayanan, B., Djaneye-Boundjou, O., and Kebede, T. (2016). Performance Analysis of Machine Learning and Pattern Recognition Algorithms for Malware Classification, journal = 2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS). pages 338–342.

[32] Nataraj, L. and Karthikeyan, S., J. G. M. B. (2011). Malware Images: Visualization and Automatic Classification. *Proceedings of the 8th International Symposium on Visualization for Cyber Security 2011*, pages 1–7.

[33] Noor, U., Shahid, S., Kanwal, R., and Rashid, Z. (2019). A Machine Learning based Empirical Evaluation of Cyber Threat Actors High Level Attack Patterns over Low level Attack Patterns in Attributing Attacks. pages 1–19.

[34] Nye, J. Deterrence and Disuasion in Cyberspace. *International Security*, 41(3):44–71.

[35] Nye, J. (2011). Nuclear Lessons for Cyber Security? *Strategic Studies Quarterly*, 5(4):18–38.

[36] Pachhala, N., Jothilakshmi, S., and Battula, B. (2023). Enhanced Malware Family Classification via Image-Based Analysis Utilizing a Balance-Augmented VGG16 Model. *Traitement Du Signal*, 40(5):2169–2178.

[37] Panda, P., Kumar, O., Marappan, S., Ma, S., S., M., and Nandi, D. V. Transfer Learning for Image-Based Malware Detection for IoT. *Sensors*, 23(6):3253–3303.

[38] Rani, N., Saha, B., and Shukla, S. (2024). A Comprehensive Survey of Advanced Persistent Threat Attribution: Taxonomy, Methods, Challenges and Open Research Problems.

[39] Rapid7. Documentation. Available at: https://docs.rapid7.com/insightidr/.

[40] Rid, T. (2011). The Cyber War Will Not Take Place. *Journal of Strategic Studies*, 35(1):5–32.

[41] Rid, T. and Buchanan, B. (2015). Attributing Cyber Attacks. *The Journal of Strategic Studies*, 38(1-2):4–37.

[42] Ross, R. (2011). Managing Information Security Risk: Organization, Mission, and Information System View. Technical report, National Institute of Standards and Technology.

[43] Sanger, D. and Brooks, M. (2024). *New Cold Wars*. Crown Publishers.

[44] Sathyanarayan, V., Kohli, P., and Bruhadeshwar, B. (2008). Signature Generation and Detection of Malware Families. *Information Security and Privacy. ACISP 2008. Lecture Notes in Computer Science*, 5107:336–349.

[45] Saxe, J. and Sanders, H. (2018). *Malware Data Science*. No Starch Press.

[46] Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ICLR 2015*, pages 1–14.

[47] Temple-Raston, D. (2021). A 'Worst Nightmare' Cyberattack: The Untold Story Of The SolarWinds Hack. *NPR*.

[48] USCYBERCOM. Our History. Available at: https://www.cybercom.mil/About/History/#:~:text=The%20Joint%20Chiefs%20of%20Staff,actors%20in%20this%20new%20domain.

[49] Xiao, N., Bo, L., Wang, T., and Chen, Y. (2024). APT-MMF: An advanced persistent threat actor attribution method based on multimodal and multilevel feature fusion. *Computers Security*, 144:103960.

[50] Xie, B., Qin, J., Xiang, X., Li, H., and Pan, L. (2018). An Image Retrieval Algorithm Based on GIST and SIFT Features. *International Journal of Network Security*, 20(4):609–616.

[51] Yuxin, D. and Siyi, Z. (2017). Malware detection based on deep learning algorithm, journal = Neural Computing and Applications. 31:461–472.