SANDIA REPORT
SAND2022-14321
Printed September 2022

Sandia National Laboratories

MalGen: Malware Generation with Specific Behaviors to Improve Machine Learning-based Detectors

Michael R. Smith
Armida Carbajal, Eva Domschot, Nicholas T. Johnson, Akul A. Goyal, Christopher C. Lamb, Joseph P. Lubars, W. Philip Kegelmeyer, Raga Krishnakumar, Sophie Quynn, Ramyaa Ramyaa, Stephen J. Verzi, Xin Zhou

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

MalGen: Malware Generation with Specific Behaviors to Improve Machine Learning-based Detectors

Michael R. Smith
Sandia National Laboratories
Albuquerque, NM
msmith4@sandia.gov

Armida Carbajal
Sandia National Laboratories
Albuquerque, NM
ajcarba@sandia.gov

Eva Domschot
New Mexico Tech
Socorro, NM
eva.domschot@student.nmt.edu

Bridget I. Haus
USC Verterbi School of Engineering
Los Angeles, CA
bhaus@usc.edu

Akul A. Goyal
University of Illinois Urbana-Champaign
Champaign, IL
akulg2@illinois.edu

Nicholas T. Johnson
Cerebras Systems Inc.
Los Altos, CA
nick@cerebras.net

Christoper C. Lamb
Sandia National Laboratories
Albuquerque, NM
cclamb@sandia.gov

Joseph P. Lubars
Sandia National Laboratories
Albuquerque, NM
jplubar@sandia.gov

W. Philip Kegelmeyer
Sandia National Laboratories
Livermore, CA
wpk@sandia.gov

Raga Krishnakumar
Sandia National Laboratories
Livermore, CA
rkrishn@sandia.gov

Sophie Quynn
Sandia National Laboratories
Livermore, CA
squynn@sandia.gov

Ramyaa Ramyaa
New Mexico Tech
Socorro, NM
ramyaa@cs.nmt.edu

Stephen J. Verzi
Sandia National Laboratories
Albuquerque, NM
sjverzi@sandia.gov

Xin Zhou
University of Houston
Houston, TX
xzhou1@sandia.gov

SAND2022-14321

ABSTRACT

In recent years, infections and damage caused by malware have increased at exponential rates. At the same time, machine learning (ML) techniques have shown tremendous promise in many domains, often out performing human efforts by learning from large amounts of data. Results in the open literature suggest that ML is able to provide similar results for malware detection, achieving greater than 99% classification accuracy [49]. However, the same detection rates when applied in deployed settings have not been achieved. Malware is distinct from many other domains in which ML has shown success in that (1) it purposefully tries to hide, leading to noisy labels and (2) often its behavior is similar to benign software only differing in intent, among other complicating factors. This report details the reasons for the difficultly of detecting novel malware by ML methods and offers solutions to improve the detection of novel malware.

We propose to detect malware by detecting behaviors commonly exhibited by malware such as DLL injection, and process hollowing. This is based on the assumption that there is a set of behaviors that are common to most malware samples and detecting them will generalize to novel malware. Additionally, detected behaviors point analysts toward appropriate handling and mitigation strategies, which is not the case with a binary benign/malicious classification. A behavior labeling method was developed and was used to label an existing malware dataset. Results show that detecting malicious behaviors is much more difficult than simply classifying malware and goodware—achieving 80% accuracy compared to reported 99% accuracy from classifying malware and goodware. This drop is due to several reasons which are detailed in the report.

We also propose to evaluate the performance of detecting novel malware by holding out a malware family for testing and training on the other families. Traditional ML evaluation will shuffle the data and then split the data into training and testing. Our method addresses the use-case when novel malware families are encountered and they require more than just a malicious or benign designation. Our results suggest that this type of evaluation is much more difficult than traditional methods and provides more realistic results, albeit, significantly worse. For our behavior detection, accuracy decreases from 80% to 68% across all behaviors when holding out a malware family from training.

We show that the degradation in performance is because each malware family has distinct characteristics resulting in high extrapolations by an ML model. Here, an ML model should return an "I do not know" response and request further analysis from an analyst. We run a number of experiments that compare novel malware families to the training data using different feature representations including a genomics-inspired distance measure and features extracted by deep learning. Generally, held-out families are significantly different from the training data, resulting in unpredictable results. This has been observed generally in the ML community [22, 9]. We empirically demonstrate this in the domain of malware detection.

In an attempt to improve the detection of malware behaviors, we examine the impact that additional synthetic data has on the performance of an ML model in detecting behaviors in novel malware families. We find that while synthetic data does improve the performance of ML models, often simpler methods perform better than more complicated ones. Two generative modeling techniques were examined to produce synthetic malware samples such that the behaviors present are able to be specified externally. The difficulty is due to finer grained analysis of the executable

and modifying the problem from a binary classification problem to a multi-label problem. The addition of synthetic data increases the overall accuracy from 68% to 70%. While far less accurate than measures presented in academic analyses, we believe that this is more representative of real-world performance and allows models to be properly placed within a malware detection system. We suggest that in highly dynamic environments ML pipelines should determine whether an ML model is competent in the area of new data and should involve mechanisms to improve over time with a human in the loop.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

Table 0-1.

Abbreviation	Definition
AE	Auto-encoder
ACC	Accuracy
AUC	Area Under the Curve
CE	Cross-entropy
CFG	Control Flow Graph
CNN	Convolutional Neural Network
CSVAE	Conditional Subspace Variational Auto-Encoder
DFA	Data Flow Analysis
DLL	Dynamically Linked Library
DNN	Deep Neural Network
DL	Deep Learning
DT	Decision Tree
GAN	Generative Adversarial Network
GBDT	Gradient-Boosted Decision Tree
DOE	Department of Energy
KNN	K-Nearest Neighbors
MA	Malware Analysis
MBC	Malware Behavior Catalog
ML	Machine Learning
PA	Program Analysis
PCA	Principle Component Analysis
PE	Portable Executable
ReLU	Rectified Linear Unit
RF	Random Forest
RSS	Residual sum-of-squares
SMOTE	Synthetic Minority Oversampling TEchnique
SVM	Support Vector Machine

1. INTRODUCTION

Malware poses a significant threat to computer networks, privacy, and safety. The threat of malware continues to increase as more domains incorporate automated components into their systems. Domains such as health care [120], critical infrastructure [77], self-driving cars [82], and IoT devices [143] are experiencing significant increases in malware attacks. Further, sophisticated actors are able to by pass detection—often with trivial effort despite considerable resources that are provided for defense. In reponse, machine learning (ML) and deep learning (DL) methods have been investigated to detect malware with high hopes of attaining the similar success experienced in other domains, where these methods achieve better than human performance [38].

Recently, ML performance has improved significantly—particularly in DL, a class of ML algorithms that uses multiple layers in a neural network. State-of-the-art performance has been achieved in computer vision [123, 42], medical diagnosis [30], machine translation [127, 92], and game play [71, 106]. This creates hype for similar success in different applications including malware detection. ML and DL in malware detection promises to reduce manual labor by orders of magnitude, reduce errors, work at scales and speeds previously unobtainable, and detect novel malware [32, 93]. As such, many anti-virus (AV) companies are turning to ML and DL to improve malware detection and mitigation.

Many research efforts in ML have reported impressive results in malware detection and malware family classification—achieving better than 99% accuracy [74, 100, 4]. Despite the success of ML and DL based malware detection in controlled settings, the models often do not perform as well once deployed. The reasons vary, but include data quality, label quality, and adversarial adaptation to the models. We hypothesize that a semantic gap exists between the ML and malware analysis (MA) communities. MA typically identifies malware based on observed malicious or unintended behaviors requiring manual examination (e.g. DLL injection and disabling security tools.). MA has yet to establish meaningful behavioral categories or create realistic and challenging benchmark datasets. ML blithely uses easy benchmark datasets, focusing merely on malware classification and is easily satisfied by its artificial success. We believe that aligning the two communities by focusing on detecting behaviors in an executable will facilitate the development of ML and data processing techniques specific for MA, and improve its performance in identifying novel malware. We do so by focusing the predictive capabilities of ML to predict behaviors in executables.

Key limitations of previous work include (1) the limited availability of behaviorally labeled data and the manually intensive process required to label malware samples with behaviors, and (2) evaluation methods of ML-based malware detection that are not representative of deployed environments. We propose that the current evaluation methods are insufficient for detecting novel malware and show that malware families not used for training (representing novel malware) are often significantly different from the other families. Thus, an ML model is forced to extraploate and its classification is not credible.

We manually label an initial dataset with behaviors and propose a more realistic evaluation of ML models for detecting *novel* malware (Chapter 3). While detecting behaviors aligns more closely with MA, it is a challenging problem.

Additionally, we examine the creation of synthetic novel malware (Chapter 5) to improve detection rates using DL generative models. We seek to build off of the success experienced in the image processing domain by using synthetic data from generative models. In the image domain, the field has advanced such that synthetic data often cannot be distinguished from real data [76] and has improved detection methods [38]. Originally, generative models could only specify which class of images to generate (e.g. faces) but could not control the attributes of the faces (e.g. hair color, eye shape, facial hair). Recent work has emerged that conditions the generation process on certain characteristics that should be exhibited [55, 54]. We examine these approaches in generating novel malware. Results on behavioral detection improve the overall balanced accuracy from 60.5% to 63.7% with the largest increase being in recall, improving from 40.1% to 58.4% (using one method).

Our primary contributions include:

- Adaptation of traditional malware analysis to examine behaviors, which is more naturally aligned with MA (Chapter 3),
- An initial behaviorally labeled dataset (Chapter 3),
- An automated approach for labeling threat reports that can be used for labeling executables (Chapter 4)
- Adaptation of two generative modeling techniques to produce malware with specific behavioral characteristics (Chapter 5),
- A three-pronged approach to validating that a specified behavior is present within a generated data point representing malware (Chapter 5),
- A demonstrable increase in overall performance in behavioral detection in ML-based detectors (Chapter 5), and
- An examination of *why* ML performs poorly when deployed by examining the percentage of samples are significantly different from the training data including novel distance measures for executables based on genomics (Chapter 6).

The report is compiled as many stand alone reports, many of which have been published previously. For this report, we include additional details that were omitted from published works due to space constraints or tangential interests by the publishing venue.

2. DETECTING MALWARE WITH MACHINE LEARNING

The ML culture generally emphasizes demonstrated performance improvements on benchmark datasets [107]. This approach has driven significant improvements but is completely dependent on an appropriate dataset. We suggest that ML-based MA can be improved by aligning the data used by ML with the goals of the MA community—specifically incorporating behavioral information. With the end goal of aligning the two communities and improving the identification of novel malware, we 1) provide ML perspectives that have led to success in other domains but may be lacking in MA, 2) survey current datasets that are used by ML for malware detection, 3) develop a method for behavioral annotations aligned with the MITRE ATT&CK® Matrix [122], 4) annotate the Microsoft Malware Classification Challenge dataset [100] with behaviors, and 5) train ML models for behavioral identification. We find that behavioral identification is a more difficult and interesting problem for ML than generally realized. A simple image-based DL model achieves slightly over 50% accuracy, transfer learning from malware family identification achieves 57% - 84% depending on the malware family while a majority class predictor outperforms both models. The results suggest that behavioral classification can generalize to novel samples from malware families not included in training but that MA-specific ML techniques are needed.

2.1. Machine Learning Background

We focus on *supervised* ML that learns by example from labeled data points. We denote the inputs as X and the labels or outputs as Y. Observed variables are represented in lower-case. Therefore, the i^{th} observation of X is written as x_i which can be a vector or a scalar. Following Friedman et al. [31] to more formally describe supervised ML, let

$$Y = f(x) + \varepsilon \tag{2.1}$$

describe the data where the noise ε has $E(\varepsilon) = 0$ independent of X. The goal of an ML algorithm, then, is to find an approximation $\hat{f}(x)$ to $f(x)$ that preserves the predictive relationship between X and Y. The approximation $\hat{f}(X)$ is learned from a training set $\mathscr{T}$ of N observed input-output pairs (x_i, y_i), $i = 1, \dots, N$.

An ML algorithm modifies the input-output relationship $\hat{f}(x_i)$ in response to the difference between the prediction $\hat{f}(x_i)$ and the observation y_i. Each learning algorithm has an associated set of parameters θ that can be modified to alter $\hat{f}(x)$ and many are *maximum likelihood estimators* assuming that the most likely values for θ provide the largest probability of observing Y given X.

The values of θ are found by minimizing a loss function L that measures the "goodness" of the model fit as a function of θ. For example, one loss function minimizes the residual sum-of-squares (*RSS*), and another, the cross-entropy (*CE*) loss when Y is a vector of K possible classes. Minimizing the loss function on the training data minimizes *training error*; however, the goal is to minimize error on unobserved data points (the *test* or *generalization error*). The expected generalization error can be decomposed as:

$$\begin{aligned} Err(x) &= E[(Y-\hat{f}(x))^2] \\ &= (E[\hat{f}(x)]-f(x))^2 + E[(\hat{f}(x)-E[\hat{f}(x)])^2] + \sigma^2 \end{aligned} \tag{2.2}$$

which is a sum, respectively, of the bias, variance, and the irreducible error. The irreducible error (σ^2) represents the inherent noise in the data (ε in Equation 2.1)—no matter how good the model is, there will be some amount of error. The bias is the difference between the average model prediction and the actual value. High bias refers to models that focus less on the training data risking oversimplification of the model. High variance models, on the other hand, focus more on the training data risking overly complex models. As the complexity of a model increases, the training error tends to decrease. The performance on training data is usually not a good indicator of how the model will *generalize* or how well it will perform on new data points. Thus, a trade-off between bias and variance is needed to achieve a model that generalizes the best to test data. For more details, see Friedman et al. [31].

There is an explicit dependence between training data, $\hat{f}(x)$, and the generalization error. Gathering, cleaning, and processing data requires large amounts of effort. An observed data point x_i needs to be represented in a format that an ML model can operate on. Most ML algorithms operate on vector representations. However, many interesting problems are *not* easily represented as vectors without discarding significant amounts of information. If the representation of the data does not contain the information required for the question that is being asked (e.g. is the behavior of this executable benign or malicious?) then this falls within the irreducible error (σ^2 from Equation 2.2). Additionally, an ML algorithm optimizes the loss on the labels and, therefore, the label needs to be aligned with the application. Better representations and labels of the phenomena can, thus, mitigate the irreducible error.

2.1.1. Overlooked Caveats of ML Successes

ML success is often built on decades of previous research and understanding of a given domain. For example, the success of convolutional neural networks (CNNs) [59] builds on decades of research in signal processing and data representation. The convolution is a mathematical operator that expresses the overlap between functions and can be thought of as blending one function with another. The convolutions in CNNs are a codification of convolutions where a function is based on the data instead of being explicitly defined. One key reason for the success of convolutions is their translational invariance, which is inherently important in object recognition because an object may be anywhere in the image. An analogous operator does not yet exist for binary executable analysis.

Success of ML in other domains has also been enabled by large amounts of labeled, relevant data. Li revolutionized computer vision and object detection by providing labels for relevant images [26]. Corresponding datasets do not yet exist for malware detection. Further, ML models do not always learn the intended concepts. For example, CNNs are biased towards learning texture rather than shapes and objects [34, 113]. This can make them susceptible to adversarial attacks and brittle to noise [105].

Additionally, the data in MA is significantly different from other domains in that it lacks proximity relationships, continuity, and ordinality, which are assumed by many ML algorithms. For example, pixel values of 123 and 122 are close in value and neighboring pixels have an assumed proximal relationship. Code blocks can jump to various locations in a binary, and values next to each other in numerical space can have significantly different meanings. Additionally, in real-world systems, goodware significantly outnumbers malware—less than 1% of all executables were reported as malware [132]. This class imbalance has been shown to exacerbate other issues in ML algorithms [116]. The combination of these issues makes applying ML to MA difficult.

2.2. Program Analysis Background

Program analysis (PA) consists of several processes that are used to reason about the behavior of a computer program and are leveraged in MA. PA is ultimately interested in program optimization and correctness. We highlight a subset of areas related to extracting features that could be used as input to ML algorithms.

In PA, a distinction is made between the *syntax* and the *semantics* of a program [41]. For programs, syntax is concerned with the form of expressions that are allowed (i.e. the sequences of symbols that are accepted by a compiler or interpreter). Semantics describe the effect of executing syntactically correct expressions (behavior). The semantics of a program require a defined syntax, at least at an abstract level. Identifying syntax is much easier than semantics. As shown in Section 2.1, an ML model depends on training data. If training data does not relate to behaviors, then expecting an ML model to learn them is unreasonable. Generally, extracting syntactic features is significantly simpler than extracting semantic features.

2.2.0.1. Static Analysis Techniques

In static analysis, a program is analyzed in a non-runtime environment. The analysis is generally performed on a version of the source code, byte code, or application binaries. Static analysis is used frequently for optimization, such as dead code elimination, or for verification such as identifying potentially vulnerable code and run-time errors. Generally, static analysis *approximates* all possible executions of a program through abstract interpretation or data-flow analysis. One challenge for static analysis is that behavior is limited to what happens internal to the program, and the environment is not analyzed.

Several static analyses techniques capture semantic information. However, many datasets used for ML favor syntactic features that are easier to capture. For example, data flow analysis (DFA) collects information about the possible states at various points in a program [112] by constructing a control flow graph (CFG) that represents the program. Each node in the CFG represents a basic block or a sequence of consecutive instructions. Control can *only* enter at the beginning of the chunk of code in a node and leaves at the end. Directed edges in the graph represent jumps between one chunk of code to another. CFGs can capture significant semantic information. However, it is not in a form that most ML algorithms can easily digest, and there are no obvious means to transform it without significant semantic loss.

Abstract interpretation [20, 21] is a theoretical framework to formalize the approximation of computing abstract semantics. Here semantics refer to a mathematical characterization of possible behavior of a program. The most precise semantics describe accurately the actual execution of a program and are called concrete semantics. Small-step, or structural oriented, semantics [90] describe a program in terms of the behaviors of its basic operations. The behavior of a program is a current state (program point and the environment) given a starting state and series of operations. For example, consider the simple code below.

```
1: n=0
2: while n < 500 do
3:    n = n+1;
4: end
5: exit
```

Analyzing the program would yield:

$$< 1, n \Rightarrow \Omega > \rightarrow < 2, n \Rightarrow 0 > \rightarrow < 3, n \Rightarrow 0 > \rightarrow < 4, n \Rightarrow 1 > \rightarrow$$
$$< 2, n \Rightarrow 1 > \rightarrow < 3, n \Rightarrow 1 > \rightarrow < 4, n \Rightarrow 2 > \cdots < 5, n \Rightarrow 500 >$$

Operational semantics, such as small-step semantics, combine logical conclusions about program syntax in order to derive semantic meaning. Assuming the interpretation of syntax is correct, this also allows for the construction of proofs about program behavior.

Big-step, or natural, semantics [48], like small-step semantics, define basic components to describe the semantics of a program. Rather than using the basic operations like small-step, big-step analytics defines the semantics of functions. Both are techniques that derive semantic meaning from a program and could be looked to as inspiration for features as opposed to techniques derived from computer vision or other domains. It is worth noting that both of these techniques limit behavior to what happens internal to a program or segment. They do not take into account the effects on the full environment as this is inherently intractable and represents a key difficulty in modeling malware for ML and MA.

Another key static analysis approach over programs is symbolic execution. Symbolic execution techniques build a mathematical representation of a program based on the input and output of various subroutines or functional blocks [3, 68]. In this representation, independent variables represent key input values. Constraint solvers, for example, can then solve for the variables, identifying what kinds of inputs are required for a particular output state [101, 44]. From a vulnerability analysis perspective, this can allow analysts to identify input that can potentially lead to system failure states, which may be exploitable. Symbolic execution techniques suffer from state explosion proportional to the size and complexity of a given program [58]. Other static analysis techniques provide disassembly and intermediate representations (from binary to machine code). However, care needs to be taken to preserve semantic information.

2.2.0.2. Dynamic Analysis Techniques

Dynamic analysis executes a program and *precisely* analyzes a single or limited number of executions of a program. The coverage of dynamic analysis is dependent on the test inputs, which for malware analysis, can be variants of the operating environment. Often, a subset of the interactions with the underlying operating system are analyzed such as system calls, or memory reads and writes. Dynamic analysis is often used to ensure program correctness and find errors in code [72].

Most dynamic analysis techniques use instrumentation to insert code into a program to collect run-time information. The instrumentation will vary based on the type information that is desired and the type of code that is available (e.g. source code, static binary, and dynamic binary). Most tools track function calls (including system calls), capture the input parameters, track application threads, intercept signals, and instrument a process tree. The output from dynamic analyses has often been heralded by ML practitioners for modeling behavior as it captures observed effects on the environment. However, because of a lack of context and the challenges outlined previously, the representations that are suitable for ML often lose the semantic information.

2.3. Motivating Case Studies

To help motivate the semantic gap between ML and MA, we walk through a case of using ML to identify malware persistence in registry keys, highlighting the difficulty in generating an appropriate dataset and extrapolating results to real-world scenarios.

Briefly, the registry is a hierarchical key-value database that stores configurations, program settings, and user profiles. The registry is capable of storing commands to execute when the system is loaded and is commonly used for maintaining persistence on the Windows operating system [70]. In addition to system software, malware takes advantage of the the registry to ensure that it is loaded as needed. As an example, a key can have the format:

```
\HKEY\_LOCAL\_MACHINE\System\...\...\ImagePath
```

and a value that can take multiple formats such as:

```
C:\Windows\System32\svchost.exe -k netsvcs
```

The example represents a path to an executable, but the values are capable of storing many complex data types (e.g., binary data, scripts, etc.). Thus, even with this relatively simple example, representing this data in a format suitable for ML is non-trivial.

2.3.1. Data Collection & Parsing

As with most use cases, collecting data is not challenging, but obtaining labels and properly representing the data is. Registry data was collected from Windows machines across a corporate network for two years, resulting in approximately 20 million (host, registry key, timestamp) tuples, with roughly 136,000 unique registry entries. Registry data was collected from executing publicly available malware in a sandbox environment producing 200 registry entries.

Despite capturing effects on the environment, the raw registry data is not suitable for ML algorithms due its variability. As there are a finite number of keys, they are represented as a 1-of-*N* encoding. The value portion is more complex and describes what is being executed. Ideally, the value consists of a path and a file that can be parsed into its relative components. However, in some cases one program will launch another such as when services are launched using `svchost.exe`. For these situations, a parser that found the launching program (e.g., `svchost`) as well as the program that is being launched. Each launching program is parsed according to the expected syntax (e.g., `svchost` should have a `-k` flag), and when found, these launching programs constitute another categorical variable. Additionally, different file types exist which are represented as categorical variables per file type including any associated options (e.g., command-line flags).

After the aforementioned parsing, the specific folders in a given path are used as terms in a traditional bag-of-words model. The resulting data is high-dimensional (over 12,000 terms) and extremely sparse with few unique observations (i.e., the number of unique rows is close to the number of columns). Principal Component Analysis (PCA) was performed to reduce the dimensionality while preserving as much information about the original space as possible. Several assumptions and trade-offs were made to produce a format suitable for ML which discarded some information.

2.3.2. Experimental Analysis and Bias

Labels are needed to identify which registry keys are associated with malicious or benign activity. Initially, any key that occured on a large number of hosts was labeled benign and those that were modified by the malware as malicious. Experimentation with this setup resulted in a cross-validated area-under-the-curve (AUC) score of 0.99. Performance this high should suggest that the ML problem is too simple and thus will not be practically useful. Upon closer inspection, the malicious examples came from specific hosts and identifying the malware labels was a simple process. Registry keys that occur on a large number of systems tend to be associated with programs and drivers in the system space (e.g., in `C:\Windows\system32`). However, the majority of the malicious keys are associated with the user and program space. A simple weak indicator that looks for absence of the keywords “windows”, “system”, or “program” to determine maliciousness provides an AUC of 0.85. Thus, the model inadvertently distinguishes system space keys versus other keys and is not likely to generalize well.

Only labeling keys modified by malware as malicious and all others as benign results in an AUC of 0.96 for ML and an AUC of 0.53 for the weak indicator—not significantly better than random.

This result is promising as the gap between ML and a simple indicator increased significantly. However, this correction is likely still optimistic. Cross-validation tends to be optimistic in general, due to the fact the errors are not independent. Also, this data is not likely to contain all possible examples of malware that uses legitimate software registry for persistence. Creating a generalizing principle beyond a signature is challenging. Another confounding factor is that malware can execute behavior that is not malicious to avoid detection, and, thus, make it difficult to derive ground-truth labels.

2.3.3. Other Examples

This paper is not the first to recognize the gap between the research and actual deployments. Sommer and Paxson [119] point out the discrepancies in network intrusion detection. They observe that the task of intrusion detection is fundamentally different from other applications, making it more challenging. They identify six key challenges: 1) ML is better for finding similarities rather than differences, 2) very high cost of classification errors, 3) a semantic gap between detection results and their operational interpretation, 4) enormous variability in what is "normal", 5) difficulties in sound evaluation of the results, and 6) operating in an adversarial setting. In the context of detecting malware, other work noted discrepancies particularly with respect to the precision of malware—indicating a large jump in false negatives when deployed in real-world settings stemming from the difference in the proportion of malware and the difficulty on modeling "normal" in executables [114].

2.4. Current Datasets

In this section, we briefly discuss the importance of benchmark datasets historically in ML research and the challenges in curating a benchmark dataset for malware, and we review existing datasets. Despite several attempts, a benchmark dataset for malware classification has yet to be widely adopted and have a high impact for ML-based malware classification.

2.4.1. The Utility of Benchmark Datasets

The progress of any research field depends on reproducible comparisons between methods to quantify progress on a given task. For ML, benchmark datasets facilitate comparisons between learning algorithms. In addition, benchmark datasets drive ML success and guide research in several application areas such as object detection [26], facial recognition [88], handwriting recognition [60], recommender systems [36], and question and answer systems [95].

Benchmark datasets facilitate research that would not otherwise be possible. A benchmark dataset dictates several important characteristics of the research that uses it. First, it determines which features are used based on data representation. Second, it determines the impact of ML models developed using the data. If the dataset misrepresents the real-world settings or is ill-suited for the task, the ML model will perform poorly despite performing well on the benchmark dataset.

2.4.2. Challenges in Curating a Malware Dataset

2.4.2.1. Dynamic Environment

Malware classification is a dynamic problem in which the target is constantly changing and evolving. In ML parlance, this is concept drift where the distribution of the target changes over time from what was used for training [33]. In cases with concept drift, performance often degrades and has been shown to be significant in malware detection [53]. Additionally, malware authors intentionally alter malware to avoid detection using several obfuscation techniques including polymorphic code and garbage code insertion. In many other domains, the attempt to deceive is not as prevalent. Malware authors can purposefully alter their malware to subvert an ML model trained on a given dataset.

2.4.2.2. Releasing Data

Many AV companies hold their collection of malware samples as proprietary. As mentioned above, malware authors could also use this information to thwart existing architectures built on this data—risking their clients' systems. Another consideration is that each collection service may be biased to certain demographics, location, network infrastructure, political ties, etc. that may attract certain types of attacks.

2.4.2.3. Feature Representation

Distributing live malware samples is a security risk, especially for those not accustomed to handling malware. As a result, most recent datasets first extract predetermined features from a set of malware examples limiting the representation.

2.4.2.4. Obtaining Labels

Many of the current datasts use tools like VirusTotal[1], which provide the output from multiple antivirus tools, to create labels. Often only samples that are identified as malware by a majority of the tools are labeled as malware and others are discarded providing a biased sample that uses the most popular examples. This is not representative of the data that will be encountered in real-world deployments. Using the most popular malware and goodware examples can create easily separable training data, and overly optimistic performance expectations [63, 86]. Several works have proposed methods for improving the labeling and not discarding as many of the "unpopular" samples [50, 108].

Table 2-1. Summary of malware datasets used for ML

Dataset	Year	Cite	Representations	# Samples	Labels	Labeling	Max Acc
			Highly Cited				
VX Heaven[1]	2010	?	Live executables	Varies	Varies	Curated	N/A[3]
VirusShare[2]	2011	> 300	Live executables	Varies	Varies	Curated	N/A[3]
MalImg [73]	2011	417	Gray-scale images	9,458	25 Families	MSSE	99.80%
MS Malware Classification [100]	2015	76	Disassembly and hexadecimal	10,868	9 Families	MSSE	99.97%
EMBER [4]	2017	46	Parsed and histogram counts	1,100,000	Good, Bad, ?	VirusTotal	99.90%
MalRec [110]	2018	11	System calls, memory contents[4]	66,301	1,270 families	VirusTotal[5]	N/A[1]
			Less Cited				
Malware Training Sets [97]	2016	2	Counts from analysis reports	4764	4 families	Curated	-
Mal-API-2019 [14]	2019	1	System call traces	7,107	8 families	VirusTotal	-
Meraz'18 Kaggle [43]	2018	~1	Parsed features	88,347	Good v Bad	Curated	91.40%[6]

[1] `http://vxheaven.org/`
[2] `https://virusshare.com/`
[3] There is no established dataset making comparisons between studies difficult.
[4] Also provides full system replays of malware execution, however the authors note non-trivial efforts to get them to work on other systems.
[5] Uses AVClass [108] which leverages VirusTotal.
[6] Reported accuracy on the Kaggle challenge leader board.

2.4.3. *Review of Datasets*

There are currently several proposed repositories for ML-based malware detection that either identify malware families or discriminate malware from goodware; these are summarized in Table 2-1.

2.4.3.1. Live Malware Repositories

There are several repositories containing live malware—posing a threat to inadvertently infecting one's system and providing malicious software to adversaries. However, the malware samples provide a valuable resource enabling MA and research. VX (Virus eXchange) heaven[2] with the mantra: "Viruses don't harm, ignorance does!" seeks to provide information about computer viruses including articles, source code, malware samples, and books to help educate whomever is interested. Several similar repositories exist including theZoo (a.k.a. the malware DB)[3] and Virus Share[4] for free, or Virus Total[5] which is available for a fee and also contains benign samples.

Ideally, a researcher has access to the raw data. Even with access to the entire malware sample, as discussed previously, getting the samples into a format suitable for ML is challenging. Often only simple features are extracted such as metadata from the PE header, imported DLLs, and byte counts (more details on extracted features are given in Section 2.4.4). Using simple features resulted in high detection rates (98.8%) [129] leaving little room for improvement. With live malware repositories, studies are difficult to compare as each selects different subsets of malware samples to analyze and there is no common base publication to trace attribution. However, the

[1]`https://www.virustotal.com/gui/home/upload`
[2]`http://vxheaven.org/`
[3]`https://thezoo.morirt.com/`
[4]`https://virusshare.com/`
[5]`https://www.virustotal.com/gui/home/upload`

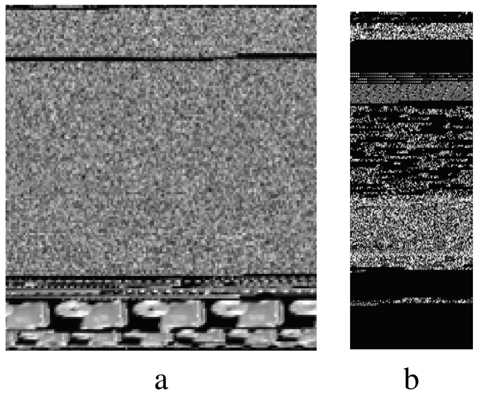


Figure 2-1. Examples of malware represented as gray-scale images from a) Fakerean and b) Dontovo.A malware families [73].

amount of malware samples is impressive. On Virus Share, there over 34 million samples as of this writing.

2.4.3.2. MalImg

The MalImg dataset [73] was motivated by the success of deep learning (DL) in image processing. In MalImg, binary values from an executable were converted to 8-bit unsigned integers, organized into a 2-dimensional array and visualized as a gray-scale image (Figure 2-1).

The authors observed that malware belonging to the same family were visually similar in layout and texture. In their preliminary analysis, the authors extracted texture features from the generated gray-scale images using GIST [124]. On a dataset with 9,458 malware samples from 25 different families, a 3-nearest neighbor classifier[6] achieved 97.18% accuracy and 99.2% when variants of a malware family were combined. Follow-up work achieved accuracy of 98.52% when using a convolutional neural network [49] and 99.80% with principal component analysis and a support vector machine [35].

2.4.3.3. MS Malware Classification

The Microsoft Malware Classification Challenge [100] was developed as a Kaggle competition to classify malware samples into one of nine malware families. It was released in 2015 and has since been used in several studies, being cited more than 70 times at the time of this writing. The hexadecimal representation of the binary content without the PE header as well as meta-information (function calls, op codes, strings, etc.) from the IDA disassembler was provided for each malware sample. Current reported performance on the dataset claims 99.70% [35] and 99.97% accuracy [49] using image-based features.

[6]A classifier that predicts the majority class of the 3-closest examples

Table 2-2. Reported accuracy, precision, recall and F1-score on the EMBER dataset [128].

Model	Accuracy	Precision	Recall	F1
MalConv [93]	98.8%	99.7	97.9	98.8
GBDT [4]	97.5%	99.0	96.2	97.1
KNN	95.1%	95.5	94.6	95.1
DT	96.9%	97.1	96.7	96.9
RF	97.0%	98.6	95.3	96.9
SVM	96.1%	96.4	95.7	96.1
DNN	98.9%	99.7	98.1	98.9
Modified MalConv [128]	99.9%	99.7	100.0	99.9

2.4.3.4. EMBER

The Endgame Malware BEnchmark for Research (EMBER) dataset [4] is a collection of extracted features from 1.1 million executables divided into 900k training and 200k test samples and has emerged as one of the most popular datasets. EMBER provides features that are consistent with previous work and has been used in several studies. The authors of EMBER achieved a 98.2% detection rate with a 1% false positive rate. This was further improved to a 99.4% detection rate with an AUC value of 0.9997 [87]. Further, Vinayakumar et al. [128] modify a DL technique aimed at malware detection (MalConv [93]) and achieve nearly perfect performance.

2.4.3.5. Malrec

Contrary to the other datasets, Malrec provides system-wide traces of malware executions that can be replayed. It is intended to address the danger of releasing live malware and the limited amount of data that can be collected when running in a sandbox. The replays capture the state of a system that is executing malware and thus captures the behaviors of malware while not releasing actual malware and provides the ability to retrospectively extract features that were not considered relevant when the malware was first executed. There are currently 66,301 malware recordings collected over a two-year period. The major downside is the very large size of the data (currently 1.3TB) and the complexity in setting up the system to extract a dataset.

The authors extracted multiple datasets from the system-wide recordings including bag-of-word counts for textual data in memory, network activity, system call traces, and counts of data instruction mnemonics. They examined a use case in which they extracted features to use for ML. They created a word list of all words between 4 and 20 characters long from the English Wikipedia—resulting 4.4 million terms. They then monitored memory reads and writes looking for byte sequences that matched words in their list. Terms were removed that appeared in a baseline of running goodware as well as frequent terms that appeared in more than 50% of the samples and rare terms that appeared in less than 0.1% of the samples resulting in ~460,000 terms. The dimensionality was further reduced to 2048 input features using PCA. DL on this data achieved a median F1-score of 97.2% across all of the malware families.

Despite having system-wide information, the PCA summary was sufficient for their dataset to achieve high accuracy. This presents somewhat of a paradox in the claims of ML and what is observed in deployed systems. Analyzing the memory contents in a bag-of-words fashion loses context, and we argue, that it is akin to learning a signature. We conclude that the ML model is able to quickly learn an effective signature-based malware detection system.

2.4.3.6. Other Datasets

Other datasets have been created, often by other security companies and hobbyists [97, 14, 43]. These datasets have not been widely adopted nor is it apparent how much maintenance they receive. We include them here for completeness, but they do not provide any new feature representations.

2.4.3.7. ML Perspectives

From an ML perspective, achieving such high classification accuracy is somewhat concerning as there is fear that the model has either overfit the training data (will have high generalization error) or the training data is easily separable. Thus, the dataset may not represent real world conditions well and lead to unrealistic performance expectations. For EMBER, the authors point out that the classes were easy to correctly classify and have attempted to make the task more challenging [103] in addition to other modifications [102]. The baseline on the updated data is 86.8%. Unfortunately, there are few results on the updated dataset.

2.4.4. Analysis of Datasets and Features

In this section, we examine which features contribute to the performance of an ML model across the datasets. We find that 1) the most useful features vary across datasets and 2) very few papers attempt to extract semantic features *and* are careful to maintain semantic information. We suggest that the ML models operate on patterns in the data not used by the MA community and that syntactic features are useful for discriminating between existing malware classes similar to how non-intuitive textures in images are useful for object detection. Similar to object detection in images, the models should not be expected to detect novel forms of malware based on their behavior as there is a semantic gap between the data and the task.

Raman [96] examined which features are the most discriminative between malware samples from VX Heaven and software that comes installed by default on Windows operating systems. They were able to achieve a true positive rate of 98.6% with a false positive rate of 5.7% by only extracting *seven* features from the files. Further examination revealed that the ML algorithm learned to discriminate between Microsoft and non-Microsoft executables. As the dataset does not represent the real-world problem well, it affects the robustness of an ML model trained on that data. A high false negative rate would be expected with a broader set of goodware.

Ahmadi et al. [1] extracted a large number of features that are commonly used in ML models from the hexadecimal representation and disassembled files from the Microsoft Malware Classification Challenge dataset with the intent of identifying features that are the most discriminative. The examined features include:

1. byte counts (BYTE).
2. the size of the hexadecimal representation and the address of the first byte sequence (MD1).
3. byte entropy (ENT).
4. image representation using Haralick features (IMG1) and Local Binary Patterns (IMG2).
5. histogram of the length of strings extracted from the hexadecimal file (STR).
6. the size of, number of line in the disassembled file (MD2).
7. the frequency of a set of symbols in the disassembled file (-, +, *,], [, ?, @) (SYM).
8. the frequency of the occurrence of a subset of 93 of possible operation codes in the disassembled file (OPC).
9. the frequency of the use of registers (REG).
10. the frequency of the use of the top 794 Window API calls from a previous analysis of malware (API).
11. characteristics of the sections in the binary (SEC).
12. statistics around using db, dw, and dd instructions which are used for setting byte, word, and double word and are used to obfuscate API calls (DP).
13. the frequency of 95 manually chosen keywords from the disassembled code (MISC)

Table 2-3 shows the classification accuracy on the training set and from using 5-fold cross-validation for each subset of extracted features using gradient-boosted decision trees. There are several feature groups that achieve over 99% accuracy including MISC which counts the occurrence of a set of hand-selected keywords. Surprisingly, MD1 and MD2 (i.e. file size) achieve about 85% and 76% accuracy respectively (random is 11.11%). This highlights a concern that there are features which may be discriminative but are an artifact of the dataset and can easily be manipulated adversarially.

Oyama et al. [81] examine which features have the largest impact on the EMBER dataset. EMBER contains several feature groups:

1. General file information from the PE header such as virtual size of the file, thread local storage, resources, as well as the file size and number of symbols.
2. Header information from the COFF header providing the timestamp, the target machine, linker versions, and major and minor image versions.
3. Import functions obtained by parsing the address table.
4. Exported functions.

Table 2-3. The reported accuracy on the training set and using 5-fold cross-validation on the Microsoft Malware Classification Challenge dataset [1].

Feature	# Features	Train Acc	5-CV Acc	Feature	# Features	Train Acc	5-CV Acc
Hexadecimal file				Disassembled file			
ENT	203	99.87%	98.62%	MISC	95	99.84%	99.17%
BYTE	256	99.48%	98.08%	OPC	93	99.73%	99.07%
STR	116	98.77%	97.35%	SEC	25	99.48%	98.99%
IMG1	52	97.18%	95.50%	REG	26	99.32%	98.33%
IMG2	108	97.36%	95.10%	DP	24	99.05%	98.11%
MD1	2	**85.47%**	**85.25%**	API	796	99.05%	98.43%
				SYM	8	98.15%	96.84%
				MD2	2	**76.55%**	**75.62%**

Table 2-4. Reported accuracy and number of features for each feature set in the EMBER dataset [81].

Feature set	Number of features	Accuracy
imports	1280	77.8
section	255	68.2
histogram	256	68.1
byte entropy	256	61.8
strings	104	61.4
general	10	56.0
header	62	52.9
exports	128	17.2
All	2,351	92.7

5. Section information including the name, size, entropy virtual size and list of strings representing section characteristics.

6. Byte histogram representing the counts of each byte value.

7. Byte-entropy histogram approximating the joint distribution of entropy and a given byte value.

8. Simple statistics about printable strings that are at least five characters long. Specifically providing information on strings that begin with “C:\", “http://", “https://" or “HKEY_".

Table 2-4 shows the accuracy for each feature group. The imports, which also have the largest number of features, has the highest accuracy as 77.8%. Oyama et al. report that header, imports, section, and histogram feature groups (together) achieve about 90% accuracy. The remaining 2.7% comes from the other feature groups.

Other work makes similar observations on various datasets further indicating a needed change in data representation:

- Count features (histograms) promotes overfitting and, combined with the labels, produces overly optimistic results [94].
- PE headers are the most discriminative [135].
- On VX Heaven, PE-Miner [111] achieves a detection rate greater than 99% only using structural information (PE and section header information), DLLs and object files.

Despite the impressive results, none of the features capture behaviors, as the data is not tailored to provide that information and the ML task is to detect malware—*not* identify behaviors.

3. BEHAVIORAL LABELS[1]

As we have shown, most datasets have focused on the features which do not contain behavioral information or it is lost when extracting features. As a first step to modeling behaviors, we take an alternative approach and provide labels for the behavior expressed in malware so the ML model can search for behavioral artifacts. Extracting behavioral information from an executable is a challenging problem that is a current research area for MA. We offer a process using threat reports for malware families to gather behavioral information.

We propose that behaviors consist of 1) a high-level intent and 2) low-level "primitives" that accomplish the behavior. A primitive is a sequence of ordered or partially ordered (i.e., one step depends on the previous step(s)) steps that must occur for the behavior to be successful. It is possible that primitives may contain conditional statements that are represented better by a directed graph than a sequence. These primitives vary by representation, malware family, or malware toolkit. They may also involve multiple systems (e.g., network and host). Thus, the feature representation is non-trivial. Additionally, the high-level intent of the executable is often not in the data. Further, multiple primitives may exist that accomplish the same behavior. For example, persistence is a common behavior for malware. This same outcome can be achieved variously by copying the malware to the *startup* folder or modifying the registry.

To label the behaviors, we leverage the MITRE Malware Behavior Catalog (MBC) [18]. MBC supports MA mapping behavior onto the MITRE ATT&CK Matrix [122]. ATT&CK documents common tactics, techniques, and procedures that advanced persistent threats use against Windows enterprise networks. The behaviors are organized according to the **objective** of the malware such as *Anti-Behavioral Analysis*, *Command and Control*, or *Persistence*. Each objective contains behaviors and code characteristics (**techniques**) that support that objective. For *Persistence* some of the techniques include *Application Shimming*, *DLL Search Order Hijacking*, and *Scheduled Task*. Each technique has an explanation for what it covers and can belong to multiple objectives—the *Hidden Files and Directories* technique could be under the *Defense Evasion* or *Persistence* objective.

We label the behaviors of a malware family using open-source threat reports and map the reported behaviors to the objectives and techniques outlined by MBC. In some cases, judgment has to be made about which category is the most appropriate. We label each family multiple times and use a peer review style to come to conclusions. The behavioral labels for each family are then extrapolated to individual examples. The current process is subjective and time intensive, and errors can be made based on variations of a malware family. Despite these limitations, the

[1]Much of this chapter comes from: Michael R. Smith, Nicholas T. Johnson, Joe B. Ingram, Armida J. Carbajal, Bridget I. Haus, Eva Domschot, Ramyaa Ramyaa, Christopher C. Lamb, Stephen J. Verzi, and W. Philip Kegelmeyer. Mind the gap: On bridging the semantic gap between machine learning and malware analysis. In Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security, pp. 49-60. 2020

behavioral labeling helps align the data to the desired task of identifying novel malware samples based on its behaviors. The labels would allow an ML model to directly learn the behaviors that may be not be discernible using only the family name. Future work includes the use of natural language processing tools to help automate the process. As new malware is analyzed, behaviors could be mapped into the MBC directly, bypassing the need for this method.

We label the Microsoft Malware Classification Challenge dataset which includes seven malware families, (*Ramnit*, *Lollipop*, *Kelihos*, *Vundo*, *Simda*, *Tracur*, *Gatak*).[2] The result of this process is a hierarchical behavioral labeling of each malware family as shown in Table 3-1. The compiled version is accessible at `https://gitlab.com/malgen/behavior_labels/`. The hierarchical structure captures both the high-level objective and employed technique(s) to meet that objective. By providing this labeling, an ML model will learn features that are associated with behaviors across all included malware families. By adjusting the target of the ML algorithms, better features and models can be developed that will improve the deployment of ML-base malware detectors.

Table 3-1. Malware Behavior Label Example for Microsoft Malware Classification Challenge

Objective:	Collection		Credential Access				Defense Evasion			…
Technique:	Local System	Man in the Browser	Hooking	Steal Web Session	Credential in Web Browser	Credentials in Files	Masquerading	Disable Sec Tools	Process Injection	…
Gatak	x	-	x	-	-	-	x	-	x	…
Ramnit	x	x	x	x	x	-	-	x	x	…
Lollipop	x	-	-	-	-	-	-	-	-	…
Kelihos	x	-	-	-	-	-	-	-	-	…
Vundo	x	-	-	-	-	x	x	x	x	…
Simda	x	-	-	-	-	-	x	x	-	…
Tracur	-	-	-	-	-	-	-	-	-	…

We are not the first to suggest the addition of behavioral labels; however, our process provides richer behavioral annotation at the cost of manual process and, as shown below, is a more challenging problem. Semantic Malware Attribute Relevance Tagging (SMART) [28] uses the output from anti-virus suites and parses keywords from the output providing a richer potential set of technical feature information than other approaches [142]. For example, the output could be `Win32.Virlock.Gen.8` or `TR/Crypt.ZPACK.Gen` and the key words extracted are `Virlock`, and `Crypt` and `ZPACK` respectively. This provides information that the malware is respectively ransomware and packed. The keywords align with the objectives in our process but do not provide consistent information on how the behavior is implemented, which our method provides. They report an accuracy of 95% on 11 possible tags. Our motives are similar to those if SMART, but the finer grained labeling that our method provides facilitates improved analysis and forces a ML algorithm to learn to distinguish behaviors that are important to MA.

High-level additional information was shown to improve the performance of an ML model [104]. We anticipate similar improved results as well as adjustments in follow-on studies that focus on behaviors.

[2] *Kelihos* versions 1 and 3 were combined because the threat reports did not distinguish between versions and we dropped Obfuscator.ACY as it was a bucket for obfuscated malware for which the family could not be determined.

3.1. Experiments on Behavioral Labels

We examine the ability of ML to generalize to novel malware on two initial behavior classifiers trained on the behavioral annotations using a binary image representation of the malware. As a malware family can have a combination of behaviors, we treat the problem as a multi-label classification problem and use a binary cross-entropy loss to encode this multi-label objective. Behavioral labels are consistent across malware families, allowing the identification of behavior in novel malware samples. In the following experiments, we hold out one family, train the models on the remaining malware families, and calculate the average accuracy for all of the behaviors on the held-out class. This allowed us to test the model's ability to reason about behaviors for a malware family it hadn't seen before. We compare the performance of the ML models with a simple majority class predictor that predicts a behavior is present if it was present in the majority of the training samples.

We establish a baseline model with a simple convolutional architecture, used for image processing, based on [91]. Using the input size of that architecture, we selected the first 1024 bytes from our malware samples as a (32,32) black and white image. This is a limited snapshot of malware but we saw accuracies above random chance when evaluating the model on family classification.

To develop a more robust model for predicting behaviors from malware binaries, we used the MalConv architecture [93], leveraging the code and model pre-trained on the EMBER dataset described in [4]. For consistency with the baseline model, we used the first megabyte of the malware sample represented as a flattened malware binary image as input. Additionally, we replaced the final fully connected layer to provide outputs for each of our behaviors, changing the output size to 56. This allowed us to fine-tune the model to evaluate the ability of transfer learning to classify malware behaviors based on features extracted for malware detection.

In Figure 3-1, we present the average accuracy across all behaviors for each variant of the experiment. Our transfer learning approach (MalConv)[3] outperforms our baseline model but the Majority Class classifier achieves better performance than both of them. This highlights the challenge ML faces when classifying behaviors for MA and the work that still needs to be performed. Since all of our test samples are from the same family, (i.e. labeled with the same behaviors), the perfect classifier would predict the same labels for each sample. If we had more families and could hold out multiple families for evaluating generalizability, then a naive classifier might not be as successful. Further, we examined the correlation of extracted features with the behavioral annotations. We used the features that were used by the winning team of the 2015 Kaggle Microsoft Malware Classification Challenge [130] (7600 features in total). Pearson correlations were calculated between the extracted features and the behaviors. Only 70 of the features had correlation $r > 0.7$ with any behavior. 64 of the 70 features that are strongly correlated with a behavior are 4-gram byte counts. Raff et al. concluded that n-byte grams were most often picking up string features [94]. One of the behaviors with a strong correlation with byte-grams is "rootkit", which often inserts malicious code into commonly used processes such as

[3]See Al Kadri et al. [47] for a more focused approach of applying transfer learning to MalConv for predicting malware families.

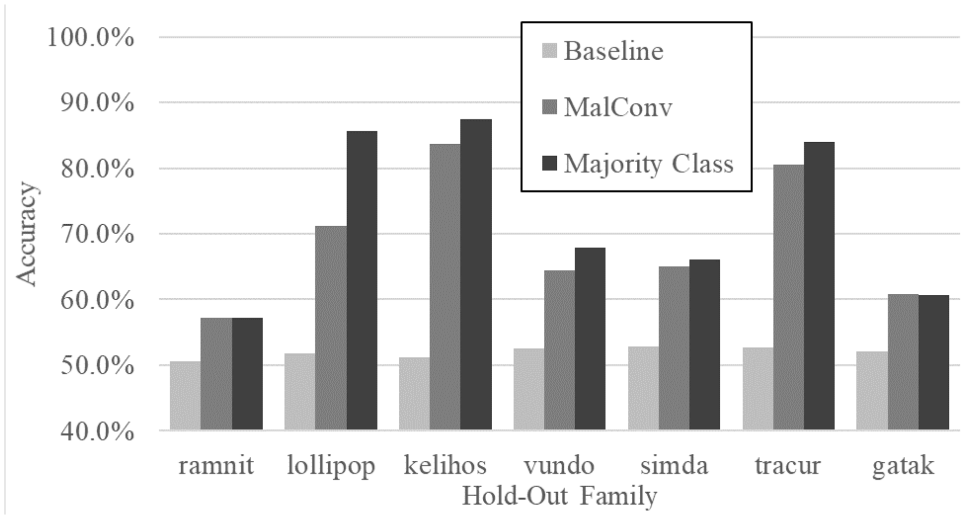


Figure 3-1. Classifying Behaviors for Unseen Families

DLLs which are often in disassembled string information and are one type of information in the 4-gram byte counts that could be correlated with behaviors. The behavior "access credentials" has a strong correlation with the instruction used to shift the bits in the register or memory, and is often used to encode or decode information.

The initial results presented here are not fully optimized but highlight a semantic gap between ML and MA based on the data used for analyses. There are several aspects that could be examined including balancing the behaviors, data generation, and hyper-parameter tuning. The results suggest that generalized behavior classification may be a more difficult problem than classifying malware families. They highlight the need for a dataset with behavioral labels and that simply using techniques that work well in other domains *does not* directly transfer to behavioral identification.

3.2. Automatic Executable Labeling

We examine the impact that a more robust set of malware binaries that are labeled with automatic executable labeling techniques has on the malware behavior classification model. One method of automatically labeling malware binaries with behavior labels is CAPA. CAPA is a tool developed by Mandiant that utilizes reverse engineering expertise by developing rules that use repetitive API calls, strings, and other features to automatically identify the capabilities in a piece of malware. CAPA is available for public release at `https://github.com/mandiant/capa`. Some of the CAPA rules provide a mapping from the identified capability to a behavior in the Malware Behavior Catalog (MBC). We found that, as of 2021-10-21, 255 CAPA rules have a mapping to a behavior in MBC.

Behaviors that are generated by CAPA are not a complete substitute in terms of fidelity for ground-truth behavior labels developed by a reverse engineer using hand labeling techniques. Nonetheless, these labels are generated quickly and can automatically label a large repository of

executable binaries. We were unable to test CAPA on the Microsoft Malware Classification Challenge dataset as CAPA requires the input binary to be in an executable format and the Microsoft Malware Classification Challenge dataset has been modified such that the binaries are no longer executable. Instead, we tested out CAPA using the Malpedia dataset, an invite-only, curated set of malware binaries found at
`https://malpedia.caad.fkie.fraunhofer.de`.

We obtained a version of the Malpedia dataset from 2021-06-07. The Malpedia dataset contains a variety of binaries from different operating systems along with corresponding metadata. We had a total of 3684 binaries, grouped into 1235 malware families, after pulling out only the windows executables from the Malpedia dataset. The final, processed Malpedia dataset did contain samples from malware families contained in the Microsoft Malware Classification Challenge dataset: kelihos (3 binaries), simda (4 binaries), and ramnit (21 binaries).

We ran the final Malpedia dataset through the capa tool and processed the CAPA output to generate behaviors from MBC for each binary. In total, 2968 binaries had at least one behavior label from capa. We found the highest behavior count to be 50 unique MBC behaviors for a single binary.

We found that the automatic executable labeling techniques such as CAPA effectively generate behavior labels for a large set of binaries relatively quickly compared to hand-labeling. Although these labels are not as accurate as hand labels produced by a reverse engineer during analysis, these labels could potentially improve the performance of malware behavior classification models such as MalConv by increasing the size of the training dataset, especially when combined with hand labels. Additionally, automatic executable labeling provides behavior labels that are specific to each executable, rather than labels that are specific to a malware family and then generalized to an executable within that family, as is the case with the Microsoft Malware Classification Challenge dataset.

We made some initial progress towards incorporating the labels produced by running CAPA over the Malpedia dataset into the MalConv behavior classification model. However, we did not finish training MalConv on a dataset labeled with automatic executable labeling tools. Rather, this work spun off into the CAPC Seedling project MAGNI, funded under the VANAHEIMR task.

4. FEATURE SELECTION FOR INFORMATION-SECURITY NATURAL-LANGUAGE PROCESSING TASKS: AUTOMATED LABELING OF ATT&CK TACTICS IN MALWARE THREAT REPORTS[1]

4.1. Introduction

Malware continues to represent a significant threat to many domains, causing trillions of dollars of damage every year. It is estimated that 2.9 million dollars is lost every minute from the global economy due to cyber-attacks [66]. A principal problem is that traditional ways of defining malware are very limited in scope. Up until recently, antivirus companies like Comodo, Kaspersky, Kingsoft, and Symantec have provided software to detect malware that relies on signature-based methods [137]. These signatures are short sequences of bytes that are unique to each malware executable, which produces brittle malware detectors with little capability of classifying new malware files. However, by the time they identify the new signature as malicious, malware generally has a large infection rate and often has already morphed into a new variant (polymorphic malware). The average length of time it takes to identify the new malware samples with traditional detection techniques is around 54 days, and it is estimated around 15% of malware is still undetected after 180 days [137].

In order to fix the above problem, a concerted effort by the research community has resorted to using machine learning (ML) methods to detect malware. These methods follow a process of feature extraction of the malware executable, followed by classification or clustering of the malware samples [137]. Feature extraction of malware involves using static analysis, dynamic analysis, or some hybrid of the two. Static analysis involves extracting features without actually executing the malware, generally from the malware binaries. Dynamic analysis involves actually running the malware and analyzing what it does while it runs, generally through the use of a debugger while the malware is contained in some sort of controlled simulation environment. Ye et al. provide a survey of ML results and show that many ML methods achieve classification rates well above 90% for identifying malware samples. However, while the ML methods do quite well on samples of malware they have already learned, these models are often easily evaded in practice with a variety of obfuscation techniques. Further complicating the issue of detecting malware with ML is adversarial learning. Adversarial learning allows attackers to perturb malware samples enough so that when they are inputted into machine learning networks, the networks are fooled into labeling them as benign [25]. Many of these adversarial samples are examples of

[1]Citation of work: Eva Domschot, Ramyaa Ramyaa, and Michael R. Smith. Feature Selection for Information-Security Natural-Language Processing Tasks: Automated Labeling of ATT&CK Tactics in Malware Threat Reports, in submission 2022

polymorphic malware, and it is not well understood why these networks are so easily misled [25].

Recent work has began to identify new ways in which zero-day malware can be identified without relying on signatures, instead working to identify malicious behaviors [115]. Here behaviors refer to actions taken by the malware such as DLL injection or process hollowing. Additionally, while behavioral analysis in information security typically refers to dynamic analysis, here we mean the labels that a malware detector predicts and can be used with inputs from static or dynamic analyses. This work focuses on labeling malware samples with behavioral labels through the use of malware threat reports following Smith et al. [115]. Malware threat reports are reports written by malware analysts, giving in-depth descriptions of how different kinds of malware behave. Unfortunately, there is often inconsistency between different authors and organizations in how these reports are written and terms that are used, making the process of labeling malware from them a manual, laborious task. Automating this process would significantly reduce analyst time and aid in labeling malware samples from threat report to executable, giving new ways in which malware could be detected. In 2020, Legoy et al. used a labeled dataset of malware threat reports that had been labeled with MITRE ATT&CK tactics and techniques [62]. As this is a multi-label dataset, each label was given an individual classifier. In order to summarize the results, they reported both the micro and macro averages of the precision, recall, and $F_{.5}$-score, averaging over the results for every label. The macro average is a standard average, where every label is treated equally [126]. In contrast, the micro average weights each sample equally, meaning it is able to capture any imbalance of the distribution between labels, which can be important when working with many labels, all with different distributions of classes. A macro average may overestimate or underestimate the average if some of the labels have many more positive samples than other labels. This differs from other ML problems not dealing with multi-label or multi-class datasets. In summation, the work done by Legoy et al. illustrates the challenges with using natural language programming (NLP) techniques that are not specific to the information security domain, their best model achieving a macro average $F_{.5}$-score of 27.52% for techniques and 59.47% for tactics.

The work in this paper focuses on the task of automating this process of labeling malware threat reports. Our contributions to this task include an examination of word embedding techniques that have commonly been used for NLP tasks, as well as feature selection methods which are not commonly used as a part of the pipeline of NLP related tasks. We demonstrate that word embedding methods that are not domain specific are insufficient for information security tasks, requiring a large amount of data in order to perform well. In response, we offer a novel feature selection approach, which shows improved performance in the labeling of MITRE ATT&CK tactics in threat reports. This work indicates that, although feature selection has not been used as commonly for NLP tasks, it may be useful for information security tasks, where more traditional methods are not able to pick out the relevant domain specific terms. We improve over the work done by Legoy et al., who used non-domain specific NLP methods. Our work shows an $F_{.5}$-score of 65% for the prediction of tactics when using feature selection techniques in order to create a feature set of the most relevant information security terms for each label, compared to 59% from Legoy et al. All of the code used for this project, as well as the lists of low frequency words, used to improve mutual information, can be found at our Git repository [2].

[2]https://gitlab.com/dummylink

4.2. Background

In this section, the background of the main methods utilized in this research will be explained, as well as the MITRE ATT&CK framework and past work done to label malware threat reports.

4.2.1. MITRE ATT&CK

The MITRE ATT&CK framework is a knowledge base of adversary tactics and techniques, taken from observations of attackers and malware [62]. In recent years it has gained prominence, as a way of creating a standardized language around discussing cyber threats. The MITRE ATT&CK framework can be divided into three different different domains, namely Enterprise, Mobile, and ICS. Enterprise describes behaviors on standard IT systems, such as Linux or Windows; Mobile describes behaviors on mobile devices, using operating systems such as Android; and ICS describes behaviors carried out against industrial control system networks. The Enterprise domain, which this work focuses on, can be divided into tactics and techniques. Tactics represent the goal of the attacker or malware sample, and a list of the twelve MITRE ATT&CK tactics is given in Table 4-1. Every tactic has multiple techniques, which are ways in which the tactic can be achieved. In this work, the behaviors align with the techniques. Despite the creation of the MITRE ATT&CK framework, its adoption has yet to be universal.

4.2.2. Related Work

In 2017, Lim et al. [64], created an annotated set of malware threat reports by hand, using the Malware Attribute Enumeration and Characterization (MAEC) vocabulary, a system defining malware behaviors, malware capabilities, malware families, malware instances, and malware collections. In the end, they created a labeled dataset of 39 malware threat reports by mapping malware actions and behaviors to MAEC labels. They then used a support vector machine (SVM) and naïve Bayes to predict these MAEC attribute labels, achieving an F1-score of 40% for defining malware capabilities, which were the most successfully predicted. Malware capabilities most closely align with MITRE ATT&CK tactics, with equivalent labels such as exfiltration and privilege escalation.

Then, in 2020, Legoy et al. [62], attempted to automate the prediction of MITRE ATT&CK tactics and techniques on a dataset of malware threat reports . Their work differed from the work of Lim et al. who annotated malware threat reports by hand. Instead, they used MITRE ATT&CK tactics and techniques to create a labelled dataset. To do this, they used the references to threat reports that MITRE ATT&CK has for each tactic and technique. Their best performing method was Linear SVC using term frequency-inverse document frequency (TF-IDF), a bag-of-words (BOW) representation that ensures that terms which appear across the majority of documents have a lower weighted representation. For the prediction of tactics, they obtained a macro average precision score of 60.26%, a macro average recall score of 58.5%, and an $F_{.5}$-score of 59.47%.

4.2.3. *Feature selection*

Feature selection has often been applied to text categorization as a way of improving performance, as well as the accuracy of the baseline classifier [140]. Text often contains many words that are distractors or do not add any information within a dataset, reducing overall accuracy of the classifier. As a result, feature selection methods are used in order to find the best set of words to use as features that convey the most amount of information and remove distractor words. The two feature selection methods we used for this research, mutual information and the Chi-squared statistic, are discussed below. For a broader discussion of possible NLP feature selection, see, for example, the survey presented by Kumbhar and Mali [57].

4.2.3.1. Mutual Information (MI)

Mutual information (I) measures the mutual dependence between two random variables and indicates how much one can determine from knowing the state of the other random variable. Mutual information can be defined as shown in Equation 4.1, where t represents the term and c represents the category [134].

$$I(t,c) = log_2 \frac{P(t,c)}{P(t)P(c)} \tag{4.1}$$

Mutual information compares the joint probability of t and c occurring together with the probabilities of t and c occurring independently. When t and c are dependent, $P(t,c)$ should have a much higher probability than $P(t)P(c)$, and $I(t,c)$ should be greater than zero [134]. When independent, $I(t,c)$ is equal to zero. Mutual information has been shown to have a bias towards terms with a low frequency [136].

Mutual information has been shown to aid in feature selection with NLP tasks. Mutual information was combined with a Bayesian algorithm to classify the topics of news documents, showing much improved results [78]. In addition, mutual information has been used for information security tasks. In the work by Amiri et al., they used mutual information to reduce the feature set for intrusion detection on network information [2]. Mutual information did well on this task where intrusions are anomalous behaviors, perhaps due to mutual information picking out rare features. This is similar to the problem being focused on in this research where many labels are small portions of the text.

4.2.3.2. Chi-square Statistic (CHI)

The chi-square metric measures the independence between two events using statistical testing. The equation for the metric can be seen below in Equation 4.2 where N is the number of documents, t is the term, c_i represents the i^{th}category, $\bar{t}$ represents the absence of the term, and $\bar{c_i}$ represents non-membership of the category [140].

$$\chi^2(t,c_i) = \frac{N[P(t,c_i)P(\bar{t},\bar{c_i}) - P(t,\bar{c_i})P(\bar{t},c_i)]^2}{P(t)P(\bar{t})P(c_i)(\bar{c_i})} \tag{4.2}$$

Chi-squared is a normalized metric, which can be compared across features of the same category [136]. However, it is known that this normalization does not work well for low-frequency terms in the feature set, unlike mutual information, which places emphasis on rare terms [136]. Chi-squared has been proven to do very well for text classification tasks, often outperforming other feature selection metrics [99].

4.2.3.3. Feature Selection for Imbalanced Datasets

Feature selection has been shown in the past to achieve better results when used on imbalanced text datasets [140], similar to the dataset utilized for this research. Instead of attempting to balance the training data, feature selection metrics are used in order to pick the most useful words out of the text, obtaining the optimal set of negative and positive features. Zheng et al. [140] showed that using feature selection that only focuses on the presence of the positive class does not achieve great benchmarks against not using feature selection. However, taking the absence of the class into consideration does have a noticeable impact in terms of the result. In addition, they note that two-sided feature selection methods such as chi-squared are not as good at this task as they seem. Two-sided feature selection methods are those which take both positive and negative features into account. In contrast, one-sided feature selection methods only take the positive features into account, meaning the classifier only has features that represent the presence of a label. These methods tend to be biased towards terms correlated with the positive label, especially when the dataset is imbalanced because the positive features occur less in the dataset and are usually scored more highly due to that fact. Instead, they found that one-sided feature selection methods could achieve better results if the best terms were picked first from the positive feature set and next, from the negative feature set. Our work differs in that no one-sided metrics were used. However, the weaknesses that this work exposes in two-sided metrics do aid in understanding the work that was done to improve mutual information.

4.2.4. Word Embeddings

Word embeddings, which map terms as vectors, are one of the key breakthroughs for NLP because they allow words with similar meanings to have similar representations. This is different from the traditional BOW models, which do not have a good way of showing that words may be related. There are several different kinds of word embeddings which include continuous bag-of-words (CBOW), skip-gram, and GloVe [85]. We focus on GloVe, which has achieved state of the art performance in many NLP tasks [10, 85].

GloVe produces term embeddings using a deep neural network based on matrix factorization techniques and is created through a least squares model trained on global word co-occurrence counts [85]. This differs from a skip-gram model, often used to implement Word2Vec, which trains on separate local context windows. For this reason, skip-gram poorly utilizes the statistics of the corpus. Research has shown that GloVe outperforms other word embeddings on benchmark tasks, such as the word analogy dataset, as well as on named entity recognition (NER) [85]. GloVe embeddings are often used as the embedding layer of long short-term memory (LSTM) algorithms for text classification tasks because LSTM networks are able to capture sequential patterns in text that BOW feature sets are not able to capture [10]. In 2017, Barry showed that an LSTM network using GloVe had better results with sentiment classification than either a BOWs approach or using Word2Vec [10].

4.3. Data

We examine feature selection and the performance of assigning behaviors defined by the MITRE ATT&CK framework to threat reports. The dataset comes from the rcATT repository, produced by Legoy et al. [3]. The data contains 1490 malware threat reports, labeled with MITRE ATT&CK tactics and techniques. While our end goal is to label the malware techniques in threat reports, this work focuses mainly on the ATT&CK tactics, mainly for reasons of computing power because there are 200 techniques that would require feature selection methods to first be used over them, a task that could take several days to finish. However, this work is extendable to the techniques as well, given more compute power. It should also be stated that feature selection only needs to be run once so long as the feature sets are stored, making the issue of processing speed a one-time problem. The threat reports come from a variety of sources, which means they do not have any explicit structure. The data is also multi-label with twelve different labels for the tactics. Furthermore, this dataset is also imbalanced, with some tactics having as few as 75 samples, as shown in Table 4-1. Both class imbalance and multi-label problems create more difficult problems for ML.

For data pre-processing, all punctuation, non-letters, and NLTK English stop words were removed from the text. Additionally, the text was lemmatized, using the NLTK lemmatizer. [65].

4.4. Approach

The next sections discuss how our feature selection and GloVe methodology were set up. All methods were tested using five-fold cross validation. Because this is a multi-label dataset, every text document can have multiple independent tactic labels. This is different from other ML problems where a data point can only have one label. To handle this problem, a separate classifier was trained for every single tactic label when feature selection was used. However, the BiLSTM model we used with a GloVe embedding did not need separate classifiers and instead outputted a one-hot encoding of twelve neurons, one for each tactic.

[3] `https://github.com/vlegoy/rcATT`

Table 4-1. Tactics Breakdown

Tactics	Number of Reports
Credential Access	306
Execution	488
Impact	75
Persistence	590
Privilege Escalation	403
Lateral Movement	343
Defense Evasion	768
Exfiltration	123
Discovery	356
Collection	236
Command and Control	406
Initial Access	202

4.4.1. Feature Selection

Many of the threat reports in the dataset contain upwards of 60,000 words; however, the mention of a MITRE ATT&CK tactic may only be a single sentence out of the entire document, which means most of the text is not useful when trying to predict the correct labels. This is one motivating reason to use feature selection methods to help remove the distracting words from the feature set. First, both mutual information and chi-squared were applied across the training set in order to reduce the features. After using feature selection, a TF-IDF approach was used to transform the dataset. Finally, Linear SVC, the best performing method used by Legoy et al., and Logistic Regression were applied to make predictions on the dataset.

4.4.1.1. Mutual Information

First, mutual information scores were generated for the training set for each individual tactic label i.e. the mutual information scores were first generated for each of the terms in the dataset for each tactic. As discussed above, a mutual information value of zero indicates independence, with higher values indicating dependence. After scoring each term in the dataset with a mutual information score, the terms that had the highest mutual information score were used in order to set the threshold. This is because mutual information was scoring all of the most informative words with the same score (the highest score). Using a lower threshold did not add any valuable features, in terms of making better predictions. Thus, any word, with a mutual information score lower than the terms that had the highest mutual information score, was removed from the feature set.

The results of this method can be seen in Table 4-2 for method MI SVC and MI Reg. It can be noted that when mutual information was applied to Linear SVC (MI SVC) the precision dropped by one percentage point from the baseline L. SVC classifier, and there is a slight improvement in the score of the recall. The same can be seen when mutual information is applied to Logistic

Table 4-2. Feature Selection Results. Mutual information provides the optimal results when considering both precision and recall. GloVe, despite achieving state of the art results in many domains, is not able to do so for information security threat reports.

Method	Macro Avg Prec.	Macro Avg Recall	Macro Avg $F_{.5}$
L. SVC	0.628	0.436	0.577
L. Reg.	0.668	0.266	0.513
MI SVC	0.617	0.457	0.576
MI Reg.	0.565	0.300	0.480
MI2 SVC	**0.677**	**0.574**	**0.654**
MI2 Reg.	0.667	0.399	0.588
CHI SVC	0.667	0.414	0.594
CHI Reg.	0.559	0.242	0.443
GloVe Pre-trained	0.412	0.218	0.350
GloVe Weighted	0.379	**0.685**	0.416
GloVe InfoSec-trained	0.395	**0.698**	0.433

Regression (MI Reg). The precision drops by close to one percentage point from the baseline Logistic Regression classifier (L. Reg.) and improves slightly for recall. These results do not indicate any substantial improvement; however, taking note of the weaknesses of mutual information (described below) indicates areas where the feature selection may be improved upon.

The work by Yang et al. shows that mutual information is heavily biased towards low frequency terms, which can create too much noise in the dataset [136]. In addition, the work by Zheng et al., discusses the weakness of using two-sided metrics for feature selection, which tends to place higher relevance on positive features, especially when used for imbalanced datasets [140]. Mutual information, which takes into consideration both negative and positive features, is a two-sided metric. Positive features refer to features that are highly correlated with the presence of a label in a document (positive class). Negative features are those that represent the absence of a label (negative class). Taking all of this into consideration, low frequency terms that appear in only one, two, or three documents were removed from the feature set for the positive class. There was no benefit from removing low frequency terms for the negative class. In fact, it caused a decrease in both precision and recall, so only low frequency terms for the positive class were removed. After this modification, Logistic Regression and Linear SVC surpassed the scores of the baseline classifiers as seen in Table 4-2 (rows MI2 SVC and MI2 Reg). For Logistic Regression (MI2 Reg), the precision score stays almost the same as the baseline L. Reg. classifier, but recall improves by 13.3 percentage points, a substantial increase. Linear SVC (MI2 SVC) has a precision score of 67.7% and a recall score of 57.4%, beating the L. SVC classifier by 4.9 percentage points and 13.8 percentage points respectively. The fact that both Logistic Regression and Linear SVC improved so much when this method of feature selection was used, further proves that this methodology is picking out the correct terms.

There are multiple possible explanations for why pulling out low frequency words from the positive class helped. One possible explanation is that mutual information is biased towards picking more terms from the positive feature set, which caused excess noise. Removing some of those terms reduced this noise and brought more balance between negative and positive features. Second, it can be noted again that the positive labels may only be a single sentence out of an entire document, and as such, infrequent terms that appear in those documents are likely to cause a lot of noise, which is why the positive class benefits from their removal. In contrast, the negative labels only indicate the lack of the label, and infrequent terms seem to aid in their prediction.

4.4.1.2. Chi-squared

Next, Chi-square scores were generated on the training set for each word in the feature set and for each individual tactic label. A p-value of .05, along with one degree of freedom, was used to obtain the critical value of 3.84 from the chi-squared distribution table. Any word with a chi-squared value greater than 3.84 was assumed to be independent of the label and left out of the feature set. The results are shown in Table 4-2 for rows CHI SVC and CHI Reg. It can be seen that using the chi-square metric improved the precision from the baseline (L. SVC) when Linear SVC was used, but the average recall score dropped by 2.2 percentage points. When Logistic Regression was used, chi-squared does worse on both recall and precision than the baseline Logistic Regression model (L. Reg.). The chi-square metric is also a two-sided metric, similar to mutual information, so it shares the same weakness of prioritizing the positive features. However, removing low frequency words did not aid the chi-square metric. This is likely because chi-square is not biased towards the infrequent terms in the same way that mutual information is.

4.4.2. GloVe Embedding

A GloVe embedding was used with a bidirectional LSTM (BiLSTM) architecture, which is a type of recurrent neural network (RNN). The BiLSTM model is comprised of a GloVe embedding layer, a BiLSTM layer, two dense layers, and a dense layer with sigmoid activation that outputs twelve neurons, one for each tactic. Binary cross entropy was used for the loss. A threshold of 0.5 was set for the predictions. In addition, the batch size was set to 128, and the model was trained for five epochs, based upon trial and error. While, the number of training epochs may seem low, using a higher number of epochs quickly led to over-fitting of the model, showing the model's inability to generalize to the information-security data. We examined several GloVe embeddings, two of which are outlined below.

4.4.2.1. Pre-trained GloVe

First, we used the common crawl GloVe embedding, created by Stanford [85]. This embedding was trained over 42 billion tokens, with a 300 dimensional vector space and was used as the embedding layer of the BiLSTM model. The results can be seen in Table 4-2 for the method pre-trained, which shows scores that are much lower than the baseline Linear SVC (L. SVC) method.

In order to further aid the BiLSTM model, a weighted binary cross entropy function was created (Weighted in Table 4-2). Class weights were first assigned for the positive and negative classes for each tactic. The weighted loss function returns the mean of the weight matrices for each class multiplied together with binary cross entropy. The average recall is almost 70%, surpassing all methods used so far. The precision, on the other hand, dips by 3.3 percentage points from the unweighted BiLSTM model, and the precision is overall much lower than any of the feature selection methods using Linear SVC or Logistic Regression. This indicates that many false positives are returned.

4.4.2.2. Information-security Trained GloVe

As can be seen from the results using a pre-trained GloVe embedding, the overall precision is low for both methods. This may be due in part to the fact that the dataset is full of highly specific information-security words that the GloVe embedding was never trained over. In order to test this theory, tools from the GloVe repository were used to create an information-security specific GloVe embedding. To do so, we downloaded a corpus that contains the first 100 million characters from Wikipedia, which is then supplemented with domain specific texts, in order to create a GloVe embedding. Since training a GloVe embedding requires large amounts of data, we had to supplement the information-security documents with those from Wikipedia. In all, a training set of 211,000 information-security documents was collected for training a GloVe embedding. This included 503 malware threat reports downloaded from the Github repository APTnotes [11], around 42,000 entries extracted from Exploit Database [109], around 165,000 CVE entries from MITRE [19], around 1400 cybersecurity blog posts extracted from Krebs on Security [56], and the 1490 reports from the examined dataset. All of the text documents were tokenized and then used to create a GloVe embedding. The results from using this embedding with the weighted loss function are shown in Table 4-2. It can be noted that there is a small improvement in results from the weighted LSTM model.

4.4.2.3. Evaluation

Overall, using an BiLSTM with a GloVe embedding scored highest on the overall recall score, with the GloVe embedding trained on the cybersecurity corpus scoring the highest with a recall score of 69.8%.

However, precision did not go above 50% for any of the three GloVe embedding methods. Overall, mutual information, with low frequency words pulled out for the positive class, did the best, with both recall and precision scoring above 50%. It also has the highest $F_{.5}$-score of 65.4%.

While GloVe has often done better on text classification than TF-IDF methods [10], it seems that in this case a word embedding did not sufficiently represent a highly domain specific dataset. While, we also tested a GloVe embedding trained over a information-security corpus, it also performed poorly. One possibility is that it was not trained over a large enough corpus, indicating that many more documents would be necessary to improve the result. In addition, many of these labels might only be represented by a few words out of an entire text document. This is different from other text classification tasks where GloVe has been proven to do well, such as sentiment analysis or classifying the topic of a document [10]. In those scenarios, often most of the text is relevant to the label being predicted. This in part helps to explain why feature selection would aid in classification so much because it removes the noise from the irrelevant portions of text.

4.5. Class Imbalance Methods

After selecting mutual information along with Linear SVC as the best performing method, further work was done to increase the performance of the classifier. This section focuses on different methods of correcting the imbalance in the data. Previously, sampling techniques have often been applied as a method of correcting the imbalance within the data [131]. Oversampling is used to create more samples of the minority class by duplicating the minority class; however, it is prone to over-fitting on the data. In contrast, undersampling removes samples of the majority class until there is an even distribution of samples. The work by Legoy et al. on the same dataset used resampling of the minority class in an attempt to improve results, but their work did not show any substantial improvement for recall or precision. Instead, we examine two different methods of correcting data imbalance.

4.5.1. SMOTE: Synthetic Oversampling

SMOTE is a method of oversampling the minority class by creating synthetic data samples [15]. The synthetic samples are created from the feature space by interpolating between *k*-nearest neighbors of the same class. For this work, SMOTE was applied to the training set with 3-nearest neighbors after feature selection had been used to pick the optimal set of features. Finally, Linear SVC was used to make predictions on the test set.

4.5.2. Snorkel: Weak Supervision

Another method to address data imbalance is to label more data. However, this is often a time consuming task because most labeling has to be done by experts who go through the text documents manually. Snorkel is an implementation of weak supervision where users can generate labeled training data by using labeling functions, which apply well known heuristics and rules to the task [98]. The labeling functions may be overlapping and have noise. Snorkel then uses an ML model to predict the best set of labels for unlabeled text documents [98].

Here, 503 unlabeled malware threat reports were downloaded from the Github repository APTnotes [11]. In order to create the labeling functions, heuristics were needed for each tactic. Snorkel seems to work best when there are labeling functions for both the positive and negative classes. This was a challenging problem given that the negative class represented the absence of a tactic. However, there is the possibility that the absence of a tactic makes other malware tactics and behaviors more likely to show up in the text. Using that assumption, the training data was used in order to find high frequency words that mainly show up for the positive label or mainly show up for the negative label. These lists of words were then used to create the labeling functions.

Once the labeling functions were created, the unlabeled set of documents from APTnotes was labeled using the Snorkel model. The Snorkel predictor will either predict one of the labels (positive or negative in this case), or it abstains from giving a prediction when the model is uncertain or there isn't a relevant labeling function for a text document. In this case, we made the assumption that abstain predictions could also be seen as an indicator of the absence of a label, so all of them were changed to negative labels. Snorkel was applied after using mutual information on the dataset.

4.5.3. Evaluation

The results of SMOTE and Snorkel are shown in Table 4-3. The table shows the results of using SMOTE and Snorkel without any feature selection, as well as the results of using SMOTE and Snorkel after mutual information has been applied. The Linear SVC baseline model is also shown here.

It can be seen that SMOTE without any feature selection greatly improves the average recall score from the baseline classifier, with an improvement of 14.5 percentage points. On the other hand, the precision dropped by two percentage points. SMOTE, with mutual information, improved even further, gaining twenty-one percentage points in recall over the baseline classifier, with precision only being one percentage point less than the baseline. This affirms that mutual information is finding a more optimal feature set, which allows SMOTE to create synthetic samples that have far less noise. Additionally, it can be seen that SMOTE with mutual information outperforms the recall of just using mutual information by eight percentage points, although it does worse in terms of precision.

Snorkel, on the other hand, only gains 3.5 percentage points in terms of precision against the baseline classifier, and the recall score drops over ten points to 31.7%. Using mutual information alongside Snorkel improves the results, but still does worse than just using mutual information on its own. It is likely that Snorkel could be improved upon if better heuristics were given to the labeling functions. The biggest challenge is in creating the labeling functions that indicate the absence of a tactic.

4.6. Discussion

In summation, the key findings of this research showed that many of the state of the art techniques used in NLP need to be adapted specifically for the information security domain. Our methods provide a 6% gain in the $F_{.5}$-score from the research done by Legoy et al. when mutual information was used as a feature selection tool when predicting tactics. Additionally, a combination of SMOTE and mutual information saw a gain of 2% in precision from their work and a gain of 6.5% in total recall, with both methods achieving higher $F_{.5}$-scores. Of note is the

Table 4-3. Data Balancing Results. Snorkel generally improves the precision while SMOTE increases the recall. Mutual information feature selection (FS) increases both the precision and the recall.

Method	Macro Avg Prec.	Macro Avg Recall	Macro Avg $F_{.5}$
L. SVC	0.628	0.436	0.577
MI2 SVC	**0.677**	**0.574**	**0.654**
SMOTE	0.604	**0.581**	0.599
Snorkel	**0.663**	0.317	0.544
SMOTE FS	0.621	**0.656**	0.628
Snorkel FS	**0.667**	0.417	0.596

fact that mutual information performed so well in this scenario when the infrequent words were removed for the positive label in the training set. Past research has shown that other feature selection methodologies, such as chi-squared, have generally out performed mutual information when used for classification tasks [134, 99]. This work indicates that mutual information can outperform chi-squared when it is pruned of some of the infrequent terms that it is biased towards. This work also showed that traditional feature selection out performed GloVe on an information security specific task, likely because the pre-trained GloVe embeddings have been trained over more generalized text documents.

In the future, this work could benefit from being extended to the MITRE ATT&CK techniques as well, although generating the feature sets for around two-hundred technique labels may be a time consuming task, which is one potential downfall of this approach. However, once generated, the feature sets do not have to be created again unless the training data were to change. Further, it should be understood that the MITRE ATT&CK framework is constantly adapting and combining or adding labels to their system as new behaviors are discovered. While this does not invalidate any of the labels within this dataset as being incorrect, it does give room towards improvement in creating a more dynamic dataset.

4.7. Conclusion

In this paper, we have discussed past research that has been done on labeling malware threat reports with behavioral labels, showing that further pre-processing of the data may be beneficial. This led to our research in using feature selection methodology as a way of creating optimal feature sets to improve overall precision and recall, as well as leading into our work with attempting to balance the dataset with other pre-processing methods such as SMOTE and Snorkel. Additionally, we also compared to current state of the art methods for NLP tasks, such as GloVe. This work showed that feature selection outdid GloVe when working with a very domain specific dataset. Finally, we also showed improvements in the results when using SMOTE to to create synthetic data to fix the data imbalance.

As malware continues to find news ways to evade detection, our purpose is to find new ways in which we can automate the labeling of malware with behaviors. The ability to detect malware by the behaviors it exhibits not only aids in detection but also aids the malware analysts, who now have starting places of behaviors that they can look for. Unfortunately, there are not many datasets of malware that are labeled with these kinds of behaviors. The work from this research aims to help with this task by giving a means to utilize all of the prior information that has already been written about malware in order to create these kinds of labeled datasets. In addition, this research aims to aid analysts by collating important information for them.

5. GENERATING NOVEL MALWARE WITH SPECIFIC BEHAVIORS[1]

Recent generative techniques are capable of generating images with specific visual characteristics by conditioning the generative process with labels representing those characteristics. To borrow some of the techniques from the image domain for our initial experiments, we use the binary representation of an executable from the Microsoft malware classification challenge (which has some of the PE header stripped off so that it is not executable). We follow the process of transforming the executable into a gray-scale image for direct consumption by a ML algorithm [74] (which also demonstrated the success of detecting malware with image representations of the executable).

We examine the conditional subspace variational auto encoder (CSVAE) [55] and a flow based generative model (GLOW) [54] for generating malware with specific behaviors. We chose the CSVAE and GLOW models empirically, as they create images with specific visual features and are relatively simpler to train than more complicated models. The CSVAE was developed to provide a neural network architecture that incorporates conditional information into the VAE, both during training and also for subsequent use as a generative model. GLOW has similar functionalities, but theoretically it can provide exact latent-variable inference without approximation based on a sequence of bijective functions. A schematic diagram of the CSVAE used in our research is shown in Figure 5-1 and GLOW is shown in Figure 5-2.

Despite the impressive performance by generative adversarial networks (GANs) in generating synthetic data, we specifically chose to not examine GANs due to the large amounts of data required to train them [51]. Various methods have been proposed to reduce the number of training images for GANs and conditional GANs [67, 51, 139, 125] that we plan to investigate as part of our future work.

We used the CSVAE[2] and Glow [3] author's GitHub code as well other GitHub sites[4,5,6] for reference and inspiration in designing and implementing our CSVAE and GLOW models. We extended this CSVAE code base to allow for subspace reconstruction as a separate path along with image reconstruction. In Figure 5-1, this is shown as the sub network below which takes in a

[1]Citation of majority of this work: Michael R. Smith, Stephen J. Verzi, Nicholas T. Johnson, Xin Zhou, Kanad Khanna, Sophie Quynn, and Raga Krishnakumar. Malware Generation with Specific Behaviors to Improve Machine Learning-based Detection. In 2021 IEEE International Conference on Big Data (Big Data), pp. 2160-2169. IEEE, 2021.

[2]`https://paperswithcode.com/paper/learning-latent-subspaces-in-variational`

[3]`https://github.com/openai/glow`

[4]`https://github.com/alexlyzhov/latent-subspaces`

[5]`https://github.com/AntixK/PyTorch-VAE`

[6]`https://github.com/y0ast/Glow-PyTorch`

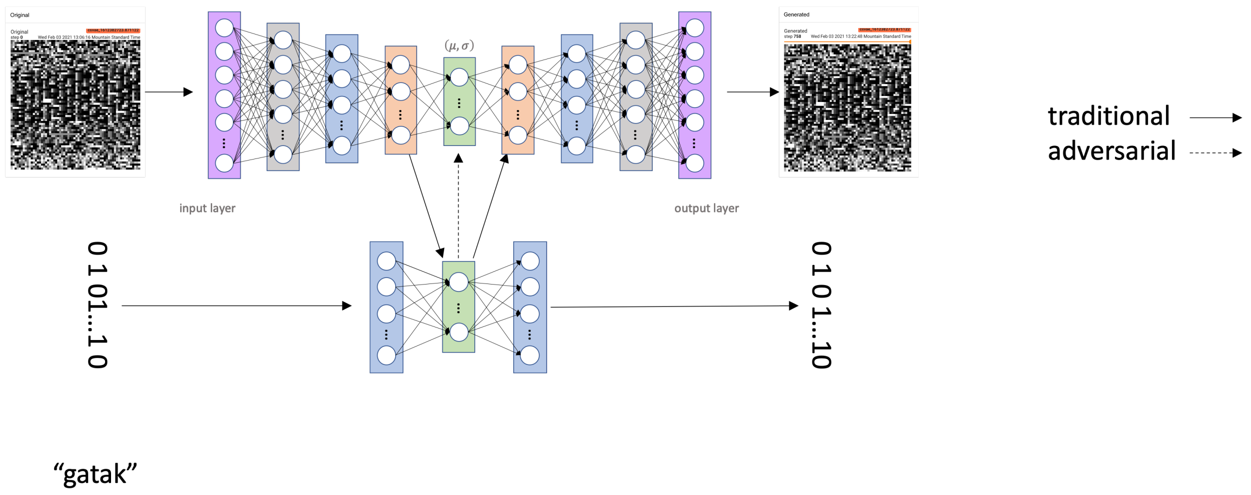


Figure 5-1. CSVAE neural network architecture used in our research. This architecture includes four convolutional sub-layers in both the encoder and decoder portions of the VAE as well as two latent layers, one each for the malware images and behaviors.

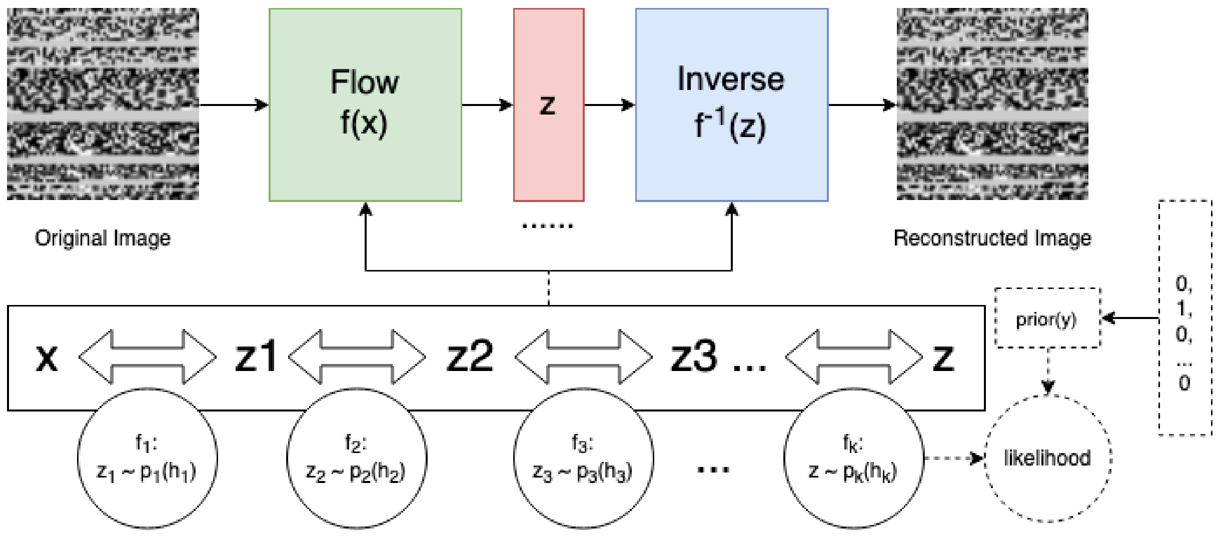


Figure 5-2. GLOW generative model used in our research. This architecture uses the label likelihood to condition the latent spaces on the behaviors.

vector of 0's and 1's corresponding to the behaviors present in an executable. Once we have trained our CSVAE and GLOW models on the malware images and behaviors [115], we use the trained model to generate malware samples with specified behaviors.

We start our investigation using only 64x64 images of the malware to establish the feasibility of using synthetic data. We only use the first 64x64 bytes and truncate the rest of the executable—recognizing that with a limited portion of the executable the behavior may or may not be present. In the generation process, the set of behaviors present is used to condition the generation process. With only 7 distinct families (combinations of behaviors) disentangling any correlations between the behaviors is also a concern. The CSVAE is beneficial here since part of its design and loss function help in disentangling behaviors from each other and from the malware images. (GLOW does not have the same mechanism and is apparent in the results).

In our CSVAE implementation, each convolutional sublayer consists of a two-dimensional convolutional neural network (CNN) layer with 5x5 kernels, a stride of 2 and padding of 2. Initially we looked at using both batch normalization and pooling, but we settled on using just batch normalization without pooling. In visual inspection of the results, pooling produced images that were significantly blurrier than not pooling. Since each byte can convey important information, we opted not to use pooling. We use leaky ReLU activation in the convolution layers. We added two additional weighting factors (β_2 and β_4 in Equation 5.1) to the first two parts of the loss function specified by Klys and colleagues [55] (see Section 4.1 of [55] for more details) so that we could separate the contribution of each latent layer's KL divergence to the composite loss function. The third part of the loss function remains the same, now using β_5 in Equation 5.2 in this manuscript.

$$\begin{aligned}\min_{\theta,\phi,\gamma}\{\mathbb{E}_{\mathscr{D}(x,y)}[\beta_1\mathbb{E}_{q_\phi(z,w|x,y)}[\log(p_\theta(x|w,z))]+\\ \beta_2 D_{KL}(q_\phi(w|x,y)\|p_\gamma(w|y))+\\ \beta_3 D_{KL}(q_\phi(z|x,y)\|p(z))-\log(p(y))]\\ \beta_4\mathbb{E}_{q_\phi(z|x)\mathscr{D}(x)}[q_\delta(y|z)\log(q_\delta(y|z))]\}\end{aligned} \tag{5.1}$$

$$\max_{\delta}\{\beta_5\mathbb{E}_{q_\phi(z|x)\mathscr{D}(x,y)}[q_\delta(y|z)]\} \tag{5.2}$$

Here, $\log(p_\theta(x|w,z))$ is the marginal log-likelihood for reconstruction of the data x conditioned on the latent space z and a subspace w. The latent subspace w is conditioned on the label vector y (in our case, the set of behaviors) as $\log(p_\gamma(w|y))$. D_{KL} represents the KL divergence, and q_ϕ represents a learned approximate posterior. The mutual information between y and z is minimized by maximizing $q_\delta(y|z)$ in Equation 5.2, since $I(Y;Z)=H(Y)-H(Y|Z)$, where $H(Y)$ is fixed. In clearer terms, β_1 weights the generative loss – how different is the model image output from the model image input; β_2 and β_3 weight the conditional latent losses for w and z, respectively – penalizing q_ϕ for deviating from the desired distribution (e.g., $\mathscr{N}(\mu,\sigma)$); β_4 weights the conditional entropy of $y|z$ subject to the maximum computed in Equation 5.2; and β_5 weights the conditional entropy between y and z. During learning we use mean squared error to compute reconstruction error for the image (x in the first term in Equation 5.1) and binary cross entropy for

Table 5-1. CSVAE parameters and values.

Parameter	Value
CNN kernel size	5x5
CNN activation	leaky ReLU
learning rate	0.001
batch size	256
training epochs	1000
β_1	20
β_2	0.00001
β_3	0.00002
β_4	10
β_5	1

behavior reconstruction (y in Equation 5.2). We also use Pytorch's distributions to compute KL divergence [83]. A table of parameters and values used in our CSVAE implementation is provided in Table 5-1.

GLOW is one type of flow-based generative model which takes high-dimensional vector x (binary image) as input then compressing it into latent space z with a sequence of transformations based on the bijective functions. In the Glow implementation, the authors propose the use of batch normalization to alleviate the problems encountered in training steps and an invertible 1 x 1 convolution as a generalization of permutations operation instead of using random or reverse permutation before transformation step. A reversible transformation is the affine coupling layer which was initially introduced in NICE [27] that minimizes the negative log-likelihood l as the following:

$$l(\mathscr{D}) \approx \frac{1}{N}\sum_{i=1}^{N} -\log p_\theta(\tilde{x}^{(i)}) + c \tag{5.3}$$

where $\tilde{x}^{(i)} = x^{(i)} + u$ represents the i^{th} instance with Gaussian noise $u \sim \mu(0,a)$, and $c = -M * \log a$ where a is determined by the discretization level of the data and M is the dimensionality of x.

In our GLOW implementation, we do not modify the code as significantly as we did with the CSVAE. We set the learning rate to 0.001 and train for 240 epochs using a batch size of 48. The parameters were adjusted as GLOW had larger memory constraints than the CSVAE.

5.1. Experiments

Here we describe the specifics that we use to run our experiments for generating synthetic data and for detecting behaviors in malware. To detect behaviors in an executable, we fine-tune the Malconv model, a deep neural network model that achieves state-of-the-art malware detection (benign or malicious) [93]. We use the pre-trained Malconv model [93] trained to detect malware on the EMBER dataset [4] and fine tune the weights of the last layer for behavior detection. To simulate a novel malware family, the Malconv model is trained by holding out one malware family and training on the remaining malware families. The held-out malware family will then represent a novel malware attack.

Malconv enhances the state-of-the-art by allowing extremely long input sequences, employing a shallow 1-dimensional convolutional network with 128 convolutional filters (covering 500 bytes each), max pooling, and a long stride of 500 bytes. The wide filters and aggressive stride allow Malconv to process very long inputs in a reasonable amount of time. Malconv also borrows many ideas from natural language processing. For instance, the model uses an embedding layer to deal with the polysemy inherent in byte values, as well as gated convolutions [24]. The model does not use batch normalization, which typically performs poorly on the type of sequential, polysemous inputs found in both natural language and executables; instead, Malconv uses DeCov regularization [17].

The fine-tuning is performed by re-initializing the final fully connected layer in the model and modifying it to output the requisite number of classes for our task. Similar to the original Malconv training, our batching process uses zero-padding to convert all inputs to the length of the longest input in the batch.

We use this model for all of our experiments and only change whether synthetic data is used as part of the fine-tuning phase. We use synthetic data from the CSVAE and GLOW so that in there are 3,500 samples for every family. We also compare our method against SMOTE [15] which, at a high-level, generates synthetic data by interpolating between data points of the same class and has performed well in many tasks. We used the malware families as the classes and not the behaviors as interpolating in the multi-label scenario is not handled by SMOTE. As interpolating between images most likely will not maintain semantic information, we generate synthetic data with SMOTE using the outputs of the neurons of the last layer of Malconv. The new Malconv model trained on the SMOTE synthetic data will only have its last layers modified.

5.2. Results

Ultimately, our goal is to improve the detection of novel malware by detecting behaviors that are shared among several malware families without first having to discover, analyze, and collect several samples of a novel malware family. Given the challenging nature of obtaining and detecting behavioral information, we seek to generate novel malware samples. However, when generating synthetic data sets, there are several steps that can be used to help verify the validity of the synthetic data. In the image domain, one can visually inspect the synthetic images to ensure realism and that certain characteristics are present. In the malware domain, that option is available but inconclusive. Therefore, we validate the results using the following techniques:

1. *Visual Inspection.* For images, one can examine generated images and determine their quality. We set up a similar examination for validating the images and examine visually how synthetic data compares to ground truth data. Our hope is that visual characteristics will be replicated as a sanity check, but by no means any conclusions can be drawn from the visualizations.

2. *Reconstruction Error.* Here, as a first step, we ensure that the reconstruction error for the images *and* the behaviors are decreasing as training progresses. While most traditional generative methods only focus on the loss from the generated image, we also consider the reconstruction on the behaviors as another indicator of how well we are able to capture the characteristics of the behaviors. While lower error is always better, determining how low is acceptable is an open question.
3. *Behavior Recognition.* We validate that behaviors are in the synthetic data based on the improvement of behavioral detection using the fine-tuned Malconv model described in Section 5.1. We specifically examine the recall rate of detecting behaviors that are present and the specificity of not predicting behaviors that are absent.

Our results for each validation method are presented below.

5.2.1. Visual Inspection

A batch of input images of malware (left) and synthesized images of malware (right) from the CSVAE are shown in Figure 5-3 and from GLOW are shown in Figure 5-4. The number of images are different since the batch sizes for GLOW and the CSVAE differed as GLOW required significantly more resources to compute. As a first pass in validating the output of a generative model, we can visually discern many of the patterns from the ground truth data. The synthesized data is significantly smoother than the actual data and some of the more subtle patterns within the examples are not present in the synthesized data. Some of the structure in the data is captured. For example, the in the bottom row in the fourth and seventh columns in Figure 5-3, there is very similar structure reconstructed. In other data points, the structure is not captured quite as well. In the first row, there are several examples where what looks like salt and pepper noise is smoothed in the reconstructed examples. Similarly, GLOW is able to capture the overall structure of images with some visual defects. Visually, the CSVAE appears to regenerate crisper structural details.

We conclude that we are able to capture some structure - the question is whether the models are able to capture the structure that corresponds to behaviors, and the less meaningful structure is not captured as precisely. That will be investigated in the next two sections dealing with behavior reconstruction and detection.

5.2.2. Reconstruction Error

The reconstruction error is simply the mean-squared error between the original values and the values on the reconstructed image. However, we also measure the reconstruction error on the recovered behaviors from the latent space. The behaviors are provided as input to the CSVAE to condition the latent space for the generation of synthetic data and are also reconstructed as shown in Figure 5-1. Behaviors used here are not themselves conditioned, but rather serve to condition the image reconstruction. We reconstruct behaviors by decoding from the latent subspace layer used to encode the behaviors, which is trained using adversarial weighted links (see [55]). The behavioral reconstruction error is also the mean squared error between the original behaviors and

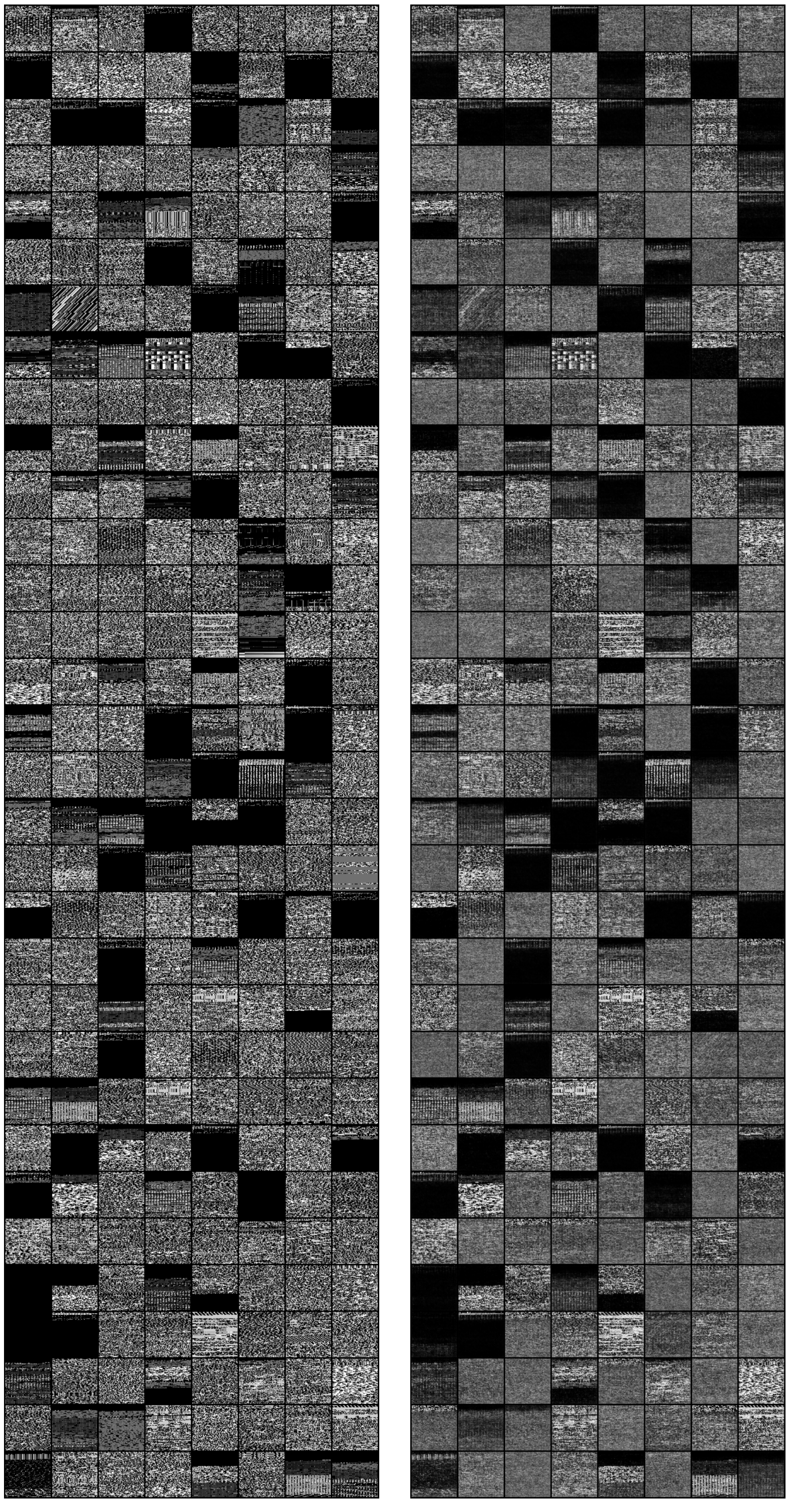
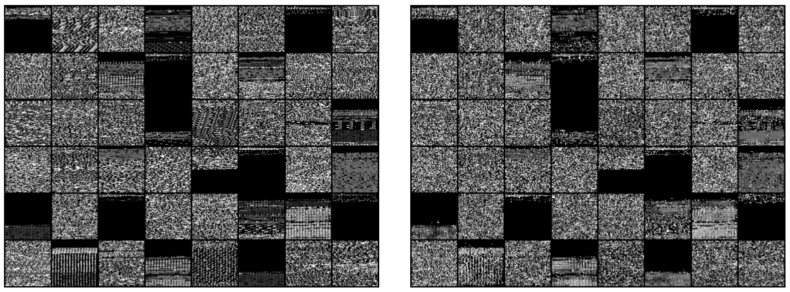
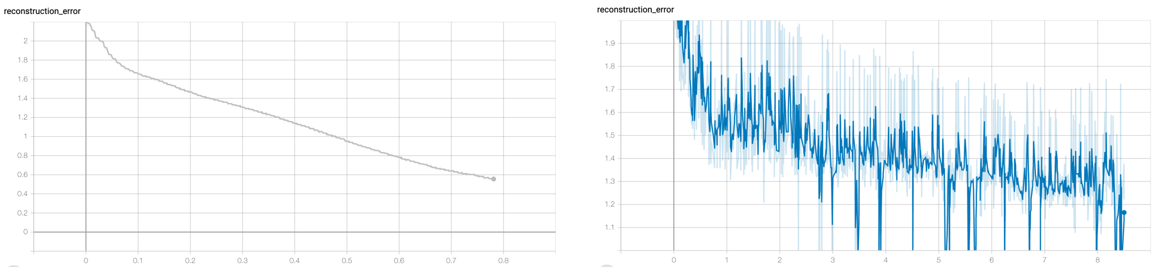

(a) (b)

Figure 5-3. (a) Example batch input data to the CSVAE model. (b) Corresponding synthetic data from the CSVAE model. Visually, many patterns from the input data on the left are present in the synthetic data on the right suggesting that the CSVAE is able to capture some nuances of the data.



(a) (b)

Figure 5-4. (a) Example batch input data to GLOW. (b) Corresponding synthetic data from GLOW. Visually, many patterns from the input data on the left are present in the synthetic data on the right suggesting that GLOW is able to capture some nuances of the data.



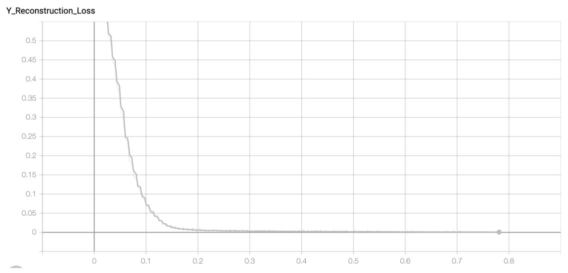
(a) (b)

Figure 5-5. Reconstruction error on a reconstructing malware images by the CSVAE. (a) TOP: Reconstruction error on a single batch of inputs is significantly smoother as training progresses. (b) BOTTOM: Reconstruction error when the batches are shuffled oscillates between batches. This difference shows a difficulty to learn the overall structure of the executable that generalizes between batches.

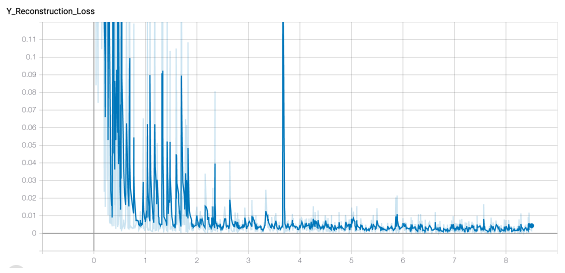
the reconstructed behaviors. We focus specifically on the CSVAE as the GLOW model provides similar results.

We consider training with increasing levels of difficulty that highlight the challenges in generating images of malware. We first train on a single batch where the data is the same for all training steps. In this case, as shown in Figure 5-5a, the training error smoothly decreases as the amount of time spent training increases. Here, the x-axis represents that amount of time in training and can roughly be correlated with the number of epochs. We then train using multiple batches and shuffling between each batch (Figure 5-5b). In this case, we trained for a significantly longer period of time and observe that (1) the reconstruction error is also significantly larger, and (2) while there is some decrease, the error oscillates considerably as training time increases—corresponding with a new shuffled batch of inputs to train on. This highlights a challenge in learning the structure of the executable. This is different from many of the natural image tasks where this is not as difficult. We are currently trying to understand the differences and how to reduce the oscillations.

Despite the challenges in producing low reconstruction error, the behavioral information is what



(a)



(b)

Figure 5-6. Reconstruction loss for reconstructing malware behaviors using the CSVAE. (a) TOP: Behavior reconstruction loss on a single batch of inputs is significantly smoother as training progresses. (b) BOTTOM: Behavior reconstruction loss from shuffled batches. The reconstruction error oscillates at the start, but then settles down, with some intermitent spikes, as training progresses. Since we are using one-hot encoding for the behaviors, it is not surprising that the behavior reconstruction loss approaches zero as training progresses.

Table 5-2. Behavioral reconstruction error when seeding the CSAE with a training image (scenario a) and when only sampling from the latent space (scenario b). Sampling directly from the latent space and using novel combinations of behaviors highlights challenges in disentangling correlations between behaviors and capturing generic structure of the executable.

Behavior Reconstruction Error	
(a) Training image + known behavior combinations	0.07
(b) Latent space sampling + known behavior combinations	29.1
(b) Latent space sampling + novel behavior combinations	43.0
(a) Training image + novel behavior combinations	43.2

is most important to maintain. Behavior reconstruction loss is shown in Figure 5-6 for both a single batch and shuffled multi-batch training, where the x-axis ranges over the amount of time (in hours) since the start of training. As with image reconstruction, shuffling the batches produces significantly larger reconstruction error (see the y-axis in Figure 5-6) and produces oscillations. Although, for behavior reconstruction, there are much better convergence properties than the image reconstruction.

Using the single non-shuffled batch, we also examine specific expanded use cases of reconstructing the behaviors shown in Table 5-2. While the absolute values do not have a strong meaning (as we are unaware of any other work examining this), their relative values give important information. When using known behavior combinations (from known malware families) and seeding the latent space using training images, the error is set at a baseline of 0.07. Once the data with known combinations of behaviors is generated by sampling directly from the latent space there is a significant increase in behavioral reconstruction error by an order of magnitude. Thus, there is an acute challenge in how to properly sample from the latent space to ensure that structure that is shared between executables is properly generated. There is another significant increase in behavioral reconstruction error when the novel combinations of behaviors

Table 5-3. The recall, specificity and accuracy for detecting behaviors with (CSVAE, GLOW, and SMOTE) and without (Malconv) synthetic data.

	Malconv			CSVAE			
	recall	specificity	acc	recall	specificity	acc	support
simda	0.250	**1.000**	0.873	0.393	**1.000**	**0.899**	42
vundo	0.315	**0.985**	0.893	**0.546**	0.925	0.861	475
tracur	0.401	0.938	0.879	0.600	**0.983**	**0.939**	751
gatak	0.159	**0.985**	0.771	0.209	0.970	**0.776**	1013
ramnit	0.146	0.978	0.737	0.199	0.970	0.752	1541
lollipop	0.530	0.876	0.857	0.637	**0.939**	**0.901**	2478
kelihos	0.456	0.899	0.870	**0.529**	**0.924**	**0.887**	3340
Ave	0.322	0.952	0.840	0.445	**0.959**	**0.859**	
Std.Dev	0.147	0.048	0.061	0.181	0.030	0.069	
	GLOW			SMOTE			
	recall	specificity	acc	recall	specificity	acc	support
simda	**0.531**	0.871	0.795	0.321	1.000	0.889	42
vundo	0.432	0.814	0.754	0.409	0.979	**0.898**	475
tracur	**0.654**	0.834	0.804	0.591	0.941	0.898	751
gatak	**0.404**	0.824	0.699	0.242	0.943	0.764	1013
ramnit	**0.396**	0.742	0.629	0.242	**0.985**	**0.77**9	1541
lollipop	**0.736**	0.807	0.799	0.669	0.832	0.799	2478
kelihos	0.494	0.742	0.725	0.477	0.878	0.843	3340
Ave	**0.521**	0.805	0.743	0.422	0.937	0.839	
Std.Dev	0.130	0.048	0.064	0.167	0.061	0.058	

are produced, highlighting a possible challenge of disentangling the correlations between behaviors. This occurs regardless of whether the latent space is sampled from or seeded from a training image.

5.2.3. Behavior Recognition

The ultimate goal is to be able to *detect* behaviors in novel malware. We examine this, comparing behavior detection using our fine-tuned Malconv model (Section 5.1). We fine-tune using our original data and data that has been augmented using the CSVAE, GLOW, and SMOTE models. Since we are holding out one family, all of the test data has the same label set. Thus, we report the recall (behaviors correctly detected), the specificity (behaviors correctly identified as not being present), and the averaged accuracy across all behaviors in Table 5-3.

The results show that behavior detection is a challenging problem. The baseline recall of behaviors is only 32.2% despite Malconv being able to detect malware effectively at over 94% detection rate [93, 4] and classify malware families at 98% using the same Microsoft malware classification challenge data that is examined here [47]. The low recall and high specificity reflect that the model has a high number of false negatives but fewer false positives.

The addition of synthetic data from the CSVAE, GLOW, and SMOTE results in a significant increase in recall, improving by 12%, 20%, and 10% respectively. SMOTE, on average, has a slight decrease in specificity; the CSVAE performs the best with a significant increases in recall and a slight increase in specificity; GLOW achieves the highest recall, but it has the lowest specificity and accuracy. This highlights that the synthetic data is able to capture the characteristics of some behaviors as the recall significantly increases. The increase in specificity from the CSVAE also highlights that the CSVAE is able to disentangle the behaviors to a certain degree while SMOTE nor GLOW is not able to do so as efficiently (nor are they designed to do so). Future work is needed to make conclusive statements about the entanglement of the behaviors, but initial results are promising in the ability to generate synthetic data that captures meaningful features for behavior detection.

6. MITIGATING THE IMPACT OF NOVEL MALWARE BY DETECTING OUT-OF-DISTRIBUTION SAMPLES[1]

6.1. Introduction

Malware infections continue to increase at exponential rates and show no signs of declining. The amount of malware variants and novel families makes it difficult for analysts to manually process and defend against them, necessitating automated mechanisms to assist. With the increasing variations of malware, machine learning (ML) has been explored as a means of mitigating the effects of malware [93, 115]. Many proposed methods report impressive results—achieving upwards of 99% accuracy and F1 scores in detecting malware from various data sets [129, 49, 128, 35]. Despite these impressive experimental results, ML-based malware detection does not perform as well in deployed situations and particularly in specialty domains such as banking or health care where large repositories of malware are not available [84, 115, 133].

In the ML community, it has been shown that commonly used *n*-fold cross-validation provides overly optimistic results as spurious correlations with the classes are present across folds but not in deployed scenarios [5]. The rising concern of the inability and the unpredictability of how a ML model generalizes to novel data is now being studied in the ML community [22]. The problem of generalizing to novel data is magnified in domains like malware detection where the environment is highly dynamic and malware authors purposefully obfuscate to evade detection.

We propose that a significant contributing factor to the large discrepancy between observed performance rates and actual performance for ML-based malware detection is significant differences in the data encountered in deployed scenarios from those data used for training. We denote not properly accounting for this assumption as a *generalization bias*. In cases where the data is similar to the training data, ML-based detectors tend to excel. However, as the encountered data differs from the data used to train the model, model performance deteriorates and becomes unpredictable. To evaluate the impact of generalization bias we hold out an entire malware family rather than partition a malware data set randomly or according to time. Random partitioning of the data provides an estimate of the performance on specific data types, and partitioning according to time that a malware sample was observed helps to evaluate data drift and how the model adapts. To facilitate predictions on classes that are not included in the training data, we predict behaviors rather than malware families [115] using the Microsoft Malware Classification Challenge Dataset [100] and corroborate results on the malpedia dataset [89].

[1]Citation of majority of this work: Michael R. Smith, Raga Krishnakumar, Joseph P. Lubars, Stephen J. Verzi, Xin Zhou, and Akul A. Goyal. All Models are Wrong, but Some(times) are Useful: Evaluating when Machine Learning Models are Useful for Detecting Novel Malware in the Wild. Sandia National Labs SAND2022-13867 C. 2022.

Recently, several papers have investigated the discrepancy between experimental and real-world ML results in computer security. Notably, Axelsson [7] recognized the *base rate fallacy* which describes how class imbalance, common in malware detection where goodware often significantly outnumbers malware, causes misleading evaluations when experimental settings do not match deployed settings. Pendlebury et al. [84] break down how experimental biases affect the evaluation metrics, specifically *spatial bias* (relating to the class balance) and *temporal bias* (relating to when data is used for training and not introducing future knowledge). Our approach evaluates how an ML-based malware detector generalizes on previously unseen malware families and is complementary to the spatial and temporal biases. Arp et al. [6] outline ten common pitfalls that are generic to ML in computer security and suggest remediations. Our work introduces an additional pitfall of *optimistic generalization* relating to assuming an overly optimistic ability of an ML model to generalize on novel data. Optimistic generalization is related to sampling bias as presented by Arp et al. [6]. Rather than focusing on the distribution of the sources of the data, our work focuses on detecting novel attacks and malware families specifically when they come from the same source.

This paper makes the following contributions:

- We identify *optimistic generalization* as the overly optimistic evaluation of the ability an ML model to generalize on novel data that deviates from the training data. This is a major concern as attackers continually morph their attacks to evade detection. Our method for evaluating *generalization bias* is complementary to spatial and temporal biases defined by Pendlebury et al. [84].

- We empirically demonstrate the effects of generalization bias in identifying behaviors in novel malware classes. Balanced accuracy reduces from 81.4% to 60.5% when generalization bias is considered (Section 6.3). Even using generative modeling approaches shown to correlate with generating behaviors [117], balanced accuracy only slightly improves to 63.8%.

- We examine the similarity between malware families and show that when a family is held out from training, it is almost always considered out of distribution (OOD) (Section 6.4).

- In support of measuring the similarities between malware samples, we present a novel genomics-based similarity measure that operates on the raw binary (Section 6.4.1). The primary benefit of such an approach is a more reasonable distance measure as opposed to using the Euclidean distance that is often used.

- We propose to use the presented methods for determining if a novel sample is in-distribution or OOD. If the data point is OOD, it should be rejected similar to Jordaney et al. [45]. However, rather than using the model confidence, we propose to use model-agnostic methods as the confidences from ML models, particularly deep learning methods, have been shown to be overly confident on obviously OOD data samples [75].

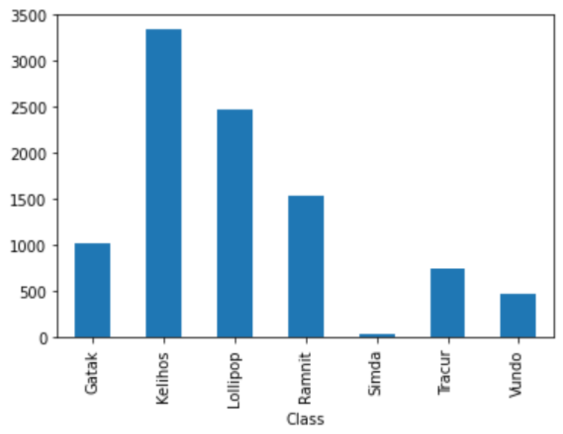


Figure 6-1. Distribution of malware families in the Microsoft Malware Classification Challenge

6.2. Behavioral Malware Classification

Ideally, ML-based malware detection would identify novel malware families without requiring labeled examples to be included in training. Most ML-based malware detection systems either operate as binary classifiers distinguishing among benign software and malware or distinguishing between malware families. While convenient for experimental settings, they do not extrapolate well to new malware families. We build on previous work that identifies malware behavior in executables where a behavior is an action that are taken by malware when it is executed [115]. For example, possible behaviors include DLL injection, process hallowing, or credential harvesting. The underlying assumption is that most malware families share a common set of behaviors. Note that behaviors here are in the labeling and *do not* require dynamic analysis. The labels could be used with features from static or dynamic analyses to train an ML model.

Behaviors are assigned broadly to malware families identified by manually reading through threat reports of each malware family [115]. Behaviors are defined using the malware behavior catalog [18] developed by MITRE. For our experiments, we use the Microsoft Malware Classification that was used as part of a Kaggle Challenge [100]. There are nine malware families, but “Kelihos v1” and “Kelihos v2” are combined as most threat reports do not distinguish between the two versions. The “obfuscated” group is dropped since it was used as a category for executables that could not be placed into a specific family due to obfuscation techniques. Figure 6-1 shows the distribution of the malware families. Note that the number of samples per malware family is highly skewed, which is traditionally difficult for ML models to handle well [15].

We also examine the Malpedia data set [89] which contains significantly more malware families with fewer examples per family. The Malpedia data set offers 2187 malware families but only 5910 samples (averaging two to three samples per malware family). We only use a subset that have the same behaviors that are present in the Microsoft Malware Classification Challenge data set. This results in 238 malware families and a total of 1049 samples of which the largest class

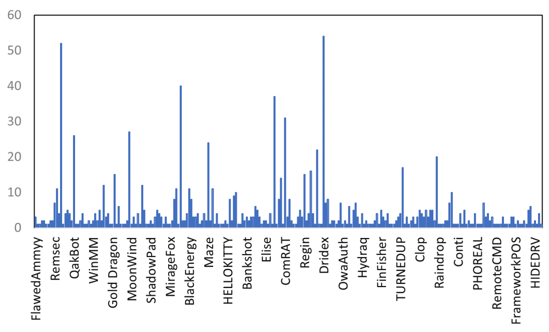


Figure 6-2. Distribution of malware families in the Malpedia data set

contains 54 samples and 88 malware families contain only 1 sample. The distribution of malware classes is shown in Figure 6-2. These two data sets represent depth and breadth of the malware families to investigate generalization bias. Malpedia provides an opportunity to learn the generalities of behaviors since they are diversely represented in multiple malware families. For Malpedia, the CAPA automated tool[2] is used to assign labels given the large number of malware families.

To detect behaviors in malware samples, we use the Malconv model [93] pre-trained on the EMBER data set [4]. Malconv is a deep learning model that operates on the raw byte sequences from an executable to classify it. Malconv can accept an extremely long input sequence and achieves state of the art performance in categorizing malware and goodware. Using transfer learning, we fine-tune Malconv to have 56 output nodes corresponding to the number of behaviors observed in the Microsoft Malware Classification challenge data set. We only train the final two layers of Malconv—after the temporal max pooling layer. We examined training other layers without significant differences in the performance. Note that identifying behaviors in malware is significantly more difficult than simply categorizing executables as malicious or benign. While categorizing executables as malicious or benign achieves upwards of 99% accuracy, 10-fold cross-validation achieves only 84.5% for MMC data. Results are consistent for the Malpedia data set achieving an accuracy of 83.2% using 10-fold cross-validation.

6.3. Generalization Aware Evaluation

We are concerned with how well a trained ML model will generalize to malware families that are not included in the training data. A strong assumption made in ML is that the test data will be drawn from the same distribution as the training data. This assumption generally does not hold for most domains, but it is especially acute in malware detection as (1) technology is rapidly changing with artifacts in the executables to incorporate new technologies and (2) malware is inherently adversarial and actively seeks to evade detection. Thus, this generalization assumption should *not* be made for any security domain.

[2]https://github.com/mandiant/capa

Despite the violation of this assumption, common ML evaluation methods typically do not take it into account when applied to security domains. We term this neglect as a *generalization bias* where unrealistic assumptions about data in deployed settings cause modelers to assume that their models will perform well on novel data, even from new malware families. To mitigate overly optimistic expectations from generalization bias, we argue for explicitly evaluating how well a model will generalize on data that may be considerably different from the training data yet is likely to be observed once deployed and provide a method for doing so. The temporal constraints proposed by Pendlebury et al. [84] partially address this by enforcing a temporal ordering. However, temporal ordering does not simulate the use case of a zero-day encounter with a novel malware family. In this paper a generalization aware evaluation holds out specific families from training and evaluates the performance of an ML on the held out family. This approach could be adapted to other scenarios. This would mimic the introduction of a yet-to-be-discovered malware family and evaluate how well an ML model would generalize to that family.

Similar to the temporal constraints from Pendlebury et al. [84], we propose the following constraint to avoid a generalization bias when dividing a dataset D into a training set Tr and a test set Ts.

C1) Malware Family Consistency. All of the objects in the testing data must *not* belong to malware families that are in the training data:

$$family(s_i) \notin families(Tr), \forall s_i \in Ts \tag{6.1}$$

where s_i is an object in the training set, $family(s_i)$ returns the malware family of s_i and $families(D)$ returns the set of malware families represented in the data set D. For measuring the generalization to novel malware, Equation 6.1 must hold. Violating C1 inflates the performance results by effectively injecting "future knowledge" [84] into the classifier. This constraint directly measures the generalization of an ML model to novel families. This constraint should be used with temporal and spatial constraints when supporting data is available.

6.3.1. Impact of Generalization Bias

By considering generalization bias, we explicitly examine the claim that ML-based malware detection will generalize to novel malware. Our primary hypothesis is that learned ML models are not able to generalize to novel malware families as they are OOD from the training data. To test our hypothesis, we first measure the accuracy of a trained ML-based malware detector that identifies the behaviors in an executable. Following Equation 6.1, we hold out all samples belonging to a given malware family from training and then evaluate on that held out family. On the Microsoft Malware Classification Challenge Dataset, we examine features used by the winning Kaggle team [130] (Say No To Overfitting or SNTO) as well as extracted from the layers of Malconv [93] and a resnet model [38] representing the executables as gray-scale image [74] to lessen the results from a specific feature representation.

As we are holding out a malware family where all samples have the same set of behaviors, we measure the recall (the percentage of correctly identified behaviors that are in an executable), the specificity (the percentage of correctly identifying when a behavior is not present), and the balanced accuracy which is the arithmetic mean of the recall and specificity. As was noted earlier, the Microsoft Malware Classification Challenge is a skewed dataset with representation from certain malware families much larger than others. As class skew affects the classification of ML models [116], we examine several methods for dealing with class imbalance to mitigate these results and to improve generalization as synthetic data generation has improved classification performance.

- **Oversampling**. Oversampling randomly replicates existing data points to balance the number of examples per class, or malware family in this case. We use oversampling to balance the number of examples per malware families and refer to this technique as *balanced*.

- **SMOTE**. Synthetic Minority Oversampling Technique, or SMOTE [15], synthetically creates additional data points by interpolating between data points of the same class rather than replicating the same data points multiple times. Since we are using binary representations as input to MalConv, we use SMOTE on the embedded version of a binary taken from the last layer of MalConv. We then only train the last layer of MalConv using the original data and and the embeddings from SMOTE.

- **GLOW**. GLOW [54] is a flow-based generative model that we use to generate the raw bytes of an executable. Theoretically it can provide exact latent-variable inference without approximation based on a sequence of bijective functions. In the GLOW implementation, the authors propose the use of batch normalization to alleviate the problems encountered in training steps and an invertible 1 x 1 convolution as a generalization of permutations operation instead of using random or reverse permutation before transformation step. A reversible transformation is the affine coupling layer which was initially introduced in NICE [27] that minimizes the negative log-likelihood l as the following:

  $$l(\mathscr{D}) \approx \frac{1}{N}\sum_{i=1}^{N} -\log p_\theta(\tilde{x}^{(i)}) + c \tag{6.2}$$

  where $\tilde{x}^{(i)} = x^{(i)} + u$ represents the i^{th} instance with Gaussian noise $u \sim \mu(0,a)$, and $c = -M * \log a$ where a is determined by the discretization level of the data and M is the dimensionality of x. We set the learning rate to 0.001 and train for 240 epochs using a batch size of 48.

- **CSVAE**. The conditional subspace variational auto encoder (CSVAE) [55] was developed to provide a neural network architecture that incorporates conditional information into the VAE, both during training and also for subsequent use as a generative model. We use the modified version from Smith et al. [117] which was extended to allow for subspace reconstruction as a separate path along with image reconstruction. The subspace is a vector of 0's and 1's corresponding to the behaviors present in an exectuable.

For all techniques, we balance the number of examples per malware family. Results are shown in Tables 6-1-6-3 corresponding respectively to the recall, specificity, and balanced accuracy when holding out the the malware family list in the first column. The results from 10-fold

Table 6-1. The average recall over 10 runs for identifying behaviors in each malware family when held during training using MalConv (MC), Balanced oversampling (Bal), SMOTE (SMO), GLOW, (GLO), and CSVAE (CSV).

Family	MC	Bal	SMO	GLO	CSV
ramnit	0.255	**0.353**	0.334	0.262	0.288
vundo	0.499	0.455	0.463	**0.477**	**0.476**
gatak	0.329	0.385	**0.412**	0.305	0.317
lollipop	0.497	0.634	**0.655**	0.613	0.498
kelihos	0.426	0.532	0.532	**0.649**	0.439
tracur	0.504	0.617	**0.622**	0.494	0.503
simda	0.299	0.366	**0.369**	0.364	0.303
Ave	0.401	0.478	**0.484**	0.452	0.403
10-fold Cross-Validation: 0.730					

Table 6-2. The average specificity over 10 runs for identifying behaviors in each malware family when held during training using MalConv (MC), Balanced oversampling (Bal), SMOTE (SMO), GLOW, (GLO), and CSVAE (CSV).

Family	MC	Bal	SMO	GLO	CSV
ramnit	0.822	0.794	0.791	**0.868**	0.788
vundo	0.781	0.874	**0.878**	0.874	0.796
gatak	0.806	0.764	0.735	**0.824**	0.806
lollipop	0.731	0.728	0.693	**0.750**	0.732
kelihos	**0.820**	0.766	0.755	0.752	0.815
tracur	0.808	0.772	0.757	**0.831**	0.811
simda	**0.900**	0.890	0.890	0.869	0.897
ave	0.810	0.798	0.785	**0.824**	0.807
10-fold Cross-Validation: 0.899					

Table 6-3. The average balanced accuracy over 10 runs for identifying behaviors in each malware family when held during training using MalConv (MC), Balanced oversampling (Bal), SMOTE (SMO), GLOW, (GLO), and CSVAE (CSV).

Family	MC	Bal	SMO	GLO	CSV
ramnit	0.538	**0.574**	0.563	0.565	0.538
vundo	0.640	0.664	0.670	**0.675**	0.636
gatak	0.567	**0.575**	0.573	0.564	0.562
lollipop	0.614	0.681	0.674	**0.682**	0.615
kelihos	0.623	0.649	0.644	**0.700**	0.627
tracur	0.656	**0.694**	0.689	0.662	0.657
simda	0.600	0.628	**0.629**	0.616	0.600
Ave	0.605	**0.638**	0.635	**0.638**	0.605
10-fold Cross-Validation: 0.845					

Table 6-4. The average and standard deviation values for recall, specificity, balanced accuracy, and accuracy for 10-fold cross-validation and generalization aware evaluation on the Malpedia data set. The standard deviation is significantly greater when doing generalization aware evaluation suggesting less predictable behavior by the learned model.

Evaluation	Recall	Std. Dev.	Spec	Std. Dev.	Bal Acc.	Std. Dev.	Acc,	Std. Dev.
10-cv	0.170	0.015	0.947	0.007	0.558	0.007	0.832	0.005
Gen aware	0.276	0.134	0.762	0.143	0.519	0.045	0.646	0.104

cross-validation are shown in the last row. This set of experiments use the embeddings from the temporal pooling layer of MalConv.

The generalization aware balanced accuracy is significantly lower than the balanced accuracy from 10-fold cross-validation—reducing from 84.5% to the best method achieving a balanced accuracy of 63.8%. Additional data can improve the performance of MalConv in identifying behaviors by 3% for some techniques which is quite significant for security domains. However, it is not capable of overcoming generalization bias.

Recall is significantly lower than specificity—partially caused by the class skew of the presence of a behavior. This corresponds with an ML model that is more likely to predict that a behavior is not present and can artificially inflate accuracy results. However, even with family balancing, recall performs significantly worse than specificity raising the question of if an ML actually learns the behavior or not. This raises an additional concern of *if* an ML model actually learns anything semantically meaningful beyond a signature that correlates with the behavior. Current results suggest that the outputs in general ML models are not able to generalize to novel malware families that are not included in training.

The results from Malpedia are similar and are summarized in Table 6-4. Here, the effect on predictions of behaviors not present is greater as specificity is 94.7% but recall is only 17.0%.

Generalization aware evaluation actually improves the recall on novel families but decreases the specificity. The difference in balanced accuracy is not as significant here are as it is with the Microsoft malware classification challenge data set. However, note that one side effect of 10-fold cross-validation versus generalization aware evaluation is the variability of the learned model. The standard deviation of the cross-validated models where constraint 6.1 is not held is small for all measures. For generalization aware evaluations, the standard deviation is at least an order of magnitude greater. This suggests that the behavior of the learned model is less predictable and should be noted before deployment. It also corroborates previous findings that the output is more unpredictable when extrapolating [22].

The question then is *why* do ML-based detectors not perform as well on these data points and what can be done to mitigate their impact?

6.4. Identifying Generalization Bias

Recently, the ML community has began investigating OOD detection techniques that specifically seek to identify data that is inconsistent with the training data [40]. We examine several approaches to measure the similarities (or distance) between malware samples. Inspired by the similarities between biological evolution and malware, we develop a genomics method to measure the similarity of malware samples in addition to using more traditional approaches such a Euclidean distance and distance between neural network embeddings. We then compare how similar malware families are to each other and the other classes.

For the Microsoft Malware Classification Challenge data set, we examine multiple feature representations leveraging embedded representations from Malconv [93], a resnet model [38], the features used by the winning Kaggle team [130], and the raw binaries using our genomics-inspired method described below. To compare the similarity between the malware families, we employ two density-based methods: HDBScan [13] and local outlier factor (LOF) [12]. We utilize our genomics-inspired distance measure in LOF. We compare with Deep MCDD [61], a neural network-based OOD detection method.

6.4.1. Genomics Similarity Measure

To measure the similarity between executables, we propose an alternative distance measure using principles from genomics. The name *computer virus* was taken from biological viruses, which are self-replicating entities that modify their hosts by inserting their information into host genomes [8, 118]. One of the most prominent features shared by computer and biological viruses is the modularity of their information—their code (or genome) can be sub-divided into self-contained sections that can move around and evolve and be transferred independently. This allows viruses to be versatile and adaptive, making them much more likely to evade host defense mechanisms, and makes identifying them using signatures significantly more challenging [37]. Looking at two virus sequences side-by-side in a linear manner may not reveal their similarities due to inter- and intra-shuffling of modules of genomic information. Malware behaves similarly, and the signatures that we can use to identify them and annotate their behaviors may be reorganized and shuffled between files.

As a result, we examined how treating malware files like viral genomes would allow us to understand evolutionary relationships, and eventually refine our understanding of the similarities of the files that could potentially be attributed to behaviors. There have been a number of methods developed in bioinformatics for aligning genomes and mapping evolutionary relationships, including ones specifically designed for complex, multi-sequence alignment [52, 121]. A limitation, however, is that many of these programs have high-order complexity, making them resource hungry. More recent work has seen the implementation of minhash-based pseudo-aligners that use k-mer dictionaries to determine Jaccard edit distances in genomes [79, 29]. Given the large number of files, and their varying sizes, we decided to use Mash, a program that uses minhash sketches to calculate Jaccard distances [79].

Programs such as Mash require DNA sequences as their input. Specifically, this consists of a string of arbitrary length composed of the characters A, T, G and C (representing the four nucleotide bases that DNA is composed of: adenine, thyamine, guanine and cytosine). We convert malware byte files (generated from binaries) into a DNA string. For this, first we removed any stretches of zeros longer than 8 (to avoid long strings of a single base that would be meaningless to match). Then we converted byte value integers into a four code array by using base 4 conversion. This four-code array was converted to DNA by assigning each of the four values to a single base. These new DNA string files were then compared using Mash. The resulting distance matrix was used as a baseline to determine relatedness between samples. For visualizing these relationships, the software platform Cytoscape was used [80].

6.4.2. Similarity of Novel Malware Families

As a first step, we visualize a 2-D representation of the data shown in Figure 6-3. The 2-D projections are obtained using the top two components from a principal component analysis (PCA) and using a t-distributed stochastic neighborhood embedding (t-SNE) projected to two dimensions. While not directly optimized for classification, the malware families cluster together—even with just two dimensions.

We also visualize the groupings of the malware families using Cytoscape in Figure 6-4. As with the PCA and t-SNE projections, we see that malware families largely group together. Connections between data points are represented with black lines linking the data points. Interestingly, there is a large number of data points that do not cluster with the primary groupings, as represented in the bottom portion of the Cytoscape visualization by the larger number of points without any black lines. Rather, singletons (unconnected, or degree-zero points) and smaller clusters are formed. This suggests that the space is sparsely populated by the training data and that there are large variations in the data. For the Cytoscape figure, we use a cut-off value of 0.2. The resulting figure for the Malpedia data set is similar in having a large number of singleton data points that are not connected to the larger cluster (Figure 6-5). However, in this data set there are no clear clusters of the same color or that stand out from the rest of the data points. This is unsurprising given the

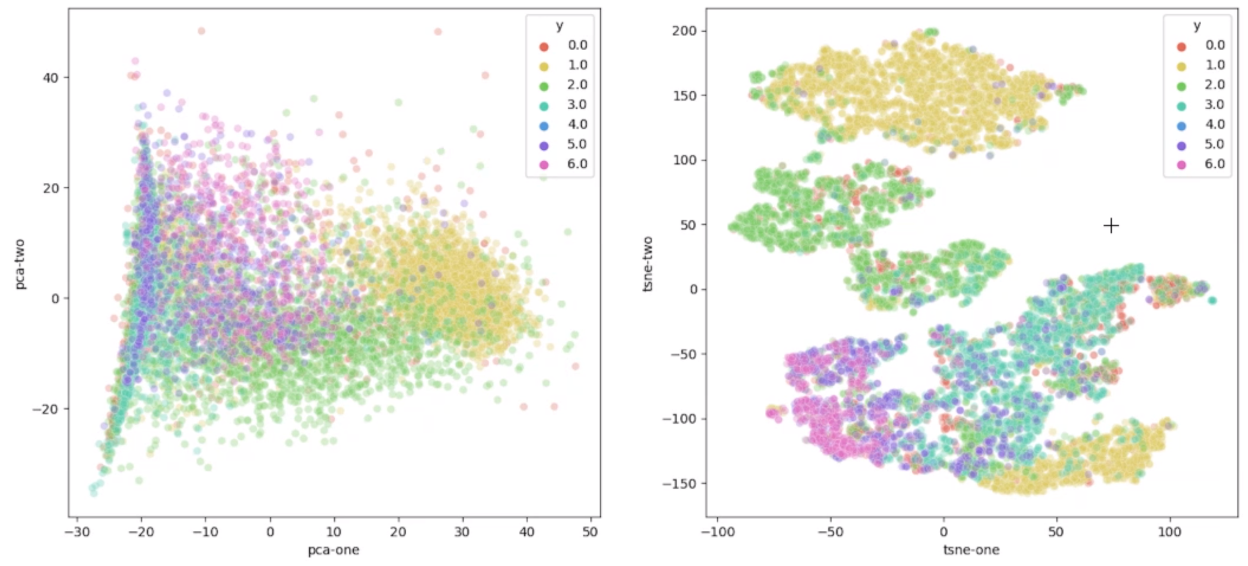


Figure 6-3. Visualization of 2-dimensional projections of the malware data using PCA (left) and t-SNE (right). Each color denotes a separate family. Although not optimized for classification, the embeddings show that malware families cluster together.

large number of families and low number of examples per family. This exemplifies the difference between the data used for training and that which may be encountered once deployed.

Quantitatively, we use generalization aware evaluation to evaluate if data points are considered noise using HDBSCAN [69] and Deep MCDD [61]. To do so, we cluster on all malware families except for the one that is held out and then evaluate the held-out family to see if it clusters with any of the learned clusters. For this analysis we use the embeddings from the temporal pooling layer in MalConv. For HDBSCAN, we use the provided $approximate_{p}redict$ function to designate which cluster additional data would belong to.

The results for the Microsoft Malware Classification Challenge data set are shown in Table 6-5. We also include for HDBSCAN a train/test split to compare performance. Consistently about 40% of the data used for training comprise outliers regardless of which malware family is held out. This is consistent with the Cytoscape images showing that a lot of the space is sparsely covered with singletons.

For the held-out malware family, more than 93% of the test data is considered an outlier for all families when held out from training; and in six families it is more than 98.8%. Results are similar for Deep MCDD. For the train/test split that has examples from all malware families, 72.4% are considered OOD by HDBSCAN. These results are concerning on several levels.

While clustering and classification have different objective functions, these results suggest that not using generalization-aware evaluation produces highly unrealistic performance estimates. If these results are accurate, it suggests that essentially *all* predictions on novel malware families are on OOD data and should *not* be trusted as the behavior of an ML model on these data points is

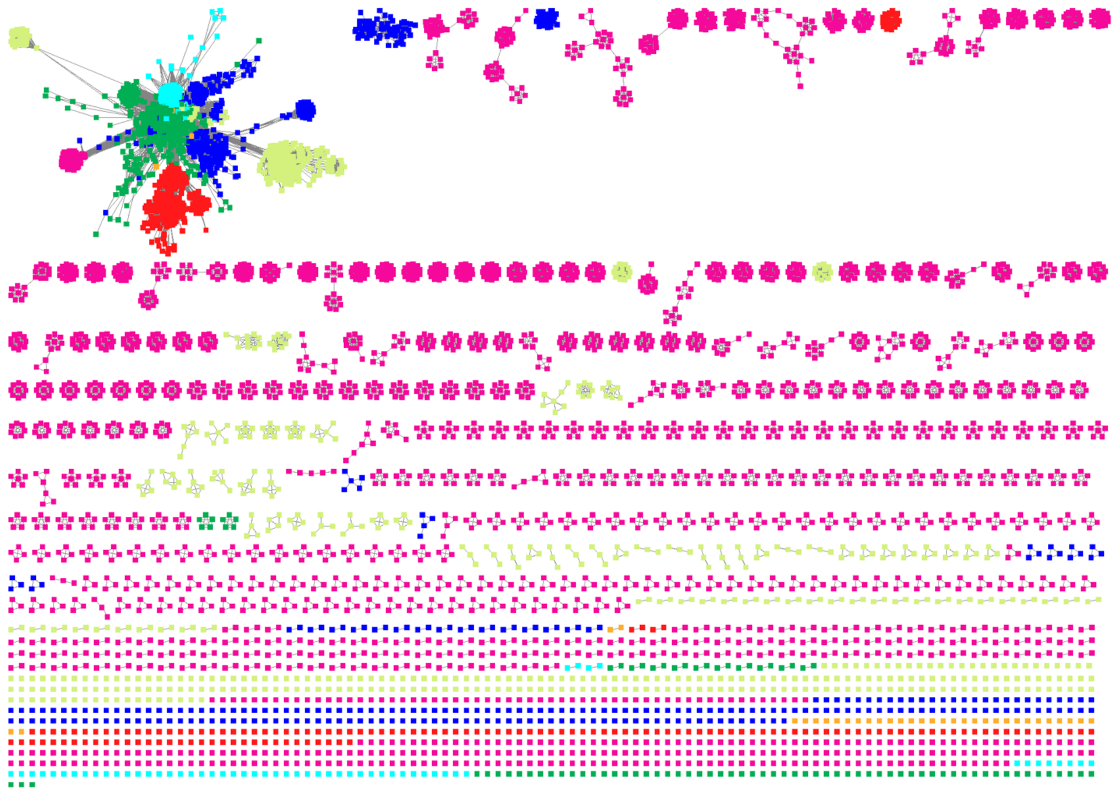


Figure 6-4. Visualization of how data points are linked together when using our genomics distance measure. While many of the families cluster together, there are many that do not link with any other data point and are singletons. This indicates possibly a sparse coverage of the input space.

unpredictable [22]. It also demonstrates that malware is constantly evolving and adapting rapidly enough that current ML-based detection approaches cannot adapt.

To further corroborate these findings, we examine the LOF scores as well and expand the analysis to include multiple feature representations. We examine using a cut-off value of 1 for LOF following common practice in the ML community. We also examine a cut-off value that makes 40% of the training data outliers. The results are shown in Table 6-6 providing a ratio of outlier percentage in train to that in test. For the non-MalConv embedding feature representations, we separate the two versions of Kelihos to show that even between different variants of a malware family generalization bias exists. In all cases, a concerningly high percentage of data points are OOD regardless of the feature representation. We note that the MalConv Dense embedding is highly sparse in general and the 10-fold cross-validation balanced accuracy is 10% lower than that from using embedding from the temporal pooling layer.

For the Malpedia data set with more malware families, the input space is inherently sparser. In this case, the percentage of training data points considered as outliers is similar to that for training—about 90% of the data. This should raise some concerns about the quality of the training data beyond the generalization bias. It also highlights different challenges of using data

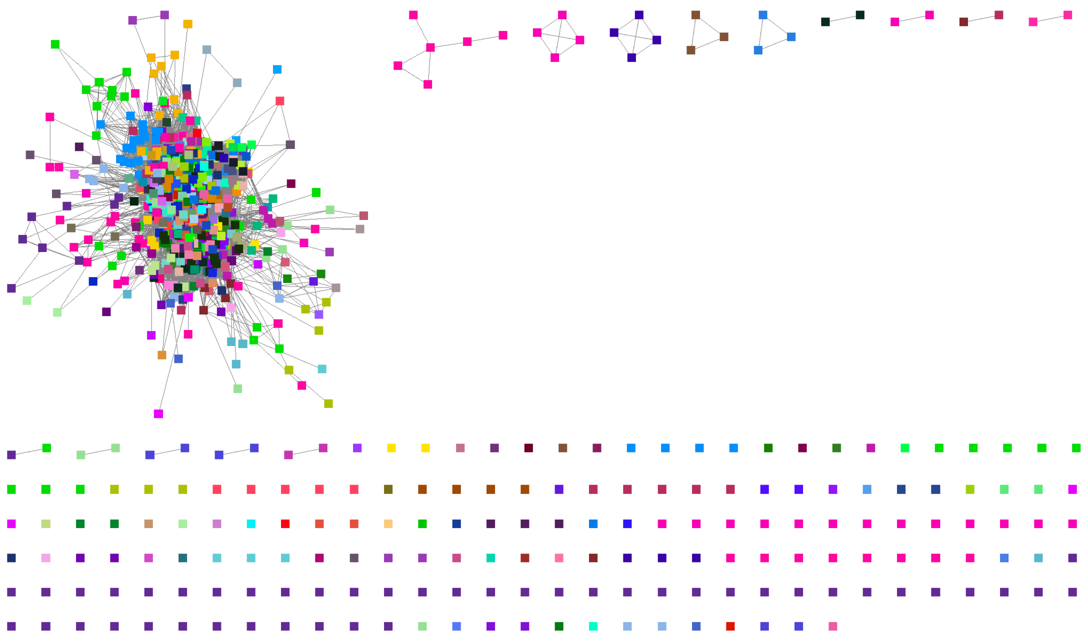


Figure 6-5. Visualization of how data points are linked together when using our genomics distance measure for the Malpedia data set. Many of the data points are connected, but without any noticeable structure. There are many that do not link with any other data point and are singletons. This indicates possibly a sparse coverage of the input space.

sets with more families but fewer examples versus less families with more examples.

6.5. Discussion

We have identified generalization bias and demonstrated that its impact on evaluation metrics is significant and should not be ignored. Traditional evaluation methods remain important to understand how well a model will perform on expected data (i.e., data similar to the training set). It is important to take additional measures to determine where a model will perform unpredictably (and very likely poorly). We propose that a generalization-aware evaluation should be conducted in conjunction with temporal and spatially aware evaluations to properly temper expectations. Our key contribution is in identifying blind spots that lead ML practitioners to assume that when encountering a novel malware family, it is virtually always OOD. Our results suggest that this is *quite significant* with close to 100% of all malware from a held-out family being OOD. This is particularly concerning for several reasons. (1) Malware families can easily evade automated approaches even when systems are defended with ML-based systems. (2) ML-based systems may provide a false sense of security if they are evaluated improperly. To our knowledge, no prior work has examined ML-based detection taking generalization bias or OOD detection into account. Taking generalization bias into account is particularly important in security domains such as malware detection where adversaries purposefully craft examples to evade detection.

Table 6-5. Percentage of data points considered outliers by HDBScan and DeepMCDD when holding out a malware family. We report both the percentage of outliers in the training data (about 40% in all cases) and the percentage in the held-out malware family with HDBScan and DeepMCDD (match to the percentage of outliers in the training data with HDBScan). For the held-out family, almost all of the data points are considered outliers for each held-out family. The last row, "split" represents when a test set is partitioned from the training set.

	Training Outlier	Test Outlier	
Family	HDBScan	HDBScan	DeepMCDD
ramnit	0.384	0.997	0.983
vundo	0.411	0.937	0.947
gatak	0.410	0.997	0.993
lollipop	0.388	0.998	0.956
kelihos	0.538	0.999	0.995
tracur	0.421	0.988	0.800
simda	0.422	1.000	0.976
split	0.439	0.724	-

Generalization to novel data is an open research question in ML, but several lines of promising research exist. Domain generalization [141] and adaptation [23] deal with adapting ML models to evolving domains. These most often require at least a handful of labeled examples and, thus, can only be employed after a samples from a new malware family have been hand labeled.

In the short term, care should be taken to first determine if the prediction of an ML model on a data point should be trusted. This has been suggested previously [45, 84] using the output of an ML model. However, ML models have been shown to be overly confident for OOD data. Our proposed genomics-inspired measure could be used as a model-agnostic measure for OOD detection. Regardless of the method, we suggest a first pass to determine if a data point is OOD or in distribution in conjunction with the prediction. The ML model could then filter out data that it has been used for training. In some cases, the data could be passed to manual examination to help expand the space of the training data.

6.6. Related Work

The misalignment between the reported success of ML-based malware detection and its performance in deployed scenarios has been observed in several works [84, 115, 6, 7, 119]. Notably, Pendlebury et al. [84] identify two main sources of experimental bias that affect the generalization of experimental evaluation to realistic real-world expectations: *spatial bias* and *temporal bias*. Here, bias specifically refers to the experimental settings that introduce certain biases, intentional or not. Spatial bias refers to the misalignment of the ratio of categories (malware vs goodware or malware families) in the training data versus data that is observed when it is deployed. Temporal bias refers to integrating future knowledge about the test data into the training phase, for example, by not structuring evaluation experiments with strict temporal

Table 6-6. Percentage of the training and testing data that are considered outliers with the various feature representations using LOF. DNA represents out genomics-inspired distance measure, Resnet uses the embeddings from a Resnet model, SNTO are the features from "Say No To Overfitting", the winning group from the Microsoft Malware Classification challenge, and the Malconv embeddings from the temporal pooling (TempPool) and last (Dense) layers.

	LOF > 1				
	DNA	Resnet	SNTO	Malconv TempPool	Malconv Dense
ramnit	0.77/0.99	0.82/0.99	0.79/0.98	0.82/1.00	0.8/0.85
vundo	0.78/1.00	0.82/0.99	0.8/1.00	0.83/0.95	0.81/0.82
gatak	0.79/0.99	0.82/0.82	0.8/0.95	0.83/1.00	0.80/0.88
lollipop	0.79/1.00	0.81/1.00	0.79/1.00	0.82/1.00	0.80/0.88
kelihos	0.81/0.88	0.82/0.88	0.77/1.00	0.86/0.99	0.81/0.90
tracur	0.78/1.00	0.82/0.90	0.80/1.00	0.84/1.00	0.81/0.87
simda	0.79/1.00	0.82/0.99	0.79/1.00	0.83/0.98	0.81/0.88
kelihos v2	0.79/1.00	0.93/0.93	0.79/1.00	-/-	-/-
average	0.79/0.98	0.83/0.94	0.79/0.99	0.83/0.99	0.81/0.88
	40%				
	DNA	Resnet	SNTO	Malconv TempPool	Malconv Dense
ramnit	0.40/0.96	0.40/0.95	0.40/0.88	0.40/0.97	0.40/0.44
vundo	0.40/1.00	0.40/0.98	0.40/1.00	0.40/0.71	0.40/0.55
gatak	0.40/0.99	0.40/0.6	0.40/0.92	0.40/0.99	0.40/0.75
lollipop	0.40/0.83	0.4/0.97	0.40/1.00	0.40/1.00	0.40/0.87
kelihos	0.40/0.83	0.4/0.86	0.40/1.00	0.40/0.99	0.40/0.9
tracur	0.40/0.99	0.4/0.70	0.40/0.99	0.40/0.94	0.40/0.81
simda	0.40/0.97	0.40/0.98	0.40/0.95	0.40/0.93	0.40/0.62
kelihos v2	0.40/0.99	0.40/0.00	0.40/1.00	-/-	-/-
average	0.40/0.95	0.40/0.76	0.40/0.97	0.40/0.93	0.40/0.71

ordering. They provide a framework, TESSERACT, for evaluating malware detection taking into account both sources of bias. Our work is complementary to spatial and temporal biases, introducing a third axis—sample bias referring to the misalignment of the types of malware that will be encountered.

Addressing spatial and temporal biases does not remove generalization bias; even after adjusting for these biases, predicted performance against novel malware families is likely to be greatly overstated. Smith et al. [115] proposed to evaluate ML-based malware detection systems by holding out an entire malware family to evaluate how a detector would perform on a novel malware class when detecting behaviors in malware. While they did not specifically examine bias that is introduced into the evaluation process, they suggested that holding out a malware family may provide more realistic expectations when evaluating how well a model will perform on novel data when it deviates from the training data. We utilize their methodology of holding out a malware family and specifically examine the similarities and differences between malware families. They also proposed to classify behaviors rather than malware families or a binary indicator of maliciousness hypothesizing that most malware share certain behaviors and that they would generalize to novel malware families.

The rise in interest in OOD detection in the ML community inspired this work. While most ML models provide a confidence output, the challenge is that the confidence measures are often (1) overly confident for data that is “far” from the training data, especially for deep learning models, making them meaningless [39, 40] and (2) the models model the maximum likelihood and do not model full joint probability of the data; in fact doing so incurs large computational overhead. We look to use a model-agnostic approach based on distances, which has been suggested as being robust to adversarial attacks [138, 16], which is an important characteristic in inherently adversarial domains such as malware detection.

6.7. Conclusion

This paper has identified and evaluated the impact of generalization bias on malware detection. The proposed constraint to hold out entire malware families from training data and to evaluate on held-out malware families revealed that ML models are not able to generalize very well even when common behaviors are being classified between the training and test sets. These issues continue to make detecting malware a challenging problem. We hope that, by identifying generalization bias and the corresponding generalization-aware evaluation, additional techniques can be developed to address this bias in experimental settings and additional techniques that address fundamental issues in classifying malware and automated executable analysis can be developed. One additional area for future work is expanding the amount of labeled data that is available with behaviors. MOTIF [46] is a new malware data set that has 3,095 disarmed PE malware samples from 454 families, labeled with ground truth confidence. Malware family labels were obtained by surveying thousands of open-source threat reports published by 14 major cybersecurity organizations between 2016 and 2021. This could provide a more in-depth analysis of the variety of malware families used for training and possibly lend itself to techniques that can learn from few examples.

Practically, we envision that generalization-aware evaluation will help to temper expectations. In the computer security domain we find that many security analysts are extremely skeptical of ML models due to improper usage and overly confident claims. Our hope is that we can bridge the gap between ML developer and security analyst by being more upfront about the performance of an ML model—when its output is more credible and when it is less credible.

7. FINAL OBSERVATIONS

This LDRD project investigated the ability to improve the detection of novel malware by ML models by (1) detecting behaviors and (2) generating novel data points with specific behaviors. While there were improved results, we uncovered a larger problem in a generalization bias that comes from assuming that the data used for training is representative for the data that will be encountered once the model is deployed. We showed that ignoring this leads to overly optimistic results as most novel malware families are OOD from the data used for training. That being said, we are confident a learned ML model will perform well on data that is similar to that which was used for training. We believe that our genomics-inspired method will improve the detection of novel malware and also help to gain information about the evolution of malware families. This LDRD has only scratched the surface of ML-based malware detection. We hope that it will lead to promising work in improving the detection of novel malware detection, ML-human teaming, and protecting our national interests.

REFERENCES

[1] Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, CODASPY '16, page 183–194, New York, NY, USA, 2016. Association for Computing Machinery.

[2] Fatemeh Amiri, MohammadMahdi Rezaei Yousefi, Caro Lucas, Azadeh Shakery, and Nasser Yazdani. Mutual information-based feature selection for intrusion detection systems. *Journal of Network and Computer Applications*, 34(4):1184–1199, 2011. Advanced Topics in Cloud Computing.

[3] Saswat Anand, Patrice Godefroid, and Nikolai Tillmann. Demand-driven compositional symbolic execution. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 367–381. Springer, 2008.

[4] Hyrum S. Anderson and Phil Roth. EMBER: an open dataset for training static PE malware machine learning models. *CoRR*, abs/1804.04637, 2018.

[5] Endre Anderssen, Knut Dyrstad, Frank Westad, and Harald Martens. Reducing over-optimism in variable selection by cross-model validation. *Chemometrics and intelligent laboratory systems*, 84(1-2):69–74, 2006.

[6] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and don'ts of machine learning in computer security. In *Proc. of USENIX Security Symposium*, 2022.

[7] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security (TISSEC)*, 3(3):186–205, 2000.

[8] John Aycock. *Computer viruses and malware*, volume 22. Springer Science & Business Media, 2006.

[9] Randall Balestriero, Jerome Pesenti, and Yann LeCun. Learning in high dimension always amounts to extrapolation. *arXiv preprint arXiv:2110.09485*, 2021.

[10] James Barry. Sentiment analysis of online reviews using bag-of-words and LSTM approaches. In John McAuley and Susan McKeever, editors, *Proceedings of the 25th Irish Conference on Artificial Intelligence and Cognitive Science, Dublin, Ireland, December 7 - 8, 2017*, volume 2086 of *CEUR Workshop Proceedings*, pages 272–274. CEUR-WS.org, 2017.

[11] Kiran Blanda. Aptnotes, 2016.

[12] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[13] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 160–172. Springer, 2013.

[14] Ferhat Özgür Çatak and Ahmet Faruk Yazi. A benchmark API call dataset for windows PE malware classification. *CoRR*, abs/1905.01999, 2019.

[15] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[16] Jules Chenou, George Hsieh, and Tonya Fields. Radial basis function network: Its robustness and ability to mitigate adversarial examples. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 102–106. IEEE, 2019.

[17] Michael Cogswell, Faruk Ahmed, Ross Girshick, Larry Zitnick, and Dhruv Batra. Reducing overfitting in deep networks by decorrelating representations. *arXiv preprint arXiv:1511.06068*, 2015.

[18] The MITRE Corporation. Malware behavior catalog. `https://github.com/MBCProject/mbc-markdown`, 2019 (Accessed July 2020).

[19] The MITRE Corporation. allitems.txt.z. `https://www.cve.org/Downloads`, May 2022.

[20] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, page 238–252, New York, NY, USA, 1977. Association for Computing Machinery.

[21] Patrick Cousot and Radhia Cousot. Abstract interpretation: Past, present and future. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, New York, NY, USA, 2014. Association for Computing Machinery.

[22] Alexander D'Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D Hoffman, et al. Underspecification presents challenges for credibility in modern machine learning. *arXiv preprint arXiv:2011.03395*, 2020.

[23] Hal Daumé III. Frustratingly easy domain adaptation. *arXiv preprint arXiv:0907.1815*, 2009.

[24] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.

[25] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. Explaining vulnerabilities of deep learning to adversarial malware binaries. *CoRR*, abs/1901.03583, 2019.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2009.

[27] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[28] Felipe N. Ducau, Ethan M. Rudd, Tad M. Heppner, Alex Long, and Konstantin Berlin. SMART: semantic malware attribute relevance tagging. *CoRR*, abs/1905.06262, 2019.

[29] Robert C Edgar, Jeff Taylor, Victor Lin, Tomer Altman, Pierre Barbera, Dmitry Meleshko, Dan Lohr, Gherman Novakovsky, Benjamin Buchfink, Basem Al-Shayeb, et al. Petabase-scale sequence alignment catalyses viral discovery. *Nature*, 602(7895):142–147, 2022.

[30] Bradley J. Erickson, Panagiotis Korfiatis, Zeynettin Akkus, and Timothy L. Kline. Machine learning for medical imaging. *RadioGraphics*, Feb 2017.

[31] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Number 10. Springer series in statistics New York, 2001.

[32] Hisham Galal. Behavior-based features model for malware detection. *Journal of Computer Virology and Hacking Techniques*, 06 2015.

[33] João Gama, Indrundefined Žliobaitundefined, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4), March 2014.

[34] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[35] Lahouari Ghouti. Malware classification using compact image features and multiclass support vector machines. *IET Information Security*, 01 2020.

[36] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4), December 2015.

[37] Hugh MB Harris and Colin Hill. A place for viruses on the tree of life. *Frontiers in Microbiology*, 11:604048, 2021.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[39] Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 41–50, 2019.

[40] Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. Deep anomaly detection with outlier exposure. In *International Conference on Learning Representations*, 2018.

[41] Matthew Hennessy. *The Semantics of Programming Languages: An Elementary Introduction Using Structural Operational Semantics*. John Wiley & Sons, Inc., USA, 1990.

[42] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018.

[43] Kaggle Inc. Malware detection – make your own malware security system, in association with meraz'18 malware security partner max secure software. `https://www.kaggle.com/c/malware-detection`, Aug 2018.

[44] Joxan Jaffar and J-L Lassez. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 111–119, 1987.

[45] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 625–642, 2017.

[46] Robert J Joyce, Dev Amlani, Charles Nicholas, and Edward Raff. MOTIF: A large malware reference dataset with ground truth family labels. In *The AAAI-22 Workshop on Artificial Intelligence for Cyber Security (AICS)*, 2022.

[47] Mohamad Al Kadri, Mohamed Nassar, and Haidar Safa. Transfer learning for malware multi-classification. In *Proceedings of the 23rd International Database Applications & Engineering Symposium*, IDEAS '19, New York, NY, USA, 2019. Association for Computing Machinery.

[48] Gilles Kahn. Natural semantics. In *Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '87, page 22–39, Berlin, Heidelberg, 1987. Springer-Verlag.

[49] M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang, and F. Iqbal. Malware classification with deep convolutional neural networks. In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, 2018.

[50] Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D. Joseph, and J. D. Tygar. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, page 45–56, New York, NY, USA, 2015. Association for Computing Machinery.

[51] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. *arXiv preprint arXiv:2006.06676*, 2020.

[52] Kazutaka Katoh, Kazuharu Misawa, Kei-ichi Kuma, and Takashi Miyata. Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform. *Nucleic acids research*, 30(14):3059–3066, 2002.

[53] W. P. Kegelmeyer, K. Chiang, and J. Ingram. Streaming malware classification in the presence of concept drift and class imbalance. In *2013 12th International Conference on Machine Learning and Applications*, volume 2, pages 48–53, 2013.

[54] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 10215–10224. Curran Associates, Inc., 2018.

[55] Jack Klys, Jake Snell, and Richard S. Zemel. Learning latent subspaces in variational autoencoders. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pages 6445–6455, 2018.

[56] Brian Krebs. Krebs on security. `https://krebsonsecurity.com/`, 2009.

[57] Pradnya Kumbhar and Manisha Mali. A survey on feature selection techniques and classification algorithms for efficient text classification. *International Journal of Science and Research*, 5(5):9, 2016.

[58] Volodymyr Kuznetsov, Johannes Kinder, Stefan Bucur, and George Candea. Efficient state merging in symbolic execution. *Acm Sigplan Notices*, 47(6):193–204, 2012.

[59] S. Lawrence, C. L. Giles, Ah Chung Tsoi, and A. D. Back. Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997.

[60] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[61] Dongha Lee, Sehun Yu, and Hwanjo Yu. Multi-class data description for out-of-distribution detection. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1362–1370, 2020.

[62] Valentine Legoy, Marco Caselli, Christin Seifert, and Andreas Peter. Automated retrieval of att&ck tactics and techniques for cyber threat reports. *arXiv preprint arXiv:2004.14322*, 2020.

[63] Peng Li, Limin Liu, Debin Gao, and Michael K. Reiter. On challenges in evaluating malware clustering. In Somesh Jha, Robin Sommer, and Christian Kreibich, editors, *Recent Advances in Intrusion Detection*, pages 238–255, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[64] Swee Kiat Lim, Aldrian Obaja Muis, Wei Lu, and Chen Hui Ong. MalwareTextDB: A database for annotated malware articles. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1557–1567, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[65] Edward Loper and Steven Bird. NLTK: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, pages 63–70, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[66] Reed Exhibitions Ltd. Cybercrime costs global economy $2.9m per minute. `https://www.infosecurity-magazine.com/news/cybercrime-costs-global-economy/`, Jul 2019.

[67] Mario Lučić, Michael Tschannen, Marvin Ritter, Xiaohua Zhai, Olivier Bachem, and Sylvain Gelly. High-fidelity image generation with fewer labels. In *International conference on machine learning*, pages 4183–4192. PMLR, 2019.

[68] Kin-Keung Ma, Khoo Yit Phang, Jeffrey S Foster, and Michael Hicks. Directed symbolic execution. In *International Static Analysis Symposium*, pages 95–111. Springer, 2011.

[69] Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software*, 2(11):205, 2017.

[70] Microsoft Corporation. Structure of the registry, 2018, (Accessed May 2020).

[71] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[72] Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing*. Wiley Publishing, 3rd edition, 2011.

[73] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. Malware images: Visualization and automatic classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, VizSec '11, New York, NY, USA, 2011. Association for Computing Machinery.

[74] Lakshmanan Nataraj, Sreejith Karthikeyan, Gregoire Jacob, and Bangalore S Manjunath. Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*, pages 1–7, 2011.

[75] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436, 2015.

[76] Thanh Thi Nguyen, Cuong M Nguyen, Dung Tien Nguyen, Duc Thanh Nguyen, and Saeid Nahavandi. Deep learning for deepfakes creation and detection: A survey. *arXiv preprint arXiv:1909.11573*, 2019.

[77] Mutsuo Noguchi and Hirofumi Ueda. An analysis of the actual status of recent cyberattacks on critical infrastructures. *NEC Technical Journal, Special Issue Cybersecurity*, 12(2):19–24, 2019.

[78] Fahmi Salman Nurfikri, Mohamad Syahrul Mubarok, et al. News topic classification using mutual information and bayesian network. In *2018 6th International Conference on Information and Communication Technology (ICoICT)*, pages 162–166. IEEE, 2018.

[79] Brian D Ondov, Todd J Treangen, Páll Melsted, Adam B Mallonee, Nicholas H Bergman, Sergey Koren, and Adam M Phillippy. Mash: fast genome and metagenome distance estimation using minhash. *Genome biology*, 17(1):1–14, 2016.

[80] David Otasek, John H Morris, Jorge Bouças, Alexander R Pico, and Barry Demchak. Cytoscape automation: empowering workflow-based network analysis. *Genome biology*, 20(1):1–15, 2019.

[81] Y. Oyama, T. Miyashita, and H. Kokubo. Identifying useful features for malware detection in the ember dataset. In *2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW)*, pages 360–366, 2019.

[82] Seunghyun Park and Jin-Young Choi. Malware detection in self-driving vehicles using machine learning algorithms. *Journal of advanced transportation*, 2020, 2020.

[83] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[84] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 729–746, 2019.

[85] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, oct 2014. Association for Computational Linguistics.

[86] Roberto Perdisci and ManChon U. Vamo: Towards a fully automated malware clustering validity analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC 2012, pages 329–338, New York, NY, USA, 2012. Association for Computing Machinery.

[87] Huu-Danh Pham, Tuan Dinh Le, and Thanh Nguyen Vu. Static pe malware detection using gradient boosting decision trees algorithm. In Tran Khanh Dang, Josef Küng, Roland Wagner, Nam Thoai, and Makoto Takizawa, editors, *Future Data and Security Engineering*, pages 228–236, Cham, 2018. Springer International Publishing.

[88] P. Jonathon Phillips, Harry Wechsler, Jeffrey Huang, and Patrick J. Rauss. The feret database and evaluation procedure for face-recognition algorithms. *Image and Vision Computing*, 16(5):295–306, 1998.

[89] Daniel Plohmann, Martin Clauss, Steffen Enders, and Elmar Padilla. Malpedia: A Collaborative Effort to Inventorize the Malware Landscape. *The Journal on Cybercrime & Digital Investigations*, 3(1), 2018.

[90] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

[91] PyTorch. *Training a Classifier*, 2020 (Accessed June 2020).

[92] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[93] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K Nicholas. Malware detection by eating a whole exe. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[94] Edward Raff, Richard Zak, Russell Cox, Jared Sylvester, Paul Yacci, Rebecca Ward, Anna Tracy, Mark McLean, and Charles Nicholas. An investigation of byte n-gram features for malware classification. *Journal of Computer Virology and Hacking Techniques*, 14:1–20, 2018.

[95] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

[96] Karthik Raman. Selecting features to classify malware. In *Proceedings of InfoSec Southwest*, 2012.

[97] Marco Ramilli. Malware training sets: a machine learning dataset for everyone, 2016 (Accessed February 2020).

[98] Alexander J. Ratner, Stephen H. Bach, Henry R. Ehrenberg, and Chris Ré. Snorkel: Fast training set generation for information extraction. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 1683–1686, New York, NY, USA, 2017. Association for Computing Machinery.

[99] Monica Rogati and Yiming Yang. High-performing feature selection for text classification. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, CIKM '02, page 659–661, New York, NY, USA, 2002. Association for Computing Machinery.

[100] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. Microsoft malware classification challenge. *CoRR*, abs/1802.10135, 2018.

[101] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[102] Phil Roth. Ember improvements, 2019 (Accessed April 2020). Part of CAMLIS.

[103] Phil Roth, Hyrum Anderson, and Sven Cattell. Extending ember, 2019. [Accessed April 2020].

[104] Ethan M. Rudd, Felipe N. Ducau, Cody Wild, Konstantin Berlin, and Richard E. Harang. ALOHA: auxiliary loss optimization for hypothesis augmentation. *CoRR*, abs/1903.05700, 2019.

[105] Lukas Schott, Jonas Rauber, Matthias Bethge, and Wieland Brendel. Towards the first adversarially robust neural network model on mnist. *CoRR*, abs/1805.09190, 2018.

[106] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

[107] David Sculley, Jasper Snoek, Alexander B. Wiltschko, and Ali Rahimi. Winner's curse? on pace, progress, and empirical rigor. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018.

[108] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. Avclass: A tool for massive malware labeling. In Fabian Monrose, Marc Dacier, Gregory Blanc, and Joaquín García-Alfaro, editors, *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses*, volume 9854 of *Lecture Notes in Computer Science*, pages 230–253. Springer, 2016.

[109] Offensive Security. Exploit database archive. `https://www/exploit-db.com/`, 2003.

[110] Giorgio Severi, Tim Leek, and Brendan Dolan-Gavitt. Malrec: Compact full-trace malware recording for retrospective deep analysis. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 15th International Conference, DIMVA 2018, Proceedings*, Lecture Notes in Computer Science, pages 3–23. Springer-Verlag, 1 2018.

[111] M. Zubair Shafiq, S. Momina Tabish, Fauzan Mirza, and Muddassar Farooq. Pe-miner: Mining structural information to detect malicious executables in realtime. In Engin Kirda, Somesh Jha, and Davide Balzarotti, editors, *Recent Advances in Intrusion Detection*, pages 121–141, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[112] Micha Sharir, Amir Pnueli, et al. *Two approaches to interprocedural data flow analysis*. New York University. Courant Institute of Mathematical Sciences . . . , 1978.

[113] Samarth Sinha, Animesh Garg, and Hugo Larochelle. Curriculum by texture, 2020.

[114] Michael R. Smith, Joe Ingram, Christopher Lamb, Timothy Draelos, Justin Doak, James Aimone, and Conrad James. Dynamic analysis of executables to detect and characterize malware. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 16–22, 2018.

[115] Michael R Smith, Nicholas T Johnson, Joe B Ingram, Armida J Carbajal, Bridget I Haus, Eva Domschot, Ramyaa Ramyaa, Christopher C Lamb, Stephen J Verzi, and W Philip Kegelmeyer. Mind the gap: On bridging the semantic gap between machine learning and malware analysis. In *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*, pages 49–60, 2020.

[116] Michael R Smith, Tony Martinez, and Christophe Giraud-Carrier. An instance level analysis of data complexity. *Machine learning*, 95(2):225–256, 2014.

[117] Michael R Smith, Stephen J Verzi, Nicholas T Johnson, Xin Zhou, Kanad Khanna, Sophie Quynn, and Raga Krishnakumar. Malware generation with specific behaviors to improve machine learning-based detection. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 2160–2169. IEEE, 2021.

[118] Alan Solomon. All about viruses. `https://web.archive.org/web/20120117091338/http:/vx.netlux.org/lib/aas10.html`. Retrieved: 2014-07-17.

[119] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*, pages 305–316. IEEE, 2010.

[120] Sophos. The state of ransomware in healthcare 2021. Technical report, Sophos, May 2021.

[121] Alexandros Stamatakis. Raxml version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 2014.

[122] Blake E. Strom, Andy Applebaum, Doug P. Miller, Kathryn C. Nickels, Adam G. Pennington, and Cody B. Thomas. Mitre att&ck™: Design and philosophy. Technical Report 18-0944-11, MITRE Corporation, McClean, VA, July 2018.

[123] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, pages 4278–4284. AAAI Press, 2017.

[124] Antonio Torralba, Kevin P. Murphy, William T. Freeman, and MArk A. Rubin. Context-based vision system for place and object recognition. In *Proceedings Ninth IEEE International Conference on Computer Vision*, volume 1, pages 273–280, 2003.

[125] Jamal Toutouh, Erik Hemberg, and Una-May O'Reilly. Data dieting in gan training. In *Deep Neural Evolution*, pages 379–400. Springer, 2020.

[126] Knowledge Transfer. Micro and macro averages for imbalance multiclass classification. `https://androidkt.com/micro-macro-averages-for-imbalance-multiclass-classification/`, Aug 2021.

[127] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[128] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman. Robust intelligent malware detection using deep learning. *IEEE Access*, 7:46717–46738, 2019.

[129] R. Vyas, X. Luo, N. McFarland, and C. Justice. Investigation of malicious portable executable file detection on the network using supervised learning techniques. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 941–946, 2017.

[130] Xiaozhou Wang, Jiwei Liu, and Xueer Chen. *Microsoft Malware Classification Challenge (BIG 2015): First Place Team: Say No to Overfitting*, 2015 (Accessed July 2020).

[131] Mike Wasikowski and Xue-wen Chen. Combating the small sample class imbalance problem using feature selection. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1388–1400, 2010.

[132] Webroot. 2019 webroot threat report. Technical report, Webroot, Broomfield, Colorado, February 2019.

[133] Michael D Wong, Edward Raff, James Holt, and Ravi Netravali. Bolstering binary datasets for malware detection through programmatic data augmentation. In *Proceedings of the AI4Cyber/MLHat: AI-enabled Cybersecurity Analytics and Deployable Defense Workshop*, 2022.

[134] Yan Xu, Gareth Jones, Jintao Li, Bin Wang, and Chunming Sun. A study on mutual information-based feature selection for text categorization. *Journal of Computational Information Systems*, 3:6, 03 2007.

[135] Guanhua Yan, Nathan Brown, and Deguang Kong. Exploring discriminatory features for automated malware classification. In Konrad Rieck, Patrick Stewin, and Jean-Pierre Seifert, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 41–61, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[136] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, page 412–420, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[137] Yanfang Ye, Tao Li, Donald Adjeroh, and S. Sitharama Iyengar. A survey on malware detection using data mining techniques. *ACM Comput. Surv.*, 50(3), jun 2017.

[138] Pourya Habib Zadeh, Reshad Hosseini, and Suvrit Sra. Deep-rbf networks revisited: Robust classification with rejection. *arXiv preprint arXiv:1812.03190*, 2018.

[139] Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. Differentiable augmentation for data-efficient gan training. *arXiv preprint arXiv:2006.10738*, 2020.

[140] Zhaohui Zheng, Xiaoyun Wu, and Rohini Srihari. Feature selection for text categorization on imbalanced data. *SIGKDD Explor. Newsl.*, 6(1):80–89, jun 2004.

[141] Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. Domain generalization: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[142] Ziyun Zhu and Tudor Dumitraş. Featuresmith: Automatically engineering features for malware detection by mining the security literature. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 767–778, 2016.

[143] ZScaler. Iot in the enterprise: Empty office edition: What happens when employees abandon their smart devices at work? Technical report, Zscaler, 2021.

DISTRIBUTION

Hardcopy—External

Number of Copies	Name(s)	Company Name and Company Mailing Address

Hardcopy—Internal

Number of Copies	Name	Org.	Mailstop
1	L. Martin, LDRD Office	1910	0359

Email—Internal (encrypt for OUO)

Name	Org.	Sandia Email Address
Technical Library	1911	sanddocs@sandia.gov

Sandia National Laboratories