An Adversarial Training Based Machine Learning Approach to Malware Classification under Adversarial Conditions

Sean M. Devine
United States Army
sean.m.devine10.mil@mail.mil

Nathaniel D. Bastian
Army Cyber Institute, U.S. Military Academy
nathaniel.bastian@westpoint.edu

Abstract

The use of machine learning (ML) has become an established practice in the realm of malware classification and other areas within cybersecurity. Characteristic of the contemporary realm of intelligent malware classification is the threat of adversarial ML. Adversaries are looking to target the underlying data and/or models responsible for the functionality of malware classification to map its behavior or corrupt its functionality. The ends of such adversaries are bypassing the cybersecurity measures and increasing malware effectiveness. We develop an adversarial training based ML approach for malware classification under adversarial conditions that leverages a stacking ensemble method, which compares the performance of 10 base ML models when adversarially trained on three data sets of varying data perturbation schemes. This comparison ultimately reveals the best performing model per data set, which includes random forest, bagging and gradient boosting. Experimentation also includes stacking a mixture of ML models in both the first and second levels in the stack. A first level stack across all 10 ML models with a second level support vector machine is top performing. Overall, this work reveals that a malware classifier can be developed to account for potential forms of training data perturbation with minimal effect on performance.

1. Introduction

Machine learning (ML) capabilities have recently been shown to offer astounding ability to automatically analyze and classify large amounts of data in complex scenarios, in many cases matching or surpassing human capabilities. Furthermore, ML is at the forefront of the application of artificial intelligence (AI) to the realm of cybersecurity. While the contemporary cyber domain has become characterized by a reliance on big data and a widespread prevalence of threats, so has the need for increased automation with regards to applying ML for active cyber defense.

Malware, or malicious software, is any program or file that is harmful to a computer user. Malware can perform a variety of functions such as stealing, encrypting or deleting sensitive data, altering or hijacking core computing functions, and monitoring users' computer activity without permission. Malware are intended to gain access to computer systems and network resources, disturb computer operations, and gather personal information without taking the consent of system's owner. Malwares come in wide range of variations like Virus, Worm, Trojan-horse, Rootkit, Backdoor, Botnet, Spyware, Adware, etc., and a particular malware may reveal the characteristics of multiple classes at the same time [1].

Malware classification is the task of determining the class of a sample that has previously been identified as malware. The use of ML for malware classification offers a solution to the diminishing feasibility of classification without automation. ML can effectively classify programs/files as malicious with a high degree of accuracy, thus increasing the level of security and the ability of network administrators to more effectively monitor their systems and computers on the network. Such a fact explains the applicability and effectiveness of using ML for classifying malware. Gandotra et al. [1] provides a survey of the various ML approach that have been proposed for detecting and classifying unknown samples into either known malware families or underline those samples that exhibit unseen behavior, for detailed analysis. Narayanan et al. [2] use several ML classification techniques (artificial neural network, k-nearest neighbors, and support vector machine) for the identification of malware data into their respective classes, whereas Liu et al. [3] use a clustering algorithm to discover new malware families as part of a ML-based malware analysis system. Others have expanded upon the traditional ML approaches using deep learning techniques. For example, Kalash et al. [4] propose a deep learning framework for malware classification that leverage convolutional neural networks by converting

URI: https://hdl.handle.net/10125/70714
978-0-9981331-4-0

malware binaries to grayscale images for training. Additionally, Cakir & Dogdu [5] used a shallow deep learning-based feature extraction method to represent any given malware based on its opcodes, and then gradient boosting was used for the classification task.

The design of these malware classifiers, however, is ultimately flawed and features key vulnerabilities that adversaries may exploit in order to diminish the classifier's effectiveness. For example, Grosse et al. [6] demonstrate how to construct highly-effective adversarial sample crafting attacks for neural networks used as malware classifiers. In general, ML models have been shown to be vulnerable to adversarial manipulation through systematic modification of features known as adversarial examples. These adversarial manipulations are also referred to as adversarial ML attacks [7]. When evaluating the susceptibility of a system to adversarial ML attacks, an important consideration is attacker knowledge of the targeted AI system [8]. The ability of the adversary to interact with components of the AI system or inspect its operation greatly influences the type of attacks that can be reasonably employed.

The first type of access is known as the 'white box' paradigm. With this type of access, the adversary has complete knowledge of the AI implementation internal state, including the software, data, weights, and input and output values. Armed with this knowledge, an adversary would be able to devise and mount a highly specific attack against the AI system. In the 'black box' paradigm, the adversary has no access to internal information of the AI system, but is able to repeatedly probe with input/output pairs to infer the inner state. While a black box implementation limits internal investigation, the property of transferability would permit an adversary to construct a proxy model of the targeted system that replicates the input/output pairings of the original. An accurate proxy will also support surrogate white box analysis. The last access type is known as the 'hidden box' paradigm. In this paradigm, the adversary must make assumptions about the AI model, including its existence (revealed perhaps through overall system behavior) and had no access to the direct system output. The hidden box paradigm is often encountered in cybersecurity applications.

Given these adversarial ML access paradigms, adversarial ML attacks often take three forms: a) poisoning attacks inject incorrectly or maliciously labeled data points into the training set so that the algorithm learns the wrong mapping (degrading prediction quality or intentionally misguide predictions altogether); 2) evasion attacks perturb correctly classified input samples just enough to cause errors in classification (misclassifying malicious behavior as benign); and 3) model inversion which repeatedly test the trained algorithm with edge-case inputs in order to reveal the previously hidden decision boundaries (inferring information about the original training data used to train the targeted model which can pose a potential information privacy risk) [9, 10].

In the first category, a poisoning attack pollutes the training data to skew model behavior. Adversarial data is fed into a model during training to shift the decision boundary in the attacker's favor. This type of attack is common in systems that must rely on observations in the operational domain for its training data. For example, AI spam detectors trained on this adversarial user feedback would incorrectly classify some types of spam emails as "non-spam" [11]. Strong data security, chain of custody, and tamper verification procedures are important to ensure the data used to develop ML models is free from adversarial influence. This may be unexpectedly difficult to prove due to the prevalence of transfer learning in the AI community, in which pre-trained algorithms are reused across domains.

The second category, evasion attack, is the most common adversarial ML use case. In an evasion attack, inputs are engineered to cause classifiers to assign the input to an erroneous class. The most threatening evasion attacks utilize adversarial inputs that are imperceptible to casual human observers and resistant to pre-algorithm data conditioning. In a classic academic example, a picture of a panda was blended with scaled noise and fed into a pre-trained image classifier. The classifier incorrectly assigned the panda image to the "gibbon" class because of changes the additive noise caused to the image in the algorithm feature space [12]. Due in part to the prevalence and maturity of adversarial input tools, evasion attacks are a potential threat anytime computer vision and natural language processing are employed in AI systems.

The third category, model inversion, is the least publicized attack category and most difficult to execute, but carries serious privacy and data security implications. In a model inversion attack, an attacker repeatedly probes the AI system to extract information about the model configuration or embedded training data. For example, researchers were able to passably recreate an image used to train a classifier by systematically probing the model and observing the available output [13]. This category of attack is applicable only to the black box access paradigm but is especially viable when no limits are set on the number of AI system queries and when the system produces weighted results in place of binary decisions. Training data privacy is especially important when the data involved are sensitive or strictly regulated.

For the malware classification task, the ML model extracts features from the programs/files to classify it as some identified type of malware. Most of these ML models are integrated into an antivirus software, making it difficult for malware authors (i.e., adversaries) to know which classifier a malware classification system uses and the underlying parameters of the classifier. However, because ML models for malware classification rely on a pool of training data in order to shape its predictions, this data serves as a target for potential adversaries seeking to exploit. For example, these adversaries can figure out what features a malware classifier uses and can manually modify them, for example, by changing some application programming interface (API) names in the import directory table [9]. In poisoning the data that these classifiers draw upon to shape its predictions, adversaries can lower the accuracy of these classifiers, or even alter the range of acceptable data, thus facilitating the obscuration of potential malware [14].

These adversarial ML attacks in the cyber domain can cause the models to misbehave or reveal information about their inner workings, which poses significant cyber risk that needs to be managed [15]. If ML-based AI systems are to succeed in helping cybersecurity, they must be secure and robust to adversarial attacks. In order to protect against the threat of these adversarial ML attacks, many proactive defense strategies have been developed to serve as countermeasures for adversarial examples. These proactive strategies can make ML models within the cybersecurity domain more robust. One such proactive strategy is known as adversarial training, which entails training classification models with adversarial examples to make the ML model more robust. These adversarial examples must be generated and injected into the training data set [9]. While there are many different approaches for generating adversarial examples, small perturbations are commonly used in practice. Note that adversarial examples should be designed to be close to the original samples and imperceptible to a human, which causes performance degradation of ML models compared to that of a human.

1.1. Related Literature

AI systems have even become essential to big data analytics and data mining. However, its widespread use has led to the identification of various vulnerabilities and its identification by Yu [16] as a major privacy concern. Such vulnerabilities are in part due to the emergence of the adversarial environment. Biggio et al. [17] point the overall design of such systems as the major flaw. The concept of the adversarial environment characterizes the current challenges facing ML-based classification models as they are subject to attacks to degrade their validity and effectiveness. Lowd & Meek [18] describe this condition as they introduce the Adversarial Classifier Reverse Engineering problem which describes the adversary's approach to learning about a malware classifier. Huang et al. [19] would complement Lowd & Meek [18] with their own application of these principles to their concept of adversarial ML. Huang et al. [19] apply a game-theoretic approach to the adversarial learning problem, an approach in which the adversary is actively attempting to determine strategy of the classifier in order to increase the effectiveness of malware attacks.

Khurana et al. [20] offer an analysis of such attacks in the contemporary adversarial learning environment as they describe an approach to combating poisoning attacks. Such attacks involve the poisoning of open-source intelligence data sources with instances designed to produce false positives/negatives by threat defense systems. The concept of the poisoning attack is more specifically the subject of the online centroid detection. According to Kloft & Laskov [21], poisoning attacks attempt to target the centroid and radius of the area of classifier's range of legitimate classifications. With the introduction of malicious training points, an adversary can force the centroid to shift in the direction of the attack. Doing so can allow new attacks to fall within the range of what the classifier considers legitimate data. Barreno et al. [22] define the adversarial model with their concept of the attack model, which looks to classify adversarial attacks based on influence, specificity, and security violation of the attack.

1.2. Research Motivation

This work responds to the need for malware classification that can operate effectively within the emerging adversarial environment. While the development of ML models has been the subject of established research, the motivation is for practical approaches to adversarial training and subsequent evaluation of such classifiers. The underlying aim is thus the development for processes of experimentally developing and testing malware classification models by leveraging an adversarial training based ML approach. The concept is that adversarial training can help decrease a classifications model's susceptibility to adversarial ML such as evasion attacks. Such attacks are the mechanism behind which adversaries alter a ML model's range of acceptability during inference. In achieving this end result, the notable secondary result is an effective, practical method by which models can be experimentally trained and evaluated for effectiveness.

2. Materials and Methods

In an effort to integrate adversarial training into the ML approach for malware classification, this work leverages a stacking ensemble method as a means of building a classification model under adversarial conditions. This approach begins with a comparison of the performance of 10 base classification models when adversarially trained on three data sets of varying data perturbation schemes. This comparison determines the best performing model per data set. In stacking the top performing models in terms of accuracy, this work looks to reveal that a ML model can be developed to account for potential forms of training data perturbation with minimal effect on the overall model performance.

2.1. Attack Model

In order to achieve the goal of assessing and analyzing malware classification model performance within an adversarial environment, this work considers adversarial training as a strategy to proactively defend against adversarial evasion attacks. The conduct of adversarial training involves the generation and injection of adversarial examples into the data set used to train ML models. Thus, this generated adversarial environment is modeled off of the random or indiscriminate data poisoning attack model. This attack model looks to disrupt a classifier's accuracy by randomly perturbing the input data in order to disrupt the training data and ultimately the integrity of the classifier [23]. Specifically, this work models a random perturbation of input data and labels in order to produce a misclassification effect on the classifier.

This attack model is ultimately aligned with our research goals and objectives in that its purpose to define the abstract method in which the adversarial examples are generated, for the sake of analyzing classifier performance under certain adversarial conditions. The attack model is not necessarily meant to directly mimic the full scope of a red-team style attack on the classifier. The scope of the attack model is important to note in this regard. An attack of this nature features a high degree of difficulty to carry out. Because this work is concerned more with ML system security with regards to the direct performance of the classifier, the technical aspects of the adversary's access to the data will be considered but not treated as part of the scope of the attack model. The scope will ultimately consist of the data perturbation scheme and the corresponding intentional poisoning of the training data, training the malware classifier on the poisoned data, and interpreting classifier output.

2.2. Data Understanding

The modeling for this work is based on a training data set of 12,536 samples previously identified as malware; the data were obtained from the Association for the Advancement of Artificial Intelligence (AAAI) 2019 Workshop on Artificial Intelligence for Cyber Security (AICS) challenge problem. Particularly, the samples were identified by a class corresponding to its malware type, and the set of malware types used for this data set consist of Viruses, Worms, Packed Malware, Trojan Horse, and AdWare. For each sample, a dynamic analysis on a Windows virtual machine was performed and the sequence of Windows API calls was extracted. In the data, the class labels and API calls are obfuscated as integers, but the correspondence between any particular API call as a unigram and as part of a bigram or trigram is preserved. Assumptions about the data are that all instances are malware, the classes cover all instances under consideration, and each instance has a single unambiguous label. The data set is in the form of an NxM matrix in JavaScript Object Notation (JSON) format and consists of six fields: "data", "row_index","column_index","col_schema","shape", and "labels." More specifically, the data schema take on the following field, data types, and ranges:

data : int [1:6317199]
row_index : int [1:6317199]
column_index: int [1:6317199]
col_schema : chr [1:106428]
shape : int [1:2]
labels : int [1:12536]

Among the fields present in the JSON file, "data", "row_index", and "column_index" correspond to the contents of the feature matrix used to train ML classifiers. Specifically, the "data" field gives the values of the nonzero entries of the feature matrix, while "row_index", and "column_index" give the row and column indices corresponding to these values. Thus, the feature matrix represents the instance, i, that corresponds to the input, $[row\_index[i], column\_index[i]]$. As such, a particular nonzero entry in the "data" field is represented by the following function, for $i = 1, ..., K$:

$$feature\_matrix[row\_index[i], column\_index[i]] = data[i] \quad (1)$$

The contents of the "col_schema" array take on the form of either unigrams, bigrams, or trigrams of Windows API calls. The array provides names for the columns of the feature matrix. For unigrams, a column name is a distinct integer corresponding to that specific

API call, for bigrams (and trigrams), the column name is a pair (triple) of integers separated by a semicolon. For example, the trigram "27;9;150" represents the sequence of API calls 27, 9, and 150. The "shape" array corresponds to the $[N, M]$ values that dictate the dimensions of the feature matrix. Finally, the "labels" array corresponds to the class of each instance within the feature matrix. Example values of the label are 0, 1, 2, 3, and 4, which map to each of the malware types (i.e., five classes). This schema is unique to the training data as it is used to assess classification performance.

2.3. Data Preparation

Preparing the data for modeling, experimentation and analysis involved first iterating through the range of the "data" schema according to the function defined in Equation (1). At this point the feature matrix was converted to compressed sparse row (CSR) format and saved to a npz file. Furthermore, this initial CSR format of the data was normalized with scaling from -1 to 1 and a fixed standard deviation.

While the described technique provided a feature matrix that could be used to train ML models, the shape of this initial matrix (12536, 6317199) posed an issue of dimensionality. High-dimensional data in general is an issue based on in increased need in the number of samples necessary for a estimator to effectively generalize. Furthermore, in reducing the dimensions, the data can be analyzed and used for training classifiers despite limitations to memory and processing power. For the purpose of this work, responding to the problem of high-dimensionality involved an implementation of singular value decomposition (SVD) on the feature matrix. This SVD technique is the equivalent to Principle Components Analysis in which the columnwise mean is subtracted from the feature values. Such a technique can be tailored so as to produce new, reduced data sets based on a desired number of dimensions. Given the initially prepared data set, the feature matrix was reduced to 100 principal components (dimensions), respectively. Similar to the initial data set, these reduced data sets were formatted as CSR data sets and written to compressed files.

For the purpose of training the malware classifiers, the data sets generated thus far represent the model input (features). The output, or prediction, of such classifiers is ultimately the classification label (0-4), a categorical variable. Such labels were extracted from the initial JSON file of the training data. Analysis and comparison of the ML models is based on the results of the models when trained on both an unperturbed, base data set, and when trained on data sets containing a perturbed set of inputs and a perturbed set of labels, respectively (i.e., data poisoning attack model). More specifically, the classification models are each adversarially trained on three separate data sets and are assessed based on a constant set-aside validation data set containing inputs and the corresponding labels that are pulled from the initial base data set. A comparison of the three data sets used for adversarial training is depicted in Figure 1 and is further explained below to highlight the differences.

C_1	C_2	C_3
• Unperturbed • Serves as a control set • $C_{1,input} = Tr_{input}$ • $C_{1,label} = Tr_{label}$	• $C_{2,input}$ = perturbation of input features • 20% of features per input in Tr_{input} • Stochastic sampling to apply a uniform perturbation (Non-zero values multiplied by 1.5; Zero values replaced with product of 0.1 and random integer between 1 and 10) • $C_{2,label} = Tr_{label}$	• $C_{3,label}$ = perturbation of labels • 20% of labels in Tr_{label} • Values stochastically changed to a uniform random integer between 0 and 4 • $C_{3,input} = Tr_{input}$

Figure 1. Comparison of the Three Adversarial Training Data Sets

The validation set was generated by designating 25% of the base data set, Tr_{input} as a constant, unaltered data set with the naming convention V_{input}. The labels that correspond to the input were similarly designated as V_{label}. These data sets served as method of comparing each classifier training iteration (regardless of input) against an immutable validation set. The remaining 75% of the base data set was used as the unperturbed control data set, which is designated as $C_{1,input}$ with the corresponding set of labels being designated as $C_{1,label}$. As the $C_{1,input}$ and $C_{1,label}$ serve as the unperturbed training set, they act as the control training set to determine the relative effect of the perturbed training sets on each malware classifier.

The overall perturbation scheme leveraging a data poisoning attack model for adversarial training entailed targeting the two aspects of training data that served as potential threat vectors for adversaries. In an attempt to poison the data, an adversary could target either the input features themselves, or the corresponding training labels in order to degrade the classification accuracy. The first set of perturbations on the base data set aimed to naively emulate the poisoning of the data inputs (dimensionality reduced features). The scheme is based on stochastic sampling to apply a uniform perturbation to a percentage of the features per input so as to effectively poison the data and alter the accuracy of a potential classification, without being so drastic as to easily recognizable. The perturbation is based on iterating through the base inputs of $C_{1,input}$

to randomly perturb 20% of the features per input. The perturbation of 20% of the data allows for a level of poisoning that is significant enough so as to disrupt the integrity of the classifier, while ensuring the perturbation remains undetectable to human perception. This ensures our poisoned data abides by the previously defined adversarial sample principles. Of the values selected, the perturbation involves multiplying non-zero values by 1.5 and changing zero values to the product of a random integer between 1 and 10 and 0.1. This stochastically perturbed data set is designated as $C_{2,input}$. The set of corresponding labels, $C_{2,label}$, is unperturbed and is equal to $C_{1,input}$.

The second set of perturbations on the base data set aimed to naively emulate the poisoning of training data classification labels. Given the lower volume of data points in the training labels data set compared to the training inputs data set, the goal with this scheme was to more randomly perturb a percentage of the labels in order to degrade the classification accuracy. Therefore, this stochastic perturbation scheme was based on iterating through the base classification labels of $C_{1,label}$ to randomly perturb 20% of the labels within the base data set. The values selected were changed to a random value between 0 and 4, representative of the different potential malware types. This data set is designated as $C_{3,label}$. Meanwhile, the set of corresponding inputs is unperturbed and is equal to the inputs in data set $C_{1,input}$.

2.4. Model Building

The foundation of this work is the implementation of the 10 most commonly used ML models for supervised learning (i.e., classification): random forest, support vector machine, gradient boosting, logistic regression, artificial neural network, linear discriminant analysis, quadratic discriminant analysis, Naïve bayes, bagging, and decision tree. It should be noted that 10 base classification models, as opposed to more, are used to ensure computational tractability during experimentation and analysis of the stacking methodology. All models are implemented using the Python 3.7 ML library, scikit-learn, or *sklearn*, which facilitates simplicity and efficiency in ML for data mining and analysis. All ML models are implemented with their respective default parameters in order to not influence the adversarial training based approach.

The support vector machine classifier calculates a set of hyperplanes based on the training data, all of which would potentially classify the inputs. In order to identify the optimal hyperplane, the classifier then looks to maximize the functional margin. The margin ultimately represents the distance between the plane and any point of either class. Further, as applied to multi-class classification, the logistic regression classifier distinguishes between classes. The classifier essentially follows a multinomial logistic function in order to model a response variable that describes the probability prediction of the classification label [24].

The multi-layer perceptron classifier is an implementation of an artificial neural network. The artificial neural network consists of hidden layers of perceptrons. Within each layer, inputs are transformed based on trained weight summation, or bias. Weights at each hidden layer are trained using backpropagation, relying on gradient to minimize a cross-entropy cost function. The result is ultimately a set of prediction probability estimates for each input [25].

Linear discriminant analysis (LDA) and quadratic discriminant analysis (QDA) are based on training linear and quadratic decision boundaries within each respective model. Decision boundaries for both models are generating according to Bayes' rule, with the conditional probability being modeled on a Gaussian distribution. the difference between the two models is in the assumed covariances of the Gaussian densities for each class. In LDA, the decision boundaries are calculated under the assumption that the densities for each class share the same covariance matrix. In QDA, there are no assumptions with regard to the covariance matrix for each density. The result, therefore, is the possibility for quadratic decision boundaries as opposed to the strictly linear decision boundaries for LDA [25].

The Naïve bayes classifier is an application of gaussian Naïve bayes classification. The model is an inherently a naive classification approach due to its assumption of conditional independence between pairs of features for a given input. This particular approach assumes the value of the likelihood of the features follows the gaussian distribution in order to calculate its prediction probabilities [25].

The decision tree classifier is essentially a non-parametric approach to classification that models a decision, or classification, as tree-like graphs. The ultimate classification is based on the linear combination of explanatory variables for a given input. The model seeks the shortest sequence of explanatory variables in order to calculate its prediction [24]. Developing the model calls for a recursive partitioning of a training sets with respect to its explanatory variable, so that inputs with similar features are grouped together. The recursive partitioning creates the structure of a tree and facilitates such analysis of the shortest combination of features that enables a prediction [25].

The random forest classifier expands upon the

decision tree in that it is an ensemble method consisting of a series of trees derived from randomized subsets of the training data. Similar to the base decision tree, each tree will produce a probabilistic prediction of the overall classification. This prediction probability is then averaged across all trees, thus producing an overall prediction for the given input features [25]. The model consists of 100 trees derived from the corresponding number of randomized samples from the training data. The bagging classifier is another ensemble method and for this research is implemented using a very similar approach. The key differences in the random subset of the training data, features are drawn without replacement.

In a similar approach, the gradient boosting classifier is an ensemble method that relies on the use of a series of decision trees as weak learners. The classifier then builds a greedy, additive model in which trees are added so as to minimize loss. The minimization is achieved by calculating the steepest descent method. The method is ultimate a numerical approach and involves a calculation of the gradient of the loss function for the given stage of the ensemble [25].

Finally, this work utilizes a meta-learning approach known as stacking to generate the adversarially trained malware classification model. The linear stacking method involves designating models to be stacked and such stack serves as Level 1 of the model. The "stacking" ensemble method, as depicted in Figure 2, trains and runs the selected models in order to capture each model's respective predictions. The predictions for each are then compiled to form a new data set in which the predictions become the features for each input and correspond to a label in the validation or test set. The new "stacked" data set is then pushed to Level 2 of the linear stacking model which consists of a base classifier. The Level 2 classifier is trained on the new data set so that it is essentially predicting a label based on the predictions of the Level 1 classification models [26].

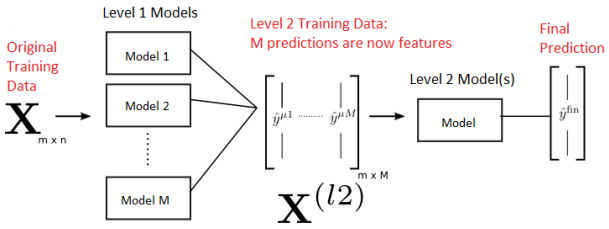

Figure 2. Description of the Data Flow Within the Linear Stacking Ensemble Method

3. Results and Discussion

3.1. Base Model Development and Deployment

The base models serve as the foundation of this research, as their performance when trained on C_1,C_2, and C_3, respectively, determines the Level 1 make-up stacked across the top performing models per data set. Table 1 lists the accuracy scores of the base models. Each model was trained on each of the three data sets, and the accuracy report is based on comparing the model prediction to the V_{label} set.

Table 1. Accuracy Scores for Base Models for Given Input Data Sets C_1, C_2, and C_3

Base Model Classification Accuracy			
Base Classifier	C_1	C_2	C_3
Random Forest	**0.9294**	0.9262	0.8931
Support Vector Machine	0.9135	0.9007	0.9103
Gradient Boosting	0.9211	0.9175	**0.9111**
Logistic Regression	0.9007	0.8923	0.8927
Artificial Neural Network	0.9167	0.9175	0.8927
LDA	0.8836	0.8820	0.8880
QDA	0.8050	0.6886	0.5060
Naïve Bayes	0.5602	0.4944	0.5817
Bagging	0.9226	**0.9215**	0.8668
Decision Tree	0.8919	0.8967	0.8002

Initial analysis of the adversarially trained ML model accuracy reveals the general trend of data set C_2, a stochastic perturbation of the features, causing a slight drop in the overall accuracy of the models and C_3, a stochastic perturbation of the labels, causing a more drastic decrease in accuracy. The main take away from the base model assessment, however, is that given the respective data poisoning attack model perturbation schemes, the results in Table 1 provide insight into the top performing models, and, thus, the models most likely to improve malware classifier performance in the subsequent stacked model.

For data set C_1, the random forest classifier is the best performing model based on its accuracy of 0.9294. It is key to note, however, that while not the top performing models relative to accuracy, the bagging and gradient boosting classifiers produce similarly high accuracies of 0.9226 and 0.9211, respectively. In the case of the bagging classifier, it appears to be the second highest performing classifier for both data sets C_1 and C_2, respectively. In both deployments, the model is outperformed by the random forest classifier. Given that the random forest classifier is the top performer for data set C_1, a second iteration of the model should not be included in the stacked model as doing so would only add duplicate label predictions from the same source. Because of this, this research designates the bagging

classifier as the top performer for data set C_2. The bagging classifier's performance when trained on C_1 also adds a degree of relative surety that it is a strong candidate for inclusion in the stacked model.

Analysis of the gradient boosting classifier demonstrates that although it is not a top performer for data set C_1 or data set C_2, its accuracy of 0.9111 when trained on data set C_3 is the highest among all of the base classifiers. It also is key to note that while the data poisoning attack model perturbation scheme in data set C_3 seems to have had a relatively drastic effect on the other base models, its effect on the accuracy of the gradient boosting classifier is relatively minimal.

While the accuracy score served as the primary performance measure to compare models above, Table 2 highlights other ML performance measures (precision, recall, F1-score) for each of the 10 base classification models across the three data sets. Follow-on experimentation and analysis of the stacking method will only display results with accuracy used as the primary performance measure for comparison.

Table 2. Other Performance Measures for Base Models for Given Input Data Sets C_1, C_2, and C_3

Base Classifier	Data Set	Precision	Recall	F1-score
Random Forest	C_1	0.92	0.77	0.83
	C_2	0.94	0.76	0.83
	C_3	0.79	0.75	0.77
Support Vector Machine	C_1	0.89	0.72	0.79
	C_2	0.90	0.66	0.73
	C_3	0.89	0.71	0.78
Gradient Boosting	C_1	0.90	0.74	0.81
	C_2	0.91	0.73	0.79
	C_3	0.89	0.72	0.78
Logistic Regression	C_1	0.85	0.70	0.75
	C_2	0.89	0.62	0.70
	C_3	0.89	0.63	0.69
Artificial Neural Network	C_1	0.80	0.82	0.81
	C_2	0.83	0.78	0.80
	C_3	0.76	0.79	0.77
LDA	C_1	0.76	0.72	0.74
	C_2	0.77	0.70	0.73
	C_3	0.78	0.72	0.75
QDA	C_1	0.64	0.84	0.70
	C_2	0.58	0.79	0.61
	C_3	0.57	0.73	0.55
Naïve Bayes	C_1	0.48	0.58	0.46
	C_2	0.49	0.54	0.43
	C_3	0.48	0.60	0.47
Bagging	C_1	0.87	0.77	0.81
	C_2	0.91	0.75	0.81
	C_3	0.72	0.73	0.72
Decision Tree	C_1	0.76	0.77	0.77
	C_2	0.78	0.76	0.77
	C_3	0.59	0.70	0.63

3.2. Stacked Model Development and Deployment

The stacked model draws upon these three top performing models across the three data sets in order to generate the Level 1 classification. Based on each classifier's predicted label for each input, the Level 2 classifier in the stack will predict an overall label for a given input in the data set. Given the Level 1 stack (generated by the random forest, bagging, and gradient boosting classifiers) and the original 10 base classification models, the product is ultimately 10 stacked models all based on a constant Level 1 stack. Similar to the base models, these 10 stacked models can be trained/evaluated on C_1, C_2, and C_3 respectively in order to compare model accuracy relative to each perturbation scheme. Table 3 lists the classification accuracy of each ML model relative to each data set as a means of comparing model performance.

Ultimately, the key takeaway from the results of the stacked model is their performance relative to the Table 1 results. When trained on C_1 the random forest classifier predicts with an accuracy of 0.9294, when trained on C_2 the bagging classifier predicts with an accuracy of 0.9215, and when trained on C_3 the gradient boosting classifier predicts with an accuracy of 0.9111. In analyzing the Table 3 results, it is clear that the stacked models do not outperform the top performing base model relative to data sets C_1, C_2, and C_3, respectively. However, it is key that the stacked models have only a minimal negative effect on the accuracy of the classifier. Specifically, the stacked model with a Level 2 support vector machine classifier achieves an accuracy of 0.9183, 0.9147, and 0.8900 for data sets C_1, C_2, and C_3, respectively.

Table 3. Accuracy for "Best of" Stacked Models for Given Input Data Sets C_1, C_2, and C_3

Stacked Model Classification Accuracy				
Level 1 Stack	**Level 2 Classifier**	C_1	C_2	C_3
Random Forest, Bagging, Gradient Boosting	Random Forest	0.9091	0.9095	0.8768
	Support Vector Machine	**0.9183**	**0.9147**	**0.8900**
	Gradient Boosting	0.9099	0.9099	0.8856
	Logistic Regression	0.6800	0.8297	0.6826
	Artificial Neural Network	0.9163	0.9115	0.8808
	LDA	0.7041	0.7085	0.6882
	QDA	0.8696	0.8788	0.8589
	Naïve Bayes	0.9167	0.9135	0.8844
	Bagging	0.9111	0.9063	0.8824
	Decision Tree	0.9111	0.9091	0.8864

In an effort to analyze the effects of this adversarial training based ML approach, this work considers two models based on a Level 1 stack of all base models (10 models stacked rather than three) with a Level 2 consisting of the gradient boosting classifier and the support vector machine classifier, respectively. In analyzing the classification accuracy of these two additional stacked models, which is provided in Table 4, it is interesting to note that the classification models also outperform the stacked models based on only the top three performing base models. The stack of all 10

models with the support vector machine as the Level 2 classifier even outperforms the top performing base models for data sets C_2 and C_3.

Table 4. Accuracy for Stack of All Base Models for Given Input Data Sets C_1, C_2, and C_3

Classification Accuracy of a Stack Across All Models			
Level 2 Classifier	C_1	C_2	C_3
Gradient Boosting	0.9222	0.9222	0.9091
Support Vector Machine	**0.9274**	**0.9264**	**0.9179**

Given this overall top performing model, a recommendation for subsequent real-world implementation is to further expand on this adversarial training based ML approach with a final ensembling. Because the stack of all models with the Level 2 support vector machine model produced a classification based on each of the three data sets, C_1, C_2, and C_3, the three models can be linearly combined relative to their predicted probabilities. Given the three sets of predicted probabilities per class per input, the maximum a posterior sum of probability values per class per input can be used to determine the overall malware classification per input. Specifically, the result will be a single set of vectors corresponding to each input of the base data set. Within each vector, select the highest value, just as the highest prediction probability is selected to produce a classification prediction; this is also known as a soft voting ensemble. The ultimate result is an accurate malware classification that is designed using adversarial training to account for the potential for a base, unperturbed data set, a perturbation of features, and a perturbation of labels.

4. Conclusions

This work using a ML approach for malware classification ultimately called for training and evaluation of ML models based on adversarial training (i.e., purposeful perturbation of the training data based on a data poisoning attack model) as a proactive defense strategy against evasion attacks. This work looked to demonstrate that doing so aided in identifying those base classification models best equipped to consider varying degrees of data poisoning. Stacking these models created a classifier that is designed considering the potential of poisoned training data, all while limiting that effect of poisoning and aimed to ensure accurate classification modeling performance.

The results of this work suggest that this adversarial training based ML approach that leverages stacking ensures strong classification model accuracy. Furthermore, by increasing the number of models (from three to 10) in the Level 1 stack, the stacked model is better equipped to classify in adversarial conditions than the base models alone. The trade off, however, is a slightly diminished classification accuracy in instances where training data may be unperturbed.

Overall, the Level 1 stack across all 10 classification models with a Level 2 support vector machine can be considered the best performing model, as it outperformed even the base models when considering adversarial conditions and perturbations within training data sets. There of course exists even more potential for this model to be expanded using a final soft voting ensemble approach designed to account for all perturbation schemes at once. This implementation can only further expand the ML approach for malware classification modeling, making it more apt to handle the potential for adversarial conditions.

4.1. Limitations and Future Work

First, the shear size of the data set and the number of features per input exposed a significant memory limitation of the machine used for classification model design and implementation. This memory limitation drove the need for a high degree of dimensionality reduction of the initial base data set. The effect was minimal due to the scope of this work. Future work, however, may call for analysis of the classification models on larger, non-dimensionally reduced data sets.

Second, the data poisoning attack model perturbation schemes used within this adversarial training based ML approach incorporated relatively naive perturbations based upon simple stochastic sampling. Specifically, in a real adversarial attack, perturbations of the data set could be more intelligently directed, rather than random in nature. Considering how robust models are to intelligently chosen perturbations might provide additional insight and value. Further, future work should investigate and compare more advanced adversarial example generation techniques [8], such as evolutionary computation (genetic algorithms, particle swarm optimization, etc.), deep learning (e.g., generative adversarial networks), and other generative methods.

Third, future work calls for the optimization of the initial base classification models. Given the objective of this work, using the "default" status of the models was sufficient due to the focus on presenting the adversarial training based ML approach. However, future efforts should focus on refining the base models prior to the stacking efforts. In particular, an experimental design could be used for hyper-parameter tuning and optimization of each of the malware classifiers. By optimizing the hyper-parameters of the base models, this

could prove to improve the classification accuracy of the final stacked model.

Finally, future work should operationalize the final soft voting ensemble approach to add the prediction probabilities of the final stacked model for each data set. This work provides somewhat of an experimental foundation, but further research should focus on practical implementation of the adversarial training based ML approach for malware classification under adversarial conditions.

References

[1] Gandotra, E., Bansal, D. & Sofat, S. (2014). Malware Analysis and Classification: A Survey. *Journal of Information Security*, 5(2), 44440.

[2] Narayanan, B., Djaneye-Boundjou, O. & Kebede, T. (2016). Performance analysis of machine learning and pattern recognition algorithms for malware classification. *Proceedings of the 2016 IEEE NAECON and Ohio Innovation Summit.* IEEE.

[3] Liu, L., Wang, B., Yu, B. & Zhong, Q. (2017). Automatic malware classification and new malware detection using machine learning. *Frontiers of Information Technology & Electronic Engineering*. 18, pp. 1336-1347.

[4] Kalash, M., Rochan, M., Mohammed, N., Bruce, N., Wang, Y. & Iqbal, F. (2018). Malware Classification with Deep Convolutional Neural Networks. *Proceedings of the 2018 9th IFIP International Conference on New Technologies, Mobility and Security.* IEEE.

[5] Cakir, B. & Dogdu, E. (2018). Malware classification using deep learning methods. *Proceedings of the ACMSE 2018 Conference*, 10, pp. 1-5. ACM.

[6] Grosse, K., Papernot, N., Manoharan, P., Backes, M. & McDaniel, P. (2016). Adversarial Perturbations Against Deep Neural Networks for Malware Classification. *arXiv preprint arXiv:1606.04435v2.*

[7] Biggio, B. & Roli, F. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84, pp. 317–331.

[8] Alhajjar, E., Maxwell, P., & Bastian, N. (2020). Adversarial Machine Learning in Network Intrusion Detection Systems. *arXiv preprint arXiv:2004.11898.*

[9] Shetty, S., Ray, I., Celik, N., Mesham, M., Bastian, N. & Zhu, Q. (2019). Simulation for Cyber Risk Management – Where are we, and where do we want to go. *Proceedings of the 2019 Winter Simulation Conference* (Ed. Mustafee et al.), pp. 726-737. IEEE.

[10] Shipp, T., Clouse, D., De Lucia, M., Ahiskali, M., Steverson, K., Mullin, J. & Bastian, N. (2020). Advancing the Research and Development of Assured Artificial Intelligence and Machine Learning Capabilities. *Proceedings of the AAAI Fall 2020 Symposium on Artificial Intelligence in Government and Public Sector.*

[11] Bursztein, E. (2018). Attacks Against Machine Learning - An Overview. Retrieved from https://elie.net/blog/ai/attacks-against-machine -learning-an-overview/.

[12] Goodfellow, I., Shlens, J. & Szegedy, C. (2015). Explaining and Harnessing Adversarial Examples. *arXiv preprint arXiv:1412.6572v3.*

[13] Fredrikson, M., Jha, S. & Ristenpart, T. (2015). Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Securit*y, pp. 1322-1333.

[14] Jagielski, M., Oprea, A., Biggio, B., Liu, C., Nita-Rotaru, C. & Li, B. (2018). Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning. *Proceedings of the IEEE Symposium on Security and Privacy*, San Francisco, CA.

[15] De Lucia, M. & Cotton, C. (2020). A network security classifier defense: against adversarial machine learning attacks. *Proceedings of the 2nd ACM Workshop on Wireless Security and Machine Learning.*

[16] Yu, S. (2016). Big privacy: Challenges and opportunities of privacy study in the age of big data. *IEEE access*, 4, pp. 2751-2763.

[17] Biggio, B., Fumera, G., & Roli, F. (2014). Security evaluation of pattern classifiers under attack. *IEEE transactions on knowledge and data engineering*, 26(4), pp. 984-996.

[18] Lowd, D., & Meek, C. (2005, August). Adversarial learning. *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 641-647. ACM.

[19] Huang, L., Joseph, A. D., Nelson, B., Rubinstein, B. I., & Tygar, J. D. (2011, October). Adversarial machine learning. *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pp. 43-58. ACM.

[20] Khurana, N., Mittal, S., & Joshi, A. (2018). Preventing Poisoning Attacks on AI based Threat Intelligence Systems. *arXiv preprint arXiv:1807.07418.*

[21] Kloft, M., & Laskov, P. (2012). Security analysis of online centroid anomaly detection. *Journal of Machine Learning Research*, 13, pp. 3681-3724.

[22] Barreno, M., Nelson, B., Sears, R., Joseph, A. D., & Tygar, J. D. (2006, March). Can machine learning be secure?. *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pp. 16-25. ACM.

[23] Marshall, A., Parikh, J. Kiciman, E., & Kumar, R. (2019). Threat Modeling AI/ML Systems and Dependencies. Retrieved from https://docs.microsoft.com/en-us/security/engineering /threat-modeling-aiml.

[24] Hackeling, G. (2014). *Mastering Machine Learning with scikit-learn: Apply effective learning algorithms to real-world problems using scikit-learn.* Birmingham: Packt Publ.

[25] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research,* 12, pp. 2825-2830.

[26] Wu, L. (2018). Stacking.py. Retrieve https://github.com/WuLC/MachineLearningAlgorithm /blob/master/python/Stacking.py.