A Thesis

entitled

Evaluating the Resilience of Machine Learning Model

to Adversarial Attacks

by

Sai Suma Sudha

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the

Masters of Engineering Science Degree in Computer Science

Dr. Ahmad Y. Javaid, Committee Chair

Dr. Devinder Kaur, Committee Member

Dr. Weiqing Sun, Committee Member

Dr. Scott Molitor, Dean
College of Graduate Studies

The University of Toledo

August 2023

An Abstract of

Evaluating the Resilience of Machine Learning Model to Adversarial Attacks

by

Sai Suma Sudha

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the Masters of Engineering Science Degree in Computer Science

The University of Toledo
August 2023

This research begins a critical assessment of a machine learning model's robustness, focusing on its resistance to adversarial attacks. We specifically look into how Carlini Wagner (CW) and Model Inversion (MI) adversarial attacks affect the model's performance and stability. The predefined model used for these attacks aims to categorize Android applications as malicious or benign. It bases its decisions on examining permissions as the input features. We duplicated the original model to conduct our experiments without affecting the integrity of the original model. We may see and evaluate the effects of adversarial samples using this method without affecting the performance of the initial model. We next trained several models with various algorithms using the adversarial data generated by two adversarial attacks. Each model performed training using the original and the adversarial samples, simulating a real world situation in which adversarial instances might be present in the training data. After training, the models were saved as pickle files to be reused later. The second step of our study was to develop a specialized classifier with the primary objective of separating original and adversarial samples. This classifier was intended to serve as a filter that would discard adversarial data while allowing actual samples to be sent to the trained model for predictions. It was established as a defense mechanism. We thoroughly evaluated the performance of our suggested strategy, the models, and the

classifier using a variety of performance indicators. At each stage of the procedure, we investigated their accuracy, false positive and false negative rates, and the F-measure to thoroughly assess the system's performance. The results of this work highlight the importance of comprehending and mitigating the effects of adversarial attacks on machine learning models, especially in the context of identifying Android malware. Our effort contributes to the broader discussion about improving the robustness of machine learning models against adversarial threats by exposing the vulnerability of these models to adversarial attacks and outlining a defense mechanism. This helps to ensure more secure and reliable malware detection, which is crucial for using Android applications securely. .

I would like to dedicate this thesis to my family. Their love and belief in me have been a constant source of motivation, guiding me toward achieving my goals. I am especially grateful to my husband, Manikanta Rayala, and my sister, Sai Sushmitha Sudha, for their faith in my abilities and their consistent motivation throughout this process. Their presence by my side has given me strength and determination during challenging times.

Acknowledgments

I express deep gratitude to my advisor, Dr. Ahmad Javaid, whose exceptional guidance and insightful ideas were crucial in successfully completing my master's degree in study and research. I am also grateful to Dr. Weiqing Sun and Dr. Devinder Kaur for their valuable contributions as members of my thesis committee. I want to thank the Department of Electrical Engineering and Computer Science for supporting me through a Research Assistantship. I thank my friends and colleagues at the CSTAR Lab (NE 2033) at the University of Toledo for their support throughout my academic journey. Lastly, I am deeply grateful to my family members for their encouragement and belief in my abilities.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CW	Carlini Wagner
ML	Machine learning
KNN	K-Nearest Neighbors
MI	Model Inversion
LR	Logistic Regression
SVC	Support Vector Classifier
RF	Random Forest
ETC	Extra Trees Classifier
XGBoost	Extreme Gradient Boosting
AdaBoost	Adaptive Boosting
FPR	False Positive Rate
FNR	False Negative Rate
IDS	Intrusion Detection Systems
CNNs	Convolutional Neural Networks
RNNs	Recurrent Neural Networks
DL	Deep Learning

S

Preface

This thesis is submitted to the University of Toledo for a Master of Engineering Science. The research presented in this thesis was conducted under the guidance and supervision of Dr. Ahmad Javaid in the Department of Electrical Engineering and Computer Science at the University of Toledo from May 2022 to July 2023. The work presented in this thesis is original to my knowledge, except where explicitly acknowledged and referenced. No part of this thesis or significantly similar work has been submitted to any other university for any degree, diploma, or qualification.

Sai Suma Sudha

July, 2023

Chapter 1

Introduction

Android is an open-source operating system, the most popular one worldwide. This platform allows developers to innovate and produce applications that meet customer requirements. However, due to its widespread use and open-source nature, it is frequently the target of disgraceful actions. Android malware, or programs created to harm or interfere with Android systems, continues to be a concern online. Robust detection solutions must continually advance as hackers develop new ways to evade detection and compromise equipment [8]. Malware can be discovered using several different techniques [6]. Adversarial samples are malware designed to evade detection by anti-malware programs. [48] Adversarial samples or adversarial examples are data sets intended to trick a model. Adversarial samples are frequently produced by making slight modifications to the original data, which are invisible to humans yet significantly impact the results of ML models. These modifications are precisely created to take advantage of the model's weaknesses in decision-making, which may lead to incorrect predictions or classifications. They can be developed for various

ML applications, such as speech recognition, image recognition, and natural language processing. They are commonly employed to assess ML models' attack susceptibility and resistance. These adversarial attacks try to trick or influence ML algorithms by carefully crafting perturbations to input data. Adversarial attacks on Android malware detection models are the focus of this study. We present a defense mechanism using classifiers trained on adversarial and original samples to strengthen the model's resistance to these sophisticated attacks [26].

1.1 Overview

This examines adversarial learning and how it affects ML models. The study of adversarial attacks and countermeasures aimed at boosting the robustness and resilience of ML models against such attacks is called adversarial learning. We want to create strong defenses and enhance these models' effectiveness by comprehending their flaws and limitations. Introduction to adversarial attacks, which are deliberate attempts to trick or manipulate ML models by adding purposefully designed perturbations to input data, comes first [7]. We use an Android malware prediction model with permissions as the input. We investigate adversarial attack types in depth, including the CW and MI attacks. These assaults show how sensitive ML models are to small changes in input data.

In adversarial ML, CW and MI are characterized as white box attacks. In a white box attack, the attacker is fully aware of the architecture, parameters, and training data of the ML model that is being attacked. The attacker can access the model's internal workings and use this knowledge to create adversarial samples that exploit the model's flaws or vulnerabilities. On the other hand, black box attacks are a sort of adversarial attack in which the attacker has little knowledge of the parameters or internal workings of the target ML model [17].

An optimization-based attack called the CW attack seeks to identify disturbances in the input data to produce adversarial samples [13]. It assumes the target model's architecture, parameters, and gradients are available. Conversely, MI is a technique that seeks to retrieve private data from the training set used to develop an ML model. It might be seen as a white box attack because it needs access to the model's predictions and knowledge of its architecture.

Additionally, we provide a cutting-edge security method that uses a specific classifier to recognize and remove adversarial samples before sending them to the primary malware detection model. We intend to improve the existing detection system's accuracy and robustness to adversarial attacks by including this defense mechanism.

1.2 Purpose and Scope

This thesis examines how adversarial attacks affect ML models that categorize Android malware. Adversarial attacks try to change the input data to modify the outputs of ML models. These attacks could lead to a model misclassifying a malicious application as benign in Android malware classification, allowing the malware to evade security safeguards. To increase the security and robustness of these systems, it is essential to understand how vulnerable ML models are to adversarial attacks. The vulnerability of ML models to adversarial attacks could substantially impact the security of digital systems, given that these models are increasingly used in cybersecurity for tasks like malware detection.

Examining multiple adversarial attack types will be a crucial aspect of the study. Additionally, the study aims to evaluate how well pre-trained models function when subjected to these adversarial attacks. We tested the models' capacity to correctly categorize Android apps as malicious or safe, even in the face of adversarial attacks. This step is crucial for understanding the models' current strengths and limitations

and potential areas for improvement.

The study also suggests designing and training a dedicated classifier for recognizing adversarial samples to increase the resilience of ML models against adversarial attacks. By differentiating malicious and original inputs, the classifier could shield the model's performance against adversarial attacks. Finally, this defense mechanism's performance in identifying and countering adversary threats will be assessed. Ensuring the suggested solution will improve the model's robustness is essential. The thesis aims to develop the field of ML security by following these goals. It aims to suggest methods to increase ML models' resistance to adversarial attacks, ensuring their dependability and efficiency in practical situations.

1.3 Organization of Thesis

Chapter 1 - Introduction: This chapter summarizes the research area—android malware detection—and explains why it is essential to understand adversarial attacks. The chapter also describes the overview of the thesis's structure and organization.

Chapter 2 - Background: This chapter gives background information on the technologies and concepts relevant to the research. It covers machine learning and its applications in malware detection. Additionally, it discusses adversarial learning and explains the idea of adversarial attacks, in which data points are created with the intention of tricking machine learning algorithms. The chapter reviews related literature that discusses current methods for classifying Android malware in the context of adversarial sample detection.

Chapter 3 - Performance of a Pre-trained ML Model under Adversarial Attacks: This chapter assesses a machine learning model's performance in the presence of adversarial assaults. The model and dataset utilized in the experiments are

described. The replication and launch of adversarial assaults, particularly CW and MI attacks, are covered in this chapter. It talks about gathering adversarial samples and conducting training and testing on original and adversarial samples. The findings of the CW and MI assaults are presented in the chapter's performance evaluation of the attacks.

Chapter 4 - Classifier Development: Design, Training, and Predictions: The main focus of this chapter is the design and training of a classifier to differentiate between original and adversarial data. It comprises a performance evaluation of various classifiers to determine the most efficient. The chapter further investigates the classifier's integration with the initial model and assesses the model's overall performance following integration.

Chapter 5 - Conclusion and Future Work: This chapter concludes, summarizes the results, explores the consequences, and suggests directions for further research.

Chapter 2

Background

2.1 Machine Learning (ML)

In Machine Learning, computers may automatically learn from experience, adapt, and improve without being explicitly programmed. It focuses on creating and improving algorithms that provide computers with data access so they can use it to anticipate the future or make judgments. It feeds on enormous amounts of data, which it can analyze to precisely identify patterns and trends that could elude the human eye. One of machine learning's most important uses is in the field of cybersecurity. Traditional security methods, particularly those that rely on known attack signatures and human interaction, must be revised due to data's sheer volume and complexity as the digital world expands exponentially rate [50]. In this scenario ML appears as a powerful tool that may use its capacity to recognize anomalies, patterns, or deviations in massive datasets, which are frequently signs of possible cyber threats.

ML algorithms can learn from previous occurrences and historical data, giving them the predictive ability to anticipate and recognize potential hazards frequently before they happen. This capacity is essential in cybersecurity since it can drastically reduce the duration between a breach and an attack response, potentially saving time, money, and sensitive information. ML has critical applications across multiple aspects of cybersecurity. ML is essential in malware detection systems (IDS) for analyzing network traffic data and spotting out-of-the-ordinary or suspicious activities [12]. To identify potential incursions and quickly react to them, it is beneficial to have the ability to analyze massive databases and uncover hidden patterns. ML may significantly increase the detection rates for malware due to its predictive capabilities. ML models may recognize subtle patterns and signs by extracting characteristics from previously identified benign and malicious files, improving the accuracy of malware detection, and reducing the number of false positives [23]. Phishing attacks, a severe cybersecurity threat, can also be successfully reduced with ML. ML algorithms can accurately detect phishing websites and emails by studying the characteristics of known phishing instances [40]. ML offers a strong defense against such threats by picking up on hidden warnings and patterns typical of phishing attacks. Spam filtering has long been a complex problem [35]. ML can accurately identify and filter out spam emails or messages using complex algorithms, improving user experience and system security. In terms of cybersecurity, ML is a game-changer. ML holds out the promise of an efficient, proactive, and flexible security mechanism capable of staying one step ahead of cyber adversaries as we continue to deal with constantly developing cyber threats.

2.1.1 Random Forest (RF)

A flexible and reliable ML model called Random Forest (RF) is frequently used for classification and regression problems. It is a component of ensemble learning

methods, a strategy that combines various learning models to provide predictions that are more precise and reliable than those that could be made using just one of the individual learning algorithms. During the training phase, the RF method generates many decision trees, each built using a random portion of the training data [46]. The output is either the mode of the classes for classification tasks or the mean estimate of the individual trees for regression tasks. This method creates a "forest" of decision trees, combining their predictions to make the result [1]. A decision tree is a structure like a flowchart where each internal node represents a test on an attribute, each branch a test result, and each leaf node a class label. However, A single decision tree is susceptible to changes in the dataset and frequently overfits the training data, resulting in poor performance on unobserved data. The RF approach deals with this issue, which builds many trees and averages their outputs to boost model robustness and reduce overfitting. The diversity and simplicity of the RF are what gives it its power. The model successfully captures diverse elements of the dataset by constructing several decision trees from random subsets of the data, leading to improved performance. The RF algorithm is a popular choice for various applications in machine learning because, despite its intricacy, it is reasonably simple to use and requires little hyperparameter modification.

2.1.2 Logistic Regression (LR)

A popular ML approach for binary classification issues is logistic regression (LR). A logistic function is used in this statistical model to simulate a binary dependent variable. The LR model calculates the likelihood that a specific input point falls under the [59]. The logistic function, the sigmoid function, is the fundamental idea underpinning logistic regression. An S-shaped curve maps any real-valued number to a value between 0 and 1. The term "logistic" refers to the fact that the outcome prediction in logistic regression is logarithmic rather than linear. This ratio is subjected

to the logistic function to determine the possibility or log-odds. The estimated probability can then be used to predict a binary outcome; if it exceeds a certain threshold, the model will predict the positive class; if not, it will indicate the negative category. The fact that LR not only offers a prediction but also the probabilities corresponding to the predictions is one of its benefits. When we need to estimate the prediction's level of certainty, this feature is useful. LR may be easily applied to numerical and categorical data by transforming categorical data into dummy variables. Additionally, by including a penalty to the loss function that the model minimizes, LR can be regularized to prevent overfitting to the training set of data. Due to its interpretability and robustness, logistic regression is frequently employed in many fields. It is notably used in the financial sector, where it is used to forecast the risk that a client would default on a loan, as well as in healthcare, where it is used to predict disease outcomes based on various warning signs. LR is a potent tool in ML and statistical modeling weapons because of its effectiveness, simplicity, and insightful information it provides about the variables influencing the prediction [36].

2.1.3 Support Vector Machine (SVM)

A powerful supervised learning technique, the Support Vector Machine (SVM), is mainly employed for binary classification. At the same time, it can also be utilized to solve multi-class classification and regression issues. The fundamental idea of SVM is to locate a hyperplane that divides the data into two classes in the best possible way [41]. The hyperplane is selected to maximize the margin, which is determined by measuring how far each class's nearest data points are from the hyperplane. As it seeks to establish the broadest "street" between types, this method makes the SVM robust to overfitting and lowers the possibility of misclassification. The ability of SVM to handle high-dimensional data is its main strength. The kernel trick, also known as projecting data onto a higher-dimensional space where it can be linearly

separated, is a technique SVM uses to handle data that cannot be linearly separated in the original environment [16]. The input data are transformed by kernel functions like the linear, polynomial, radial basis function (RBF), or sigmoid kernels, allowing SVM to locate complex decision boundaries in the transformed space. SVM is adaptable and capable of handling challenging, real-world datasets due to the freedom to select a suitable kernel function. SVM is one of the most well-liked machine learning algorithms because of its reliability, adaptability, and high performance across various areas. It has been extensively employed in multiple domains, including general pattern recognition tasks, hand-written character identification, picture recognition, bioinformatics, and text and hypertext categorization.

2.1.4 K-Nearest Neighbors (KNN)

The lazy learning algorithms include instance-based learning algorithms such as KNN. The name "lazy" alludes to the fact that KNN waits until prediction time to use the training data instead of building a generalized model during the training phase. It is also regarded as a non-parametric technique because no explicit assumptions are made on the functional form of the data [28]. KNN is very useful when the data distribution is uncertain or does not adhere to the assumptions of parametric models because of this characteristic. The fundamental idea behind KNN is to categorize an object based on how it resembles instances in the training set. KNN determines the 'k' examples from the training dataset closest to the new instance when a prediction is needed for an unobserved instance (thus the name K-Nearest Neighbors). Standard distance metrics determining "closeness" include Euclidean, Manhattan, and Minkowski distances. The method then places the new instance in the class with the most members among its closest 'k' neighbors. The object is put into the category of its nearest neighbor when k is equal to 1.

KNN's ease of use, interpretability, and capacity for multi-class issues have made

it useful in various applications. It is frequently utilized in disciplines including pattern recognition, anomaly detection, text mining, and recommendation systems [53]. Despite its advantages, KNN is susceptible to the dimensionality curse. When managing high-dimensional data, dimensionality reduction techniques may be necessary because the performance rapidly declines as the feature space's dimensions rise. Additionally, because KNN is a lazy learner, it can be expensive to compute and slow when making predictions, especially for large datasets. Despite this, the KNN algorithm is adequate for various data-driven applications, especially those that benefit from its simple and intuitive approach to classification and regression.

2.1.5 Extra Trees Classifier (ETC)

Extremely Randomized Trees, another name for the ETC, is an ensemble learning technique that produces many decision trees and aggregates their results. It belongs to the same class of ensemble approaches as Random Forest and Gradient Boosting. Still, because it adds more randomization throughout the model-building process, it further lowers the variance of the model [10]. The fundamental tenet of ensemble methods, such as ETC, is that a collection of "weak learners" can combine to create a "strong learner." A random subset of the data is used to build each decision tree in the ensemble independently. The way ETC separates nodes sets it apart from previous ensemble decision tree-based approaches. ETC employs randomly chosen split points instead of Decision Trees and Random Forests, which select optimal split points. In exchange for a reduction in variance and an increase in bias, the model is more resistant to overfitting [2]. Because no optimal rule is searched for each node, this random selection of characteristics and thresholds to split upon results in a higher bias, offset by a minor variance (because the trees differ more from one another). A crucial hyperparameter for ETC is the number of trees in the forest. Due to its higher randomness, ETC typically requires more trees than a Random

Forest to achieve comparable performance levels. However, as each tree is generated individually, the training of ETC can be done entirely in parallel, resulting in quick computation. The fact that ETC uses random thresholds and is hence insensitive to input scaling can make it more user-friendly than other techniques that call for input standardization. ETC is a robust and practical ensemble learning method that excels at working with sizable datasets with high dimensional space and can be applied to regression and classification applications.

2.1.6 Extreme Gradient Boosting (XGBoost)

Extreme Gradient Boosting, or XGBoost, is a powerful, scalable ML method that excels at predictive modeling applications due to its high performance and effectiveness. Its foundation is the gradient boosting framework, which iteratively combines weak predictive models, typically decision trees, to produce a robust predictive model [18]. The technique of gradient descent optimization to reduce the loss function is called gradient boosting. Due to its accuracy and processing capacity, XGBoost has become extremely well-known in the data science community and has been the algorithm of choice in many winning solutions of machine learning competitions. One unique aspect of XGBoost is its regularization process, which works to prevent overfitting by limiting the complexity of the model. While most boosting algorithms use a loss function, XGBoost augments this function with a regularization term. Both L1 (Lasso Regression) and L2 (Ridge Regression) regularization are included in the regularization term. By setting some feature weights to zero, L1 regularization encourages sparsity, whereas L2 regularization reduces the coefficients of less significant features but leaves them in place. Compared to conventional gradient boosting, XGBoost is a more generalized model because of the extra complexity control [25]. The built in routine for addressing missing values in XGBoost is another essential feature. In real world datasets, missing data is a frequent problem that can significantly impact how

well-machine-learning-algorithms-perform.- XGBoost-is-more-resistant-to-missing-data-than-other-algorithms-since-it-automatically-determines-the-appropriate-imputation-technique-based-on-the-training-loss.- The-capacity-to-process-information-in-parallel,-which-accelerates-learning,-and-built-in-cross-validation-at-each-iteration,-which-minimizes-the-amount-of-boosting-iterations,-are-two-further-benefits-of-XGBoost.- For-many-machine-learning-problems,-XGBoost-is-the-algorithm-of-choice-because-of-its-speed,-performance,-and-adaptability.-

2.1.7 Adaptive Boosting (AdaBoost)

The-adaptive-boosting-machine-learning-algorithm,-called-AdaBoost,-is-employed-for-classification-and-regression-issues.- One-of-the-earliest-and-best-ensemble-algorithms,-it-combines-several-weak-classifiers-to-produce-a-robust-classifier.- Simply-put,-a-weak-classifier-performs-poorly-but-is-still-superior-to-guessing-at-random-[20].- AdaBoost-is-adaptive-in-that-it-iteratively-modifies-the-distribution-of-the-data-to-emphasize-cases-that-were-incorrectly-classified-in-the-past,-enhancing-the-ensemble's-performance.- Each-weak-classifier-in-AdaBoost-is-trained-using-a-random-subset-of-the-entire-dataset.- Each-training-example-in-the-dataset-is-given-a-weight-by-AdaBoost-after-each-training-round,-which-establishes-the-likelihood-that-each-instance-will-be-included-in-the-training-set-for-the-following-classifier.- To-increase-their-chance-of-being-used-as-part-of-the-training-set-for-the-subsequent-classifier,-samples-incorrectly-identified-in-the-previous-round-are-given-greater-weight.- This-procedure-continues-for-numerous-rounds-or-until-the-algorithm-has-added-a-predetermined-number-of-weak-learners,-whichever-comes-first.- Thus,-the-final-model-comprises-numerous-vulnerable-learners,-each-focusing-on-a-particular-data-area-that-was-challenging-for-the-prior-models.- AdaBoost-has-the-benefit-of-being-less-prone-to-overfitting-than-other-learning-algorithms.- It-strives-to-fit-each-point-exactly.- Hence-it-is-sensitive-to-noisy-data-and-outliers-[49].- AdaBoost-is-highly-adaptable-and-may-be-used-for-binary-and-

multiclass classification problems. It can also incorporate any learning technique. AdaBoost can be used with any form of classifier, even though it is commonly employed with decision tree classifiers. As a result, a robust classifier is created that combines the advantages of each member while also making up for any flaws.

2.2 Adversarial Learning

A sophisticated branch of ML called adversarial learning concentrates on learning in a hostile environment. Adversarial learning algorithms focus on scenarios where the data distribution may be purposefully modified or adversely affected by an opponent, unlike typical ML algorithms, which learn from a fixed data distribution. The adversary's objective is typically to make the ML algorithm produce errors. In contrast, the learner aims to categorize or predict data, even hostile data accurately.

In traditional ML, we typically assume that the data distribution is stationary and that the distribution of the test data looks similar to that of the training distribution. This is no more true in hostile circumstances when attackers can deliberately modify the data [43]. As a result, adversarial learning algorithms are made to be stronger and more resistant to these tricks.

This project aims to strengthen the Android malware detection system's resilience against adversarial attacks. In this case, adversarial samples act as fake entities to defeat the model's detection mechanism [42]. The approach includes adversarial learning mechanisms to address this. As a result, it can effectively learn from benign and adversarial samples and produce precise predictions. Using adversarial learning considerably boosts the system's effectiveness and strengthens its defenses against potential security breaches and adversarial attacks.

2.2.1 Adversarial Attacks

In ML, adversarial attacks involve modifying the input data to confuse the ML models and cause them to provide false results. These assaults use the models' built-in weaknesses and pose severe risks to ML and DL models. White-box and black-box attacks are the two main types of adversarial attacks. Attackers using white-box techniques can access the model's architecture, parameters, and training data. The attacker can design sophisticated attacks using this knowledge that frequently go undetected [44]. In contrast, black-box attacks assume that the attacker is only aware of the model's inputs and outputs and is mindful of how the model functions internally. In these situations, the attacker attempts to create malicious programs using this input-output information.

Serious concerns about the dependability and robustness of ML models have been raised due to their vulnerability to adversarial attacks. The potential impact of adversarial attacks could be extensive and harmful as ML models are increasingly used in crucial domains, including cybersecurity, healthcare, and autonomous cars [21]. To ensure the safe and efficient usage of ML models, it is essential to comprehend and mitigate adversarial attacks [43].

2.3 Literature Review

2.3.1 Overview of Android Malware Classification

Malware attacks on the Android platform have significantly increased due to the widespread availability of Android smartphones over the past ten years. Android malware classification has been the subject of numerous studies, particularly on ML-based solutions, due to their performance benefits over traditional rule-based approaches. Typically, collections of Android applications classified as benign or malicious are

used to train ML models. To categorize new, undiscovered applications, they learn the characteristics that set these categories apart, such as requested permissions, API requests, or network activities [27]. Numerous ML methods have been used to categorize Android malware over time, including Decision Trees, Random Forests, Support Vector Machines, Neural Networks, and others [5]. Researchers have also looked into feature selection techniques to reduce the dimensionality of the data and enhance model performance. Malware categorization has also seen an increase in the application of DL. Since they can automatically learn feature representations from raw data, models like CNNs and RNNs have proven particularly effective [4]. This eliminates the need for manual feature extraction. However, adversarial attacks, which alter the input data to make the model misclassify the application, can seriously damage the efficacy of these models. Due to this difficulty, the categorization of Android malware using adversarial ML algorithms has been studied.

2.3.2 Adversarial Attacks in Machine Learning

The safety and reliability of machine learning models are now seriously threatened by adversarial attacks. An adversarial attack involves altering the input data to a machine learning model in a way that leads to inaccurate predictions or classifications by the model. These alterations, often called adversarial instances, are frequently subtle and invisible to the naked eye yet significantly impact a model's output.

Three categories can be used to classify adversarial attacks broadly [15]. *Evasion attack* is an adversarial attack that occurs during testing, where the attacker modifies the input data in a way that leads to an error in the machine learning model. These modifications to the input data are frequently made to be undetectable and hidden [39]. *Poisoning attack* is an adversarial attack during a machine learning model's training phase. The attacker inserts carefully constructed samples into the training set to control the learning process. As a result, the model learns wrong associations

and generates false predictions or conclusions. *Exploratory attack*, the adversary aims to comprehend or expose the model's internals, such as its parameters or structure. This attack often involves querying the model to learn how it behaves.

These attacks severely impact applications for image recognition, natural language processing, and malware detection [34] [30]. They take advantage of ML models' great dimensionality and complexity, which make it challenging to predict and protect against all potential weaknesses. Adversarial attacks against Android malware detection may involve adding, removing, or altering dangerous software features to avoid detection. These dangers highlight the necessity of strong models that can withstand adversarial attacks and continue to deliver accurate and trustworthy predictions. Understanding and countering adversarial attacks has been the subject of extensive research, resulting in developing defense mechanisms and improving models. However, the adverse circumstances are still changing. Therefore this is a very active area of research.

2.3.3 Existing Techniques for Adversarial Sample Detection

To protect ML models against adversarial attacks, adversarial sample detection is essential. This component aims to identify and eliminate changing inputs that attempt to reduce the reliability and accuracy of model predictions. Many different strategies have been suggested over time to deal with this issue.

Adversarial training is one of the approaches that is frequently employed. Adversarial samples are included in the training process to strengthen the model's resistance to adversarial attacks, which is the basic idea underlying adversarial training. The idea is predicated on the concept that exposing the model to adversarial samples during training will allow it to identify and categorize such instances later effectively. Although this method requires a lot of resources, the outcomes have been encouraging. With the development of the fast gradient sign method, which has since been

widely used, Goodfellow and colleagues were significant in helping to establish this concept [24].

Defensive distillation aims to make ML models more resilient to adversarial attacks. Similar to how distillation in chemistry produces a pure substance from a mixture, the term "distillation" is a metaphorical description of the process when a model is trained to generalize the softer output of another model. Two rounds of model training are used in the technique. In the first round, a standard model is trained to provide output probabilities for each class [38]. The targets for the second round of training are then replaced with the output probabilities from the initial training rather than the complex labels at the beginning. The goal is to reduce the sensitivity of the model's decision boundaries by smoothing and strengthening them. Defensive distillation was proposed by Papernot et al. [38].

Feature squeezing is an adversarial detection method that reduces the search space that can be exploited by simplifying the representations of model inputs and making it more challenging to produce adversarial samples. This can be done by reducing the color depth of the images, applying a spatial filter to smooth out the images, or compressing the input data. This approach has been proved by Xu et al. in their study [56].

Gradient Masking or Regularization is a defense strategy used in ML to strengthen the model's resistance to adversarial attacks. To make it more difficult for adversaries to provide adversarial samples, it operates by modifying or masking the gradients of the loss function related to the input during the training phase [11]. The decision boundaries of the model are smoothed or made flat to reduce the impact of adversarial perturbations. Gradient masking can prevent gradient-based attacks but may not always increase system robustness, leaving the system open to other attacks [9].

Chapter 3

Performance of a Pre-trained ML Model under Adversarial Attacks

3.1 Model and Dataset Description

A sophisticated NATICUSdroid system is an ML, specifically on the Random Forest Classifier. This model aims to distinguish between benign (harmless) and malicious (harmful) Android applications. The system's capacity to differentiate between the two proposes a practical solution for the growing malware problem in the Android ecosystem, as stated in [33]. The efficacy of NATICUSdroid is primarily due to how it examines the permissions that the applications request. The system analyzed the permission conditions of over 29,000 excellent and harmful Android applications over approximately ten years (2010-2019). These permissions specify how an application may use features or access specific data on a device. NATICUSdroid can determine

the most critical permissions by carefully examining historical trends in these permissions. These permissions are a combination of native (included in the Android system) and custom (specified by the app developers) permissions. The system gathers and examines these permissions to distinguish between good and bad applications. An application may be marked as suspicious if it requests many permissions or permissions that are not ordinarily required for its stated function. NATICUSdroid can determine the essential permissions by carefully examining historical trends in these permissions. These permissions are a combination of native (included in the Android system) and custom (specified by the app developers) permissions. The system gathers and examines these permissions to distinguish between good and bad applications. An application may be marked as suspicious if it requests many permissions or permissions that are not ordinarily required for its stated function.

3.2 Adversarial Attack methods

Adversarial attacks are techniques used to confuse ML algorithms by supplying carefully crafted data. These attacks exploit the model's weaknesses and could lead to inaccurate predictions or classifications. We have employed the CW Attack and the MI-Attack, two well-known adversarial attack strategies, to evaluate NATICUSdroid's robustness 3-1 this shows the workflow of the attack.

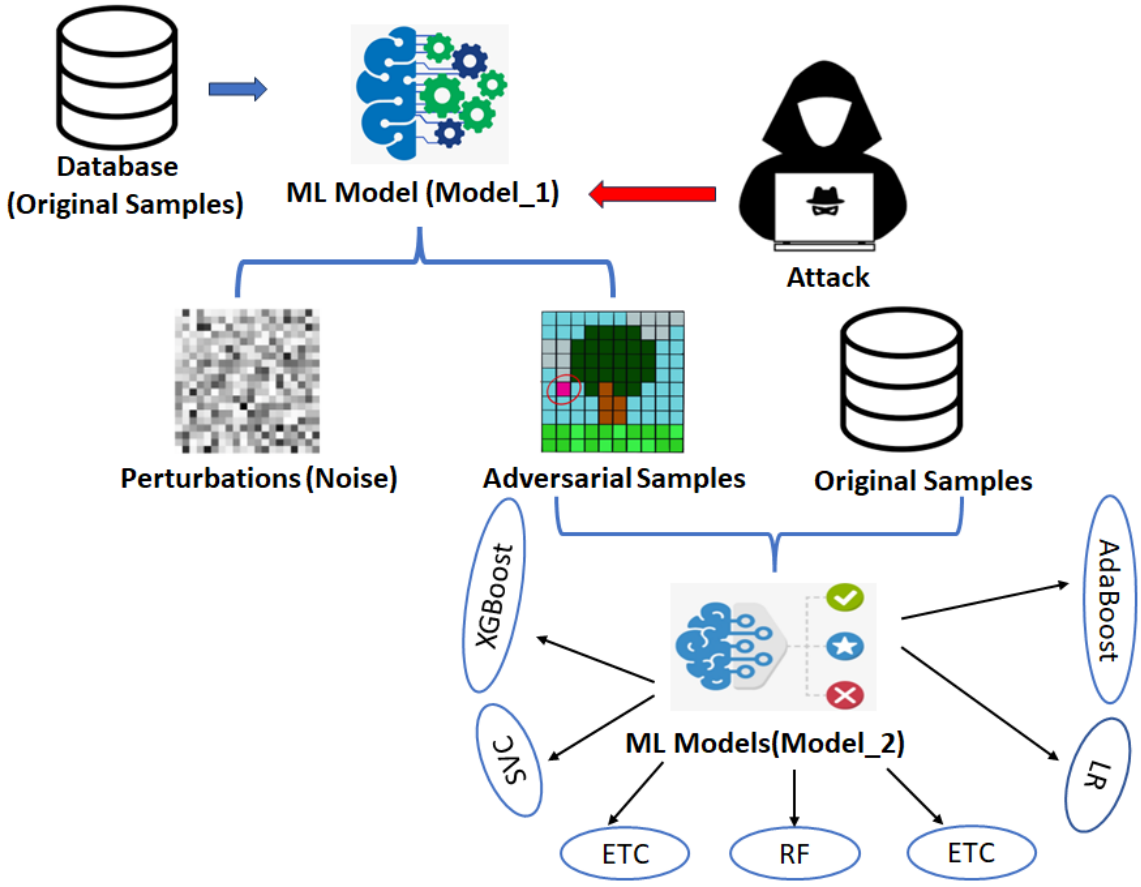

Figure 3-1: Workflow of Launching the Adversarial Attack

3.2.1 Carlini Wagner (CW)

Due to the CW Attack's shown efficiency in producing adversarial scenarios and evaluating the resilience of machine learning models, we decided to use it in our research. This particular kind of adversarial approach, which Nicholas Carlini and David Wagner first described in 2017, excels at creating adversarial samples that are challenging for models to recognize and require minor modifications from the initial input [14]. In applying the CW Attack on our NATICUSdroid model, we began by defining an optimization problem. The objective of this challenge was to find the lowest perturbation that could be added to the initial input to alter the model's

output, thereby making it an adversarial example.

The formulation of this adversarial attack can be represented as

$$min\delta \|\delta\|_p + c.f(x+\delta) where \;\; x+\delta \;\; \epsilon \;\; \{-1,0,1\}^n \tag{3.1}$$

In equation (3.1), x represents an original input instance, δ denotes the adversarial perturbation, $f(x+\delta)$ is the classification function that guides the perturbation to trigger misclassification and n is the number of dimensions [58]. The term $||\delta||_p$ represents the Lp norm, serving as the distance metric to measure the magnitude of the perturbation. Here, $||\delta||_p$ is minimized while ensuring that $f(x+\delta)$ guides the model to the target class. The function f is carefully crafted to become less than or equal to zero when the perturbed instance is misclassified as the target class. To maintain valid binary values (0 or 1), the adversarial examples are clipped and rounded after the perturbation is added.

To solve this optimization problem, the CW attack leverages the change of variables and reformulates the problem as follows

$$minimize||\frac{1}{2}(tanh(w)+1) - x||_2^2 + e.f(\frac{1}{2}(tanh(w))+1) \tag{3.2}$$

This approach enables using standard gradient-based optimization algorithms, such as the trust region method in your implementation, to address the issue. The CW attack, which provides high fooling rates while preserving imperceptible perturbations, thus serves as an efficient adversarial attack strategy in our project.

Hyperparameters can affect the learning process, including how complex the learned model is, how quickly it learns, and how well it performs. 3.1 shows the table for hyperparameters.

Table 3.1: Hyperparameters for CW attack.

Parameter	**Value**
Epsilon	0.3
Perturbation Factor	0.76
Number of Iterations	300
Learning Rate	0.1

As the maximum permitted perturbation for each pixel in the image (or feature in the data), *epsilon* is frequently utilized in adversarial attacks. It is a method of regulating the size of the perturbation and, consequently, the visibility of the adversarial example. The *perturbation* factor is likely a hyperparameter specific to this CW attack technique. It is generally possible to scale the noise added during the attack using the perturbation factor.

The attack optimization process runs for the specified *number of steps or iterations.* More iterations may produce adversarial instances that are more effective, but they also take longer to calculate.

The CW attack's gradient descent optimization approach uses *Learning Rate* as a hyperparameter. The learning rate determines how much the input needs to be changed in response to the computed gradient of the loss function. A higher learning rate will result in more drastic input changes at each step, which could speed up convergence and increase the risk of overshooting the loss function's minimum [45].

3.2.2 Model Inversion (MI)

Fredrikson first proposed the idea of MI attacks, an adversarial attack strategy that tries to extract sensitive data from a machine learning model by flipping the model's behavior [22]. It involves reconstructing inputs and replicating the initial

training data using the model's predictions. In other words, the approach aims to recover the input data from the output by reversing the model's decision-making process.

Utilizing access to the model and its outputs, a MI attack against the NATICUS-droid model tried to aim to extract private information about the training data. This sensitive information is the features (Android permissions) of benign or malware apps that the model has trained to recognize as essential [57].

Table 3.2: Hyperparameters for MI attack.

Parameter	**Value**
max_iter	300
learning_rate	0.1
threshold	0.5

The *max_iter* indicates the function's maximum number of iterations. The function will have more possibilities to identify a successful adversarial sample if this number is more significant, but it will also take longer. *Learning_rate* defines how much the inputs are modified throughout each iteration based on the determined gradient. The *threshold* is the cutoff used to determine when a prediction is sufficiently close to the target label. Lowering this amount will raise the *threshold* for a prediction to be declared a match, potentially making the attack more difficult but the matches more precise.

3.3 Collection of Adversarial Samples

Adversarial samples are carefully crafted adjustments to the original data that are almost unnoticeable to humans but significantly affect the predictions or classifi-

cations made by ML models [48]. Creating adversarial samples often entails making minor changes to the original input data. These perturbations, for instance, can involve gently modifying the pixel values in an image or the numerical values in a data set. These adjustments aim to trick the ML model into making a wrong prediction or categorization. Adversarial attacks are techniques used to confuse ML algorithms by supplying carefully crafted data. These attacks exploit the model's weaknesses and could lead to inaccurate predictions or classifications. In the context of your study, adversarial samples are generated using the CW Attack and the MI, two well-known adversarial attack techniques [37]. The performance and robustness of the NATICUSdroid model are then evaluated under challenging circumstances using these adversarial samples. Testing with adversarial samples is like simulating an adversarial attack on the model. If the model fails to perform well when adversarial examples are included (i.e., misclassifying many instances), this refers to model flaws that must be fixed. This procedure can highlight the model's weak points susceptible to such adversarial manipulation. The model's resilience against such sophisticated attacks can be strengthened by identifying these weak places, improving the model's performance in real-world situations. The goal is to harden NATICUSdroid to withstand attacks that use adversarial samples to trick the system and continue to do a successful job of malware identification.

3.4 Performance Evaluation of the Attacks

This section assesses the effectiveness of adversarial attacks such as the CW and MI attacks. Evaluation metrics are derived by initiating these hostile attacks on a NATICUSdroid model. I created a replica of the model and launched the attacks [55].

Evaluation measures used to evaluate their performance includes Accuracy determined by how many positive and negative predictions were accurate.FPR measures

the proportion of negative occurrences mistakenly labeled as positive, FNR measures the proportion of positive occurrences mistakenly labeled as negative. F-measure is harmonic means of Precision (the proportion of correct positive identifications), and Recall (the percentage of genuine positives that were correctly identified). These analyses were carried out for each batch of 100 samples.

3.4.1 Results of Carlini Wagner

The results of CW attacks show that the attack is successful as the Accuracy of the NATICUSdroid model reduced after the attack. The evaluation is done for every 100 samples, and the results before and after the attack were also mentioned. 3.3 shows the results before and after the CW attack. The results were also plotted as 3-2. As shown by the drastic shifts in evaluation measures, the observed pattern indicates the model's performance after the CW attack is considerably impacted during the first 5000 samples. This may result from the model adjusting to the adversarial manipulation caused by the CW attack.

The performance indications change less drastically after processing 5000 samples. This may be because the model is more stable. After all, it has already adapted to the patterns of the adversarial data [29]. However, it's crucial to remember that, depending on how this model is applied, even minor deviations in findings could have a significant impact.

Table 3.3: Performance Metrics Before and After CW Attack

Performance Metric	**Before CW Attack**	**After CW Attack**
Accuracy	97.88	67.55
False Positive Rate	1.69	4.13
False Negative Rate	2.54	60.63
F-Measure	97.88	54.88

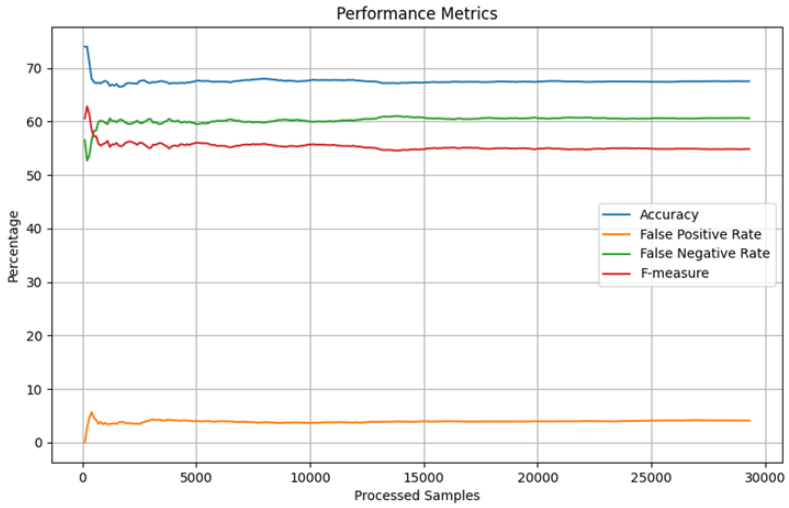

Figure 3-2: Results of CW attack for every 100 Samples

A comparison of numerous evaluation metrics for an ML model both before and after the CW adversarial attack is shown in the table 3.3 and 3-2 shows the results in a graph. The Accuracy metric gauges how accurately the model predicts things in general. Before the CW attack, the model had a remarkable accuracy of 97.88%, correctly classifying 97.88% of the data. However, following the CW strike, the model's accuracy drastically decreased to 67.55%, showing that it needed to be more successful at making accurate forecasts under hostile circumstances. The false positive rate is the percentage of benign (or negative) samples mistakenly labeled as mali-

cious (or positive). The false positive rate was low before the CW attack, at 1.69%, but increased to 4.13% afterward. This growth indicates that the attack may have mistakenly caused the model to label more innocuous samples as malicious. The percentage of harmful (or positive) samples mistakenly classified as benign (or negative) is known as the False Negative Rate. In this instance, the false negative rate increased from 2.54% before the CW attack to 60.63% afterward. This suggests that the model underwent adversarial conditions and failed to accurately recognize a sizable number of false samples, which raises substantial security concerns. Finally, a measure that combines precision and recall—the F-Measure or F-score—also drastically dropped from 97.88% before the CW strike to 54.88% afterward. The decline in the F-score indicates that the CW attack significantly reduced the model's precision and recall, suggesting a compromise in the model's effectiveness in its prediction capability.

3.4.2 Results of Model Inversion

Performance measurements for the Model Inversion (MI) attack indicate success. Each of these adversarial attack strategies was assessed for every 100 samples. Performance metrics were monitored before and after the attacks, allowing for a complete evaluation of its resilience and susceptibility to attack 3-3 shows the performance metrics before and after the attack.

We observed a considerable difference in the performance metrics throughout the first 1200 samples of the MI attack 3.4 shows the plot for every 100 samples. This suggests that the attack significantly affected the performance. The accuracy rate decreased, and the FPR had minor changes, but the FNR experienced the most significant change, increasing drastically. As a result, the F-measure, which considered both precision and recall, decreased significantly.

Table 3.4: Performance Metrics Before and After MI Attack

Performance Metric	**Before MI Attack**	**After MI Attack**
Accuracy	97.88	50.1
False Positive Rate	1.69	0.09
False Negative Rate	2.54	99.41
F-Measure	97.88	1.17

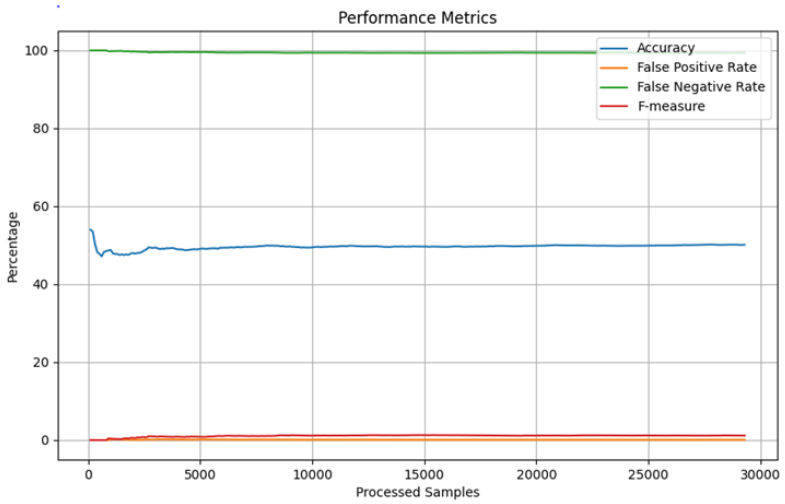

Figure 3-3: Results of MI attack for every 100 Samples

The comparison of various evaluation metrics for an ML model before and after it was subjected to the MI adversarial attack is shown in the table 3.4 and 3-3 shows the graphical representation of the results. The overall accuracy of the model's predictions is measured by its accuracy. The model had a high accuracy of 97.88% before the MI attack, correctly classifying 97.88% of the occurrences. But after being subjected to the MI attack, the model's accuracy dropped to 50.1%. This significant fall indicates a considerable decline in the model's capacity to produce reliable predictions in the face of opposition. The FPR measures how many benign (or negative) samples

are mistakenly classified as malicious (or positive). Interestingly, in this instance, the FPR drops from 1.69% prior to the MI attack to a meager 0.09% following the strike. This suggests that fewer innocuous cases were mistakenly labeled as harmful due to the attack. The percentage of harmful (or positive) samples that were mistakenly classified as benign (or negative) is known as the FNR. Here we notice a significant shift: the FNR soars from 2.54% before the MI attack to an unsettling 99.41% following the strike. This considerable rise shows that the model misidentified most harmful cases under adversarial circumstances. Last, the F-assessment, commonly known as the F-score, a balanced assessment of precision and recall, decreased sharply from 97.88% before the MI attack to just 1.17% after that. The MI assault caused a significant decrease in the model's prediction precision and recall, which is indicated by the sharp drop in the F-score.

3.5 Training with Adversarial and Original Samples

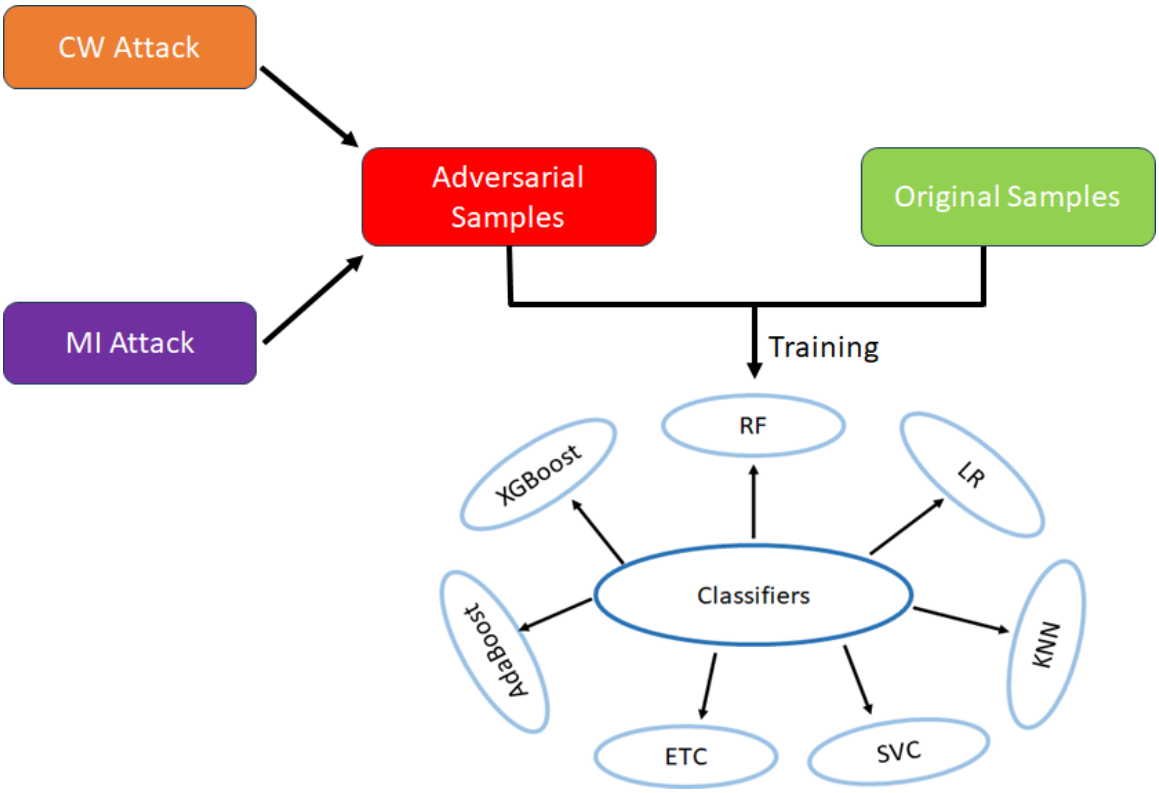

Figure 3-4: Training process of the classifiers with adversarial samples

A key component in enhancing the robustness and resilience of machine learning models to adversarial attacks is adversarial training. This strategy is essential in ML, especially in fields where models could be subject to attacks meant to trick or mislead them. Both adversarial and original samples are used in the training set for adversarial training. Adversarial samples have been marginally altered to make the model forecast incorrect. On the other hand, original samples are the typical, unaltered data the model intends to learn from. The machine learning model gains experience with adversarial samples in addition to conventional data by integrating both types

of samples in the training set. This procedure enhances the model's capacity to identify and correctly categorize adversarial samples by training it to operate in a hostile, more diversified environment [51]. Several machine learning classifiers are used during the adversarial training process. They are K-Nearest Neighbors (KN), Support Vector Classifier (SVC), Random Forest (RF), Extra Trees Classifier (ETC), XGBoost (XG), and AdaBoost (AB). Each classifier has advantages and disadvantages, adding to the model's overall resilience. 3-4 This shows how the classifier training process takes place. These trained classifiers are saved as pickle files after the training step is over. The trained classifiers can be readily loaded using these pickle files in the future without retraining them. ML frequently uses this technique since it uses less time and computing power [31]. The training results are then displayed, demonstrating the effectiveness of each of these classifiers in percentages. These outcomes, which show how well each classifier can handle adversarial and original data, could be accuracy scores, precision, recall, or any other applicable performance parameter.

Classifier	**Training Accuracy**	**Validation Accuracy**	**Test Accuracy**	**Test F-Score**
LR	88.80	88.73	88.73	0.89
KN	92.52	90.36	90.39	0.90
SVC	94.17	90.72	90.57	0.90
RF	98.50	89.40	89.46	0.89
ETC	97.70	89.46	89.49	0.89
XGBoost	95.54	90.39	90.45	0.90
AdaBoost	86.32	86.30	85.86	0.86

Table 3.5: Training, Validation, and Testing results for various classifiers on original and adversarial samples generated at CW Attack.

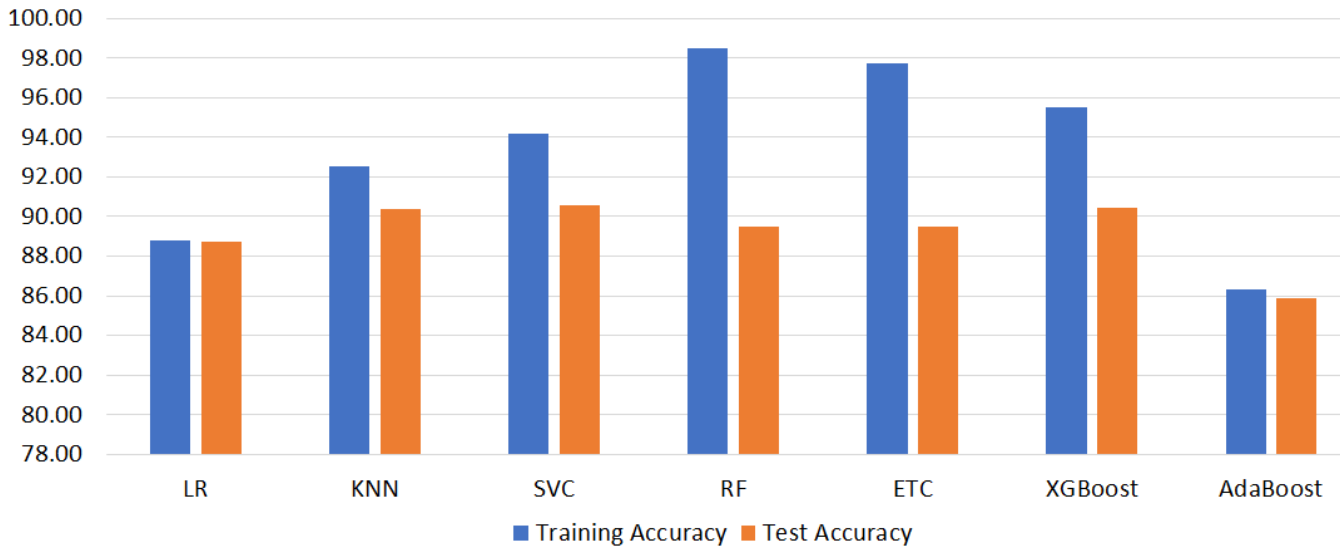

Figure 3-5: Training and Testing Accuracy results of the classifiers on original and adversarial samples generated at CW Attack

In 3.5, numerous classifier performances are described in detail. KNN performed slightly better, achieving training accuracy of 92.52%, validation accuracy of 90.36%, test accuracy of 90.39%, and test F-Score of 0.90, compared to LR, which achieved 88.80%, 88.73%, 88.73%, and 0.89 respectively. 3-5 gives a comparison of Training and Testing accuracy of the classifiers. With a training accuracy of 94.17%, a validation accuracy of 90.72%, a test accuracy of 90.57%, and a test F-Score of 0.90, SVC performed even better. With a training accuracy of 98.50%, the RF classifier performed admirably. Still, its validation and test accuracies were only marginally better, at 89.40% and 89.46%, respectively, with a test F-Score of 0.89.

With a training accuracy of 97.70%, validation accuracy of 89.46%, test accuracy of 89.49%, and test F-Score of 0.89. ETC performed similarly to the RF classifier. With a training accuracy of 95.54%, a validation accuracy of 90.39%, a test accuracy of 90.45%, and a test F-Score of 0.90, XGBoost demonstrated a solid balance between training and testing. AdaBoost has lesser performance metrics with a training

accuracy of 86.32 validation accuracy of 86.30%, test accuracy of 85.86%, and a test F-Score of 0.86. In general, there were observable differences in the training accuracies of classifiers, even though most of them had comparable test accuracies and F-scores. These variations should be considered when choosing a classifier since they may affect their capacity to generalize to new data.

Classifier	**Training Accuracy**	**Validation Accuracy**	**Test Accuracy**	**Test F-Score**
LR	97.37	97.32	97.51	0.95
KNN	97.99	97.49	97.64	0.95
SVC	98.04	97.82	97.95	0.96
RF	98.90	98.05	98.24	0.97
ETC	98.55	97.91	98.09	0.96
XGBoost	98.73	98.09	98.29	0.97
AdaBoost	95.59	95.58	95.81	0.92

Table 3.6: Training, Validation, and Testing results for various classifiers on original and adversarial samples generated at MI-Attack

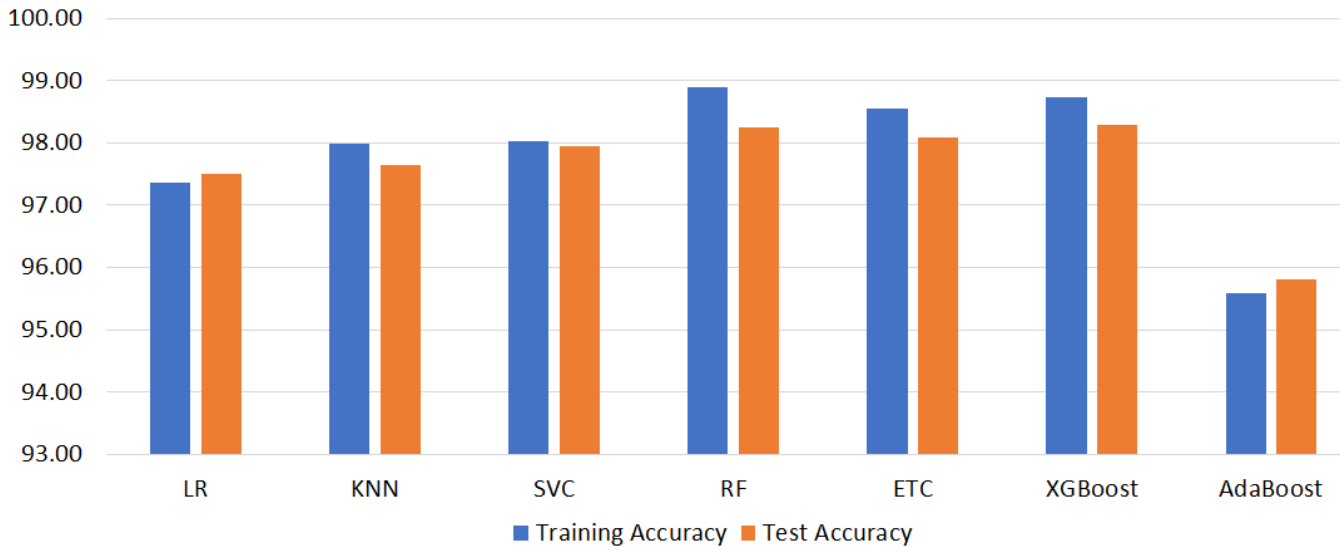

Figure 3-6: Training and Testing Accuracy results of the classifiers on original and adversarial samples generated at MI Attack

This 3.6 shows the results for seven different classifiers, each trained using both adversarial samples generated by MI and original samples. 3-6 gives a comparison of Training and Testing accuracy of the classifiers. With a Test F-Score of 0.95, the LR classifier achieved a training accuracy of 97.37%, validation accuracy of 97.32%, and test accuracy of 97.51%. With a similar Test F-Score of 0.95, K-Nearest Neighbors (KN) obtained greater training accuracy of 97.99%, validation accuracy of 97.49%, and test accuracy of 97.64%. With a training accuracy of 98.04%, validation accuracy of 97.82%, and test accuracy of 97.95%, the SVC classifier advanced further. A bit higher, at 0.96, was the Test F-Score.

With RF achieving the maximum training accuracy of 98.90%, validation accuracy of 98.05%, and test accuracy of 98.24%, the RF and ET classifiers performed better than expected. The Test F-Score, at 0.97, was also the highest. With a Test F-Score of 0.96, the ET classifier reported a training accuracy of 98.55%, validation accuracy of 97.91%, and test accuracy of 98.09%. With a training accuracy of 98.73%, validation

accuracy of 98.09%, and test accuracy of 98.29%, the XGBoost classifier performed similarly to RF, ETC. It received a Test F-Score of 0.97 as well. Last, the AdaBoost classifier returned accuracy values for training, validation, and testing of 95.59%, 95.58%, and 95.81%, respectively. Its Test F-Score of 0.92 was the lowest of any classifiers.

Chapter 4

Classifier Development: Design, Training, and Predictions

4.1 Performance Comparison of Classifiers for Defense Mechanism

By serving as a defense mechanism that differentiates between original and adversarial samples, classifiers are crucial to an entire architecture. By acting as a gatekeeper to the primary Android malware detection model, these classifiers ensure that only "clean" or original samples are processed further. This project uses several different classifiers (LR, KN, SVC, RF, ETC, XGBoost, and AdaBoost) [3], each using a different ML algorithm, 4-2 shows the image of the classifiers, training with both adversarial and original samples. These classifiers can distinguish between the two, given that they trained on a dataset that included original and adversarial

samples [47].

Once trained, the classifiers divide the test set's data into two groups: original and adversarial. The important part is that only the data identified by these classifiers as "original" are forwarded to the primary model for additional processing and predictions. This approach reduces the possibility that adversarial samples would affect the performance of the primary model, increasing its robustness and reliability in recognizing Android malware.

The effectiveness of these classifiers is crucial since it directly affects the system's ability to detect malware on Android devices as an entire system. Each classifier's performance can be compared using a range of measures (Training Accuracy, Test Accuracy, FPR, FNR, and F-Measure) [54], allowing one to choose which classifier(s) would be most suited for further integration with the primary model. These classifiers defend against adversarial attacks, preserving the accuracy of the primary model's malware detection while protecting the integrity of the data it processes. **??** shows the performance comparison of different classifiers [52].

Classifier	**Training Accuracy**	**Test Accuracy**	**FPR**	**FNR**	**F-measure**
LR	94.85	94.86	4.05	6.24	0.95
KNN	72.31	66.77	0.07	66.55	0.50
SV	99.17	98.60	1.09	1.72	0.99
RF	99.89	97.96	2.06	2.02	0.98
ETC	99.48	97.82	2.47	1.89	0.98
XGBoost	99.52	98.78	0.83	1.61	0.99
AdaBoost	91.36	91.65	3.03	13.70	0.91

Table 4.1: Performance comparison of different classifiers.

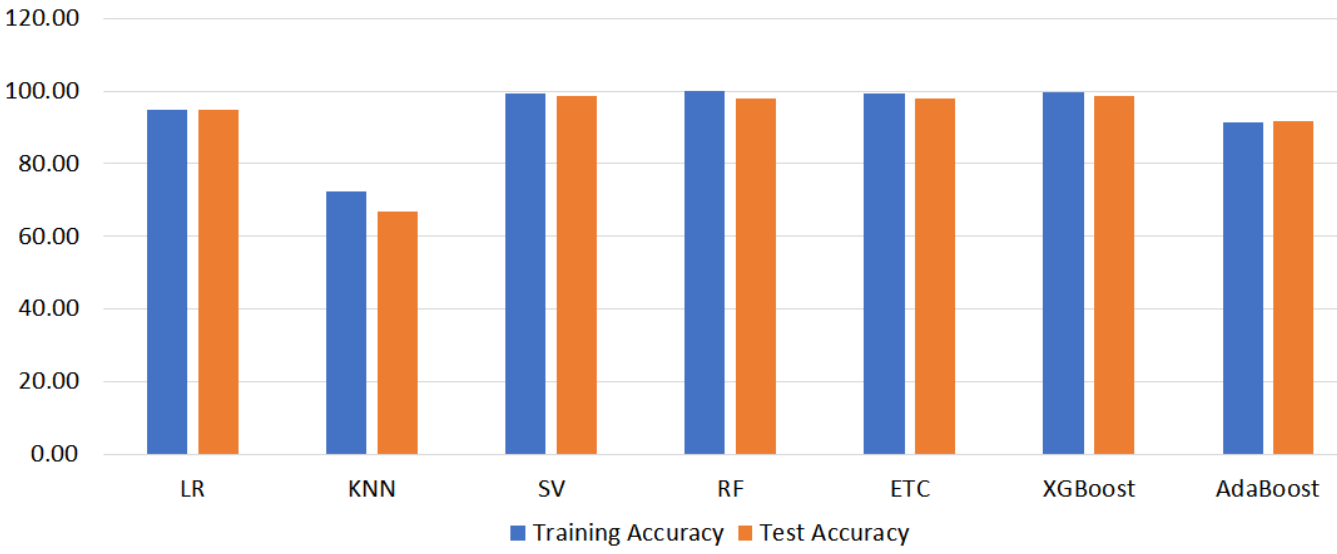

Figure 4-1: Comparison of Training and Testing Accuracy results of the different classifiers

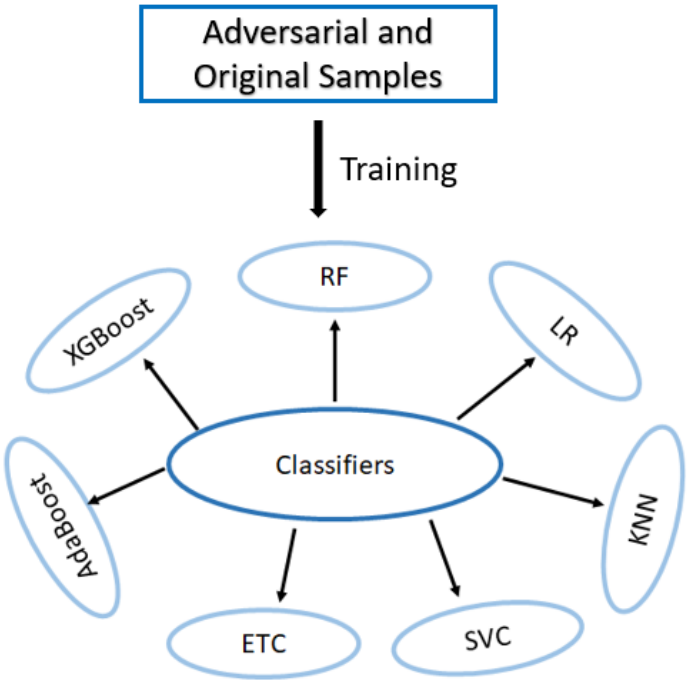

Figure 4-2: Training the classifiers with original and adversarial samples

The results of the classifiers used as a defense mechanism against adversarial attacks is presented in 4.1. The LR classifier demonstrated a consistent and robust predictive performance on both visible (training) and unseen (test) data, achieving training and test accuracy of 94.55% and 94.60%, respectively. 4-1 shows the comparison chart of the training and testing accuracy of different classifiers. It kept its FPR and FNR at comparatively low levels of 4.44% and 6.38%, respectively, indicating that it struck a decent balance in reducing both false alarms and misses. The high harmonic means of precision and recall shown by the F-measure of 0.95 further attests to its remarkable performance. When compared to other classifiers, the K-Nearest Neighbors (KNN) classifier performs significantly worse, with training accuracy of 73.48% and test accuracy of 68.02%. The model has a very low FPR, which is impressive at 0.05%. It has a high FNR of 64.18%, though, which means there were a lot of missed detections. The F-measure of 0.53, which is significantly lower than those of other classifiers, reflects this. The Random Forest (RF) and Support Vector (SV) classifiers outperform the competition, obtaining above 99% training and 97% test accuracy. They continue to have low FPR and FNR scores, indicating good class separation. Both have an F-measure of 0.98, which shows almost perfect recall and precision. With training accuracy above 99%, test accuracy around 98%, and an F-measure of 0.98, the Extra Trees (ET) and XGBoost (XG) classifiers perform similarly to the SV and RF classifiers. These classifiers handle adversarial attacks effectively overall, with ET having a little higher FPR than SV and RF and XG having a significantly higher FNR. Last, the AdaBoost (AB) classifier performs worse, with training and test accuracy of 91.54% and 91.47%, respectively. Although it still has a fare FPR of 3.30 percent, the FNR is more significant than average at 13.8 percent, which indicates more misses. Despite being lower than the others, it shows a reasonable precision and recall balance with an F-measure of 0.91.

4.2 Classifier Workflow as a Defense Mechanism and Model Predictions

4-3 describes the workflow of classifier and model predictions.

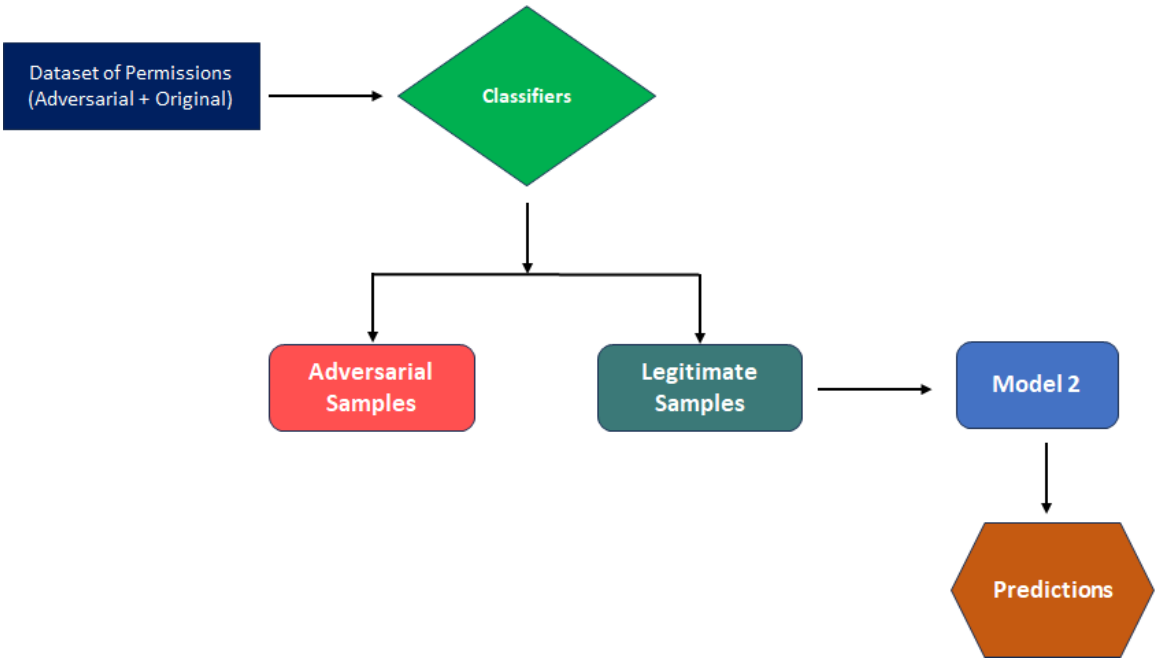

Figure 4-3: The Classifier's Workflow and Model Predictions

4.3 Integration with the Model

In our study, the classifiers are meant to improve the original model rather than replace it to make it more resilient to adversarial attacks. This section explains the steps taken to combine these classifiers with the original model to detect dangers that could occur [19]. The classifiers serve as a filter or gateway to the original model after being trained on a dataset that includes original and adversarial samples. These classifiers efficiently divide incoming samples into two groups: adversarial and original. The original samples are then saved to send to the primary model for

additional prediction tasks while the adversarial instances are marked [32].

The architecture will therefore be more resistant to adversarial attacks by integrating the classifiers with the original model. The classifiers work to prevent competing samples from changing the original model's prediction, preserving the precision and consistency of the model's output. However, it's crucial to remember that the success of this approach primarily depends on how well the classifiers can differentiate between the original and adversarial samples. To select the best classifiers for integration with the original model, it is essential to consider the performance comparison of these classifiers outlined in the previous section.

4.4 Performance of the Model

The original samples are provided to the model for additional predictions when the classifier distinguishes between the original and adversarial examples. In this stage, the model's performance on the original samples is evaluated, and its robustness to potential adversarial attacks is determined. The evaluation's findings indicate how well the model works with the original samples and whether it can continue to predict correctly even when faced with adversarial examples. These findings demonstrate the model's capacity to generalize and produce accurate predictions based on actual data. The term "classifier" in the context of the results presented refers to a particular model or technique used for classification. Every classifier is an individual model or algorithm trained to categorize samples into specific classes. The results below show how well the model—as determined by the particular classifier—performed in predicting the labels of the original samples. The previously mentioned metrics, including accuracy, FPR, FNR, and F-measure, thoroughly assess the model's effectiveness on the original samples. FPR and FNR offer information on the model's capacity to avoid making false positive and false negative predictions, while F-measure provides a fair evaluation

of precision and recall. Accuracy assesses the overall accuracy of the predictions.

Classifier	Accuracy	FPR	FNR	F-measure
LR	94.23	7.17	4.39	0.94
AdaBoost	90.91	11.45	6.74	0.91
ETC	96.68	3.16	3.48	0.97
KNN	96.22	4.44	3.12	0.96
RF	97.05	2.62	3.28	0.97
SVC	95.97	4.42	3.64	0.96
XGBoost	96.32	3.89	3.46	0.96

Table 4.2: Prediction Results of Different Models on Classifier-Identified Original Samples

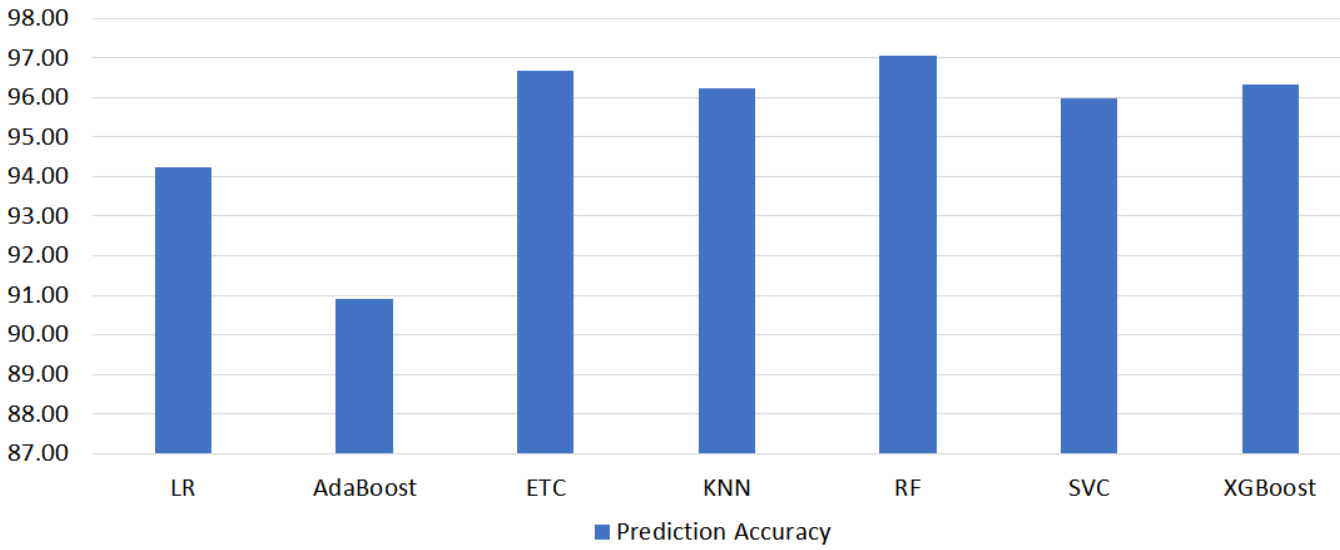

Figure 4-4: Comparison of Prediction Accuracy of the Models

The accuracy metric shows the percentage of samples that each classifier correctly classifies. 4.2 shows the prediction results of different models. A higher accuracy

demonstrates improved performance in correctly classifying positive and negative examples 4-4 compares models' accuracy. In this instance, the classifiers had accuracy levels between 90.91% and 97.05%. The FPR calculates the frequency with which negative samples are incorrectly deemed positive. A decreased FPR suggests an improved capacity to reduce the misclassification of negative examples. The FPR values that the classifiers achieved ranged from 2.62% to 11.45%. The FNR measures the frequency of incorrectly labeling positive samples as negative. A lower FNR suggests a better capacity to classify positive samples accurately. The FNR values the classifiers achieved ranged from 3.12% to 6.74%. The F-measure, which combines precision and recall measurements, evaluates the classifier's performance. It provides a balanced assessment by considering false positive and false negative rates. The classifiers' F-measure scores ranged from 0.91 to 0.97.

Chapter 5

Conclusion and Future Work

The study aimed to increase an Android malware detection model's resistance to adversarial attacks. A classifier was used as a first line of defense and successfully identified adversary samples apart from the original samples. As a result, the primary predictive model was protected from direct adversarial disturbance, enabling better predictions. The model was strengthened against attacks by training on adversarial samples. The trained model's resiliency against these adversarial attacks proved this method's effectiveness. The addition of the classifier system improved the model's performance and decreased the possibility of adversarial samples passing undetected. This demonstrates how such layered defensive measures might strengthen the model's resistance to adversary manipulation.

As this project progresses, several important areas will be the focus for enhancement and exploration. The model's exposure to various adversarial attacks is one of the main areas of interest. The model has only been trained and tested against a small number of adversarial techniques at this time. We seek to improve the robustness of

the model against unidentified or sophisticated adversarial samples, hence maintaining a high level of performance, by broadening the range of adversarial attacks used during model training. Exploring other defense mechanisms to make the model much stronger. Our evaluation was limited to binary datasets (0 or 1) and did not include other categorical or decimal data. There is potential for future research to extend these tests to explore other types of datasets.

References

[1] Nasiba Mahdi Abdulkareem and Adnan Mohsin Abdulazeez. Machine learning classification based on radom forest algorithm: A review. *International journal of science and business*, 5(2):128–142, 2021.

[2] L Abhishek. Optical character recognition using ensemble of svm, mlp and extra trees classifier. In *2020 International Conference for Emerging Technology (INCET)*, pages 1–4. IEEE, 2020.

[3] Prerna Agrawal and Bhushan Trivedi. Machine learning classifiers for android malware detection. In *Data Management, Analytics and Innovation: Proceedings of ICDMAI 2020, Volume 1*, pages 311–322. Springer, 2021.

[4] Mohammed K Alzaylaee, Suleiman Y Yerima, and Sakir Sezer. Dl-droid: Deep learning based android malware detection using real devices. *Computers & Security*, 89:101663, 2020.

[5] Brandon Amos, Hamilton Turner, and Jules White. Applying machine learning classifiers to dynamic android malware detection at scale. In *2013 9th inter-*

national wireless communications and mobile computing conference (IWCMC), pages 1666–1671. IEEE, 2013.

[6] Bela Amro. Malware detection techniques for mobile devices. *arXiv preprint arXiv:1801.02837*, 2018.

[7] Kshitiz Aryal, Maanak Gupta, and Mahmoud Abdelsalam. A survey on adversarial attacks for malware analysis. *arXiv preprint arXiv:2111.08223*, 2021.

[8] Moses Aprofin Ashawa and Sarah Morris. Analysis of android malware detection techniques: a systematic review. 2019.

[9] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International conference on machine learning*, pages 274–283. PMLR, 2018.

[10] Bhoopesh Singh Bhati and CS Rai. Ensemble based approach for intrusion detection using extra tree classifier. In *Intelligent Computing in Engineering: Select Proceedings of RICE 2019*, pages 213–220. Springer, 2020.

[11] Franziska Boenisch, Philip Sperl, and Konstantin Böttinger. Gradient masking and the underestimated robustness threats of differential privacy in deep learning. *arXiv preprint arXiv:2105.07985*, 2021.

[12] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials*, 18(2):1153–1176, 2015.

[13] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 3–14, 2017.

[14] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. Ieee, 2017.

[15] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. A survey on adversarial attacks and defences. *CAAI Transactions on Intelligence Technology*, 6(1):25–45, 2021.

[16] Mayank Arya Chandra and SS Bedi. Survey on svm and their application in image classification. *International Journal of Information Technology*, 13:1–11, 2021.

[17] Steven Chen, Nicholas Carlini, and David Wagner. Stateful detection of black-box adversarial attacks. In *Proceedings of the 1st ACM Workshop on Security and Privacy on Artificial Intelligence*, pages 30–39, 2020.

[18] Iyad Lahsen Cherif and Abdesselem Kortebi. On using extreme gradient boosting (xgboost) machine learning algorithm for home network traffic classification. In *2019 Wireless Days (WD)*, pages 1–6. IEEE, 2019.

[19] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[20] De-Cheng Feng, Zhen-Tao Liu, Xiao-Dan Wang, Yin Chen, Jia-Qi Chang, Dong-Fang Wei, and Zhong-Ming Jiang. Machine learning-based compressive strength prediction for concrete: An adaptive boosting approach. *Construction and Building Materials*, 230:117000, 2020.

[21] Samuel G Finlayson, John D Bowers, Joichi Ito, Jonathan L Zittrain, Andrew L Beam, and Isaac S Kohane. Adversarial attacks on medical machine learning. *Science*, 363(6433):1287–1289, 2019.

[22] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1322–1333, 2015.

[23] Dragoş Gavriluţ, Mihai Cimpoeşu, Dan Anton, and Liviu Ciortuz. Malware detection using machine learning. In *2009 International multiconference on computer science and information technology*, pages 735–741. IEEE, 2009.

[24] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[25] Florian Huber, Artem Yushchenko, Benedikt Stratmann, and Volker Steinhage. Extreme gradient boosting for yield estimation compared with deep learning approaches. *Computers and Electronics in Agriculture*, 202:107346, 2022.

[26] Olakunle Ibitoye, Rana Abou-Khamis, Ashraf Matrawy, and M Omair Shafiq. The threat of adversarial attacks on machine learning in network security–a survey. *arXiv preprint arXiv:1911.02621*, 2019.

[27] Jin Li, Lichao Sun, Qiben Yan, Zhiqiang Li, Witawas Srisa-An, and Heng Ye. Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics*, 14(7):3216–3225, 2018.

[28] Yannan Li, Jingbo Wang, and Chao Wang. Proving robustness of knn against adversarial data poisoning. In *CONFERENCE ON FORMAL METHODS IN COMPUTER-AIDED DESIGN–FMCAD 2022*, page 7, 2022.

[29] Jing Lin, Laurent L Njilla, and Kaiqi Xiong. Secure machine learning against adversarial samples at test time. *EURASIP Journal on Information Security*, 2022(1):1, 2022.

[30] Xiang Ling, Lingfei Wu, Jiangyu Zhang, Zhenqing Qu, Wei Deng, Xiang Chen, Yaguan Qian, Chunming Wu, Shouling Ji, Tianyue Luo, et al. Adversarial attacks against windows pe malware detection: A survey of the state-of-the-art. *Computers & Security*, page 103134, 2023.

[31] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[32] Chengzhi Mao, Ziyuan Zhong, Junfeng Yang, Carl Vondrick, and Baishakhi Ray. Metric learning for adversarial robustness. *Advances in neural information processing systems*, 32, 2019.

[33] Akshay Mathur, Laxmi Mounika Podila, Keyur Kulkarni, Quamar Niyaz, and Ahmad Y Javaid. Naticusdroid: A malware detection framework for android using native and custom permissions. *Journal of Information Security and Applications*, 58:102696, 2021.

[34] K Meenakshi and G Maragatham. A review on security attacks and protective strategies of machine learning. *Emerging Trends in Computing and Expert Technology*, pages 1076–1087, 2020.

[35] Pavas Navaney, Gaurav Dubey, and Ajay Rana. Sms spam filtering using supervised machine learning algorithms. In *2018 8th international conference on cloud computing, data science & engineering (confluence)*, pages 43–48. IEEE, 2018.

[36] Fred C Pampel. *Logistic regression: A primer.* Number 132. Sage publications, 2020.

[37] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learn-

ing. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[38] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE symposium on security and privacy (SP)*, pages 582–597. IEEE, 2016.

[39] Marek Pawlicki, Michal Choraś, and Rafał Kozik. Defending network intrusion detection systems against adversarial evasion attacks. *Future Generation Computer Systems*, 110:148–154, 2020.

[40] Tianrui Peng, Ian Harris, and Yuki Sawa. Detecting phishing attacks using natural language processing and machine learning. In *2018 ieee 12th international conference on semantic computing (icsc)*, pages 300–301. IEEE, 2018.

[41] Ashis Pradhan. Support vector machine-a survey. *International Journal of Emerging Technology and Advanced Engineering*, 2(8):82–85, 2012.

[42] Hemant Rathore, Sanjay K Sahay, Piyush Nikam, and Mohit Sewak. Robust android malware detection system against adversarial attacks using q-learning. *Information Systems Frontiers*, 23:867–882, 2021.

[43] Ishai Rosenberg, Asaf Shabtai, Yuval Elovici, and Lior Rokach. Adversarial machine learning attacks and defense methods in the cyber security domain. *ACM Computing Surveys (CSUR)*, 54(5):1–36, 2021.

[44] Bita Darvish Rouani, Mohammad Samragh, Tara Javidi, and Farinaz Koushanfar. Safe machine learning and defeating adversarial attacks. *IEEE Security & Privacy*, 17(2):31–38, 2019.

[45] Wenjie Ruan, Xinping Yi, and Xiaowei Huang. Adversarial robustness of deep learning: Theory, algorithms, and applications. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 4866–4869, 2021.

[46] Anjaneyulu Babu Shaik and Sujatha Srinivasan. A brief survey on random forest ensembles in classification model. In *International Conference on Innovative Computing and Communications: Proceedings of ICICC 2018, Volume 2*, pages 253–260. Springer, 2019.

[47] Lwin Khin Shar, Biniam Fisseha Demissie, Mariano Ceccato, and Wei Minn. Experimental comparison of features and classifiers for android malware detection. In *Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems*, pages 50–60, 2020.

[48] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[49] Aboozar Taherkhani, Georgina Cosma, and T Martin McGinnity. Adaboost-cnn: An adaptive boosting algorithm for convolutional neural networks to classify multi-class imbalanced datasets using transfer learning. *Neurocomputing*, 404:351–366, 2020.

[50] Tony Thomas, Athira P. Vijayaraghavan, Sabu Emmanuel, Tony Thomas, Athira P. Vijayaraghavan, and Sabu Emmanuel. Machine learning and cybersecurity. *Machine Learning Approaches in Cyber Security Analytics*, pages 37–47, 2020.

[51] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017.

[52] Chenyue Wang, Linlin Zhang, Kai Zhao, Xuhui Ding, and Xusheng Wang. Advandmal: Adversarial training for android malware detection and family classification. *Symmetry*, 13(6):1081, 2021.

[53] Lishan Wang. Research and implementation of machine learning classifier based on knn. In *IOP Conference Series: Materials Science and Engineering*, volume 677, page 052038. IOP publishing, 2019.

[54] Yuandi Wang. Performance comparison of android malware detection methods. In *Journal of Physics: Conference Series*, volume 1827, page 012176. IOP Publishing, 2021.

[55] Jing Wu, Mingyi Zhou, Ce Zhu, Yipeng Liu, Mehrtash Harandi, and Li Li. Performance evaluation of adversarial attacks: Discrepancies and solutions. *arXiv preprint arXiv:2104.11103*, 2021.

[56] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[57] Yingchao Yu, Xueyong Liu, and Zuoning Chen. Attacks and defenses towards machine learning based systems. In *Proceedings of the 2nd International Conference on Computer Science and Application Engineering*, pages 1–7, 2018.

[58] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824, 2019.

[59] Xiaonan Zou, Yong Hu, Zhewen Tian, and Kaiyuan Shen. Logistic regression model optimization and case analysis. In *2019 IEEE 7th international conference on computer science and network technology (ICCSNT)*, pages 135–139. IEEE, 2019.

Appendix A

Source Code Snippets for Adversarial Attacks

A.1 CW

```
def caligari_wager_attack(X, y, model,
epsilon=0.3, perturb_factor=0.76,
num_iterations=300, learning_rate=0.3):
    processed_samples = []
    accuracies = []
    false_positive_rates = []
    false_negative_rates = []
    fmeasures = []

    def obj_func(perturbation, x, y_target, model, epsilon):
```

```
        x_adv = x + perturbation
        x_adv = np.round(x_adv)
        x_adv = np.clip(x_adv, 0, 1)
        y_pred = model.predict(x_adv.reshape(1, -1))
        y_pred = int(y_pred[0])
        loss = 0 if y_pred == y_target else 1
        constraint = np.sum(np.abs(perturbation)) - epsilon
        penalty = 1e3 if constraint > 0 else 0
        return loss + penalty
    X_adv = np.copy(X)
    perturbations = np.zeros_like(X)
    for i in range(X.shape[0]):
        x = X[i]
        y_target = int(y[i])


        bounds = [(None, None) for _ in range(X.shape[1])]
        initial_perturbation = np.random.choice([-1, 0, 1], size=X.shape[1],
        p=[epsilon / 2, 1 - epsilon, epsilon / 2])

        options = {
            'maxiter': num_iterations,
            'initial_tr_radius': learning_rate
        }


        attack_result = scipy.optimize.minimize
```

```
        (obj_func, initial_perturbation, args=(x, y_target, model, epsilon),
        method='trust-constr', bounds=bounds, options=options)
        perturbation = attack_result.x
        perturbations[i] = perturbation
        x_adv = x + perturb_factor * perturbation
        X_adv[i] = np.round(x_adv)
        X_adv[i] = np.clip(X_adv[i], 0, 1)


        if (i + 1) % 100 == 0:
            y_pred_adv = model.predict(X_adv[:i + 1])
            acc, fpr, fnr, fmeasure = calculate_performance_metrics
            (y[:i + 1], y_pred_adv)
            processed_samples.append(i + 1)
            accuracies.append(acc * 100)
            false_positive_rates.append(fpr * 100)
            false_negative_rates.append(fnr * 100)
            fmeasures.append(fmeasure * 100)
            print(f"PROCESSED {i + 1} SAMPLES")
            print(f"Accuracy after processing {i + 1} samples: {acc * 100:.2f}%")
            print(f"False positive rate after processing {i + 1}
            samples: {fpr * 100:.2f}%")
            print(f"False negative rate after processing {i + 1}
            samples: {fnr * 100:.2f}%")
            print(f"F-measure after processing {i + 1}
            samples: {fmeasure * 100:.2f}%")
```

```
    return X_adv, perturbations, processed_samples,
    accuracies, false_positive_rates,
    false_negative_rates, fmeasures
```

A.2 MI

```
def invert_model(model, target_label, max_iter=1000,
learning_rate=0.01, threshold=0.5):
    x_inverted = np.random.uniform(0, 1, size=X.shape[1])
    for _ in range(max_iter):
        x_inverted = np.clip(x_inverted, 0, 1)
        prediction = model.predict_proba(x_inverted.reshape(1, -1))
        gradient = prediction[:, target_label] - threshold
        x_inverted -= learning_rate * gradient
    return x_inverted

processed_samples = []
accuracies = []
false_positive_rates = []
false_negative_rates = []
fmeasures = []

# Generate adversarial samples using the
Model Inversion Attack for the entire dataset
X_adv = np.zeros_like(X)
for i, target_label in enumerate(y):
```

```
    X_adv[i] = invert_model(model, target_label)
    if (i + 1) % 100 == 0:
        y_pred_adv = model.predict(X_adv[:i + 1])
        acc_after, fpr_after, fnr_after,
        fmeasure_after = calculate_performance_metrics(y[:i + 1], y_pred_adv)
        processed_samples.append(i + 1)
        accuracies.append(acc_after * 100)
        false_positive_rates.append(fpr_after * 100)
        false_negative_rates.append(fnr_after * 100)
        fmeasures.append(fmeasure_after * 100)
        print(f"PROCESSED {i + 1} SAMPLES")
        print(f"Accuracy AFTER processing {i + 1} samples: {acc_after * 100:.2f}%")
        print(f"False positive rate AFTER processing {i + 1}
        samples: {fpr_after * 100:.2f}%")
        print(f"False negative rate AFTER processing {i + 1}
        samples: {fnr_after * 100:.2f}%")
        print(f"F-measure AFTER processing {i + 1}
        samples: {fmeasure_after * 100:.2f}%")
```

Appendix B

Source Code Snippets for Training Classifiers

```
def modelling(key, classifier):
    roc_lst = []

    train_start = time()
    classifier.fit(X_train, y_train)
    train_end = time()
    trn_time = train_end - train_start

    train_acc = classifier.score(X_train, y_train)*100
    # calculating the training accuracy
    detction_start = time()
    y_pred = classifier.predict(X_test)
```

```
    detection_end = time()
    tst_time = detection_end - detction_start

    cm = confusion_matrix(y_test, y_pred)
    cm_dict[key] = cm

    probs = classifier.predict_proba(X_test)
    probs = probs[:, 1]
    roc_lst.append([y_test, probs])
    roc_dict[key] = roc_lst

    tst_acc = accuracy_score(y_test, y_pred)*100
    fScore = f1_score(y_test, y_pred)

    val = cross_val_score(estimator = classifier,
    X = X_train, y = y_train, cv = 10, n_jobs = -1)
    val_acc = val.mean()*100

    return train_acc, val_acc, tst_acc, fScore, trn_time, tst_time, cm_dict[key]

dataset = pd.read_csv("binarydata.csv")
#dataset = shuffle(dataset)

# Load the adversarial samples
adversarial_dataset = pd.read_csv("adversarial_samples_29333.csv")

# Concatenate the original and adversarial samples
```

```
combined_dataset = pd.concat([dataset, adversarial_dataset])
# Shuffle the combined dataset
combined_dataset = shuffle(combined_dataset)
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
models = {
        'LR': LogisticRegression(random_state = 0, n_jobs = -1),
        'KN': KNeighborsClassifier
        (n_neighbors=5, p=2, metric = 'minkowski'),
        'SV': SVC(kernel = 'rbf', random_state=0,probability=True),
        'RF': RandomForestClassifier
        (criterion='gini', random_state = 0, n_estimators=10, n_jobs = -1),
        'ET': ExtraTreesClassifier
        (criterion='gini', min_samples_leaf = 2, n_estimators=5, n_jobs = -1),
        'XG': XGBClassifier(n_jobs = -1),
        'AB': AdaBoostClassifier(learning_rate=0.1, n_estimators=70),
        }
```

Appendix C

Source Code Snippets for Classifier as a defense mechanism

```
# Shuffle the combined dataset
combined_dataset = shuffle(combined_dataset)

# Separate the features (X) from the target variable (y)
X = combined_dataset.iloc[:,:-1].values
y = combined_dataset.iloc[:,-1].values

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_stat

# Define your classifiers
```

```
classifiers = {
    'LR': LogisticRegression(random_state = 0, n_jobs = -1),
    'KN': KNeighborsClassifier(n_neighbors=5, p=2, metric = 'minkowski'),
    'SV': SVC(kernel = 'rbf', random_state=0, probability=True),
    'RF': RandomForestClassifier(criterion='gini', random_state = 0, n_estimators=10
    'ET': ExtraTreesClassifier(criterion='gini', min_samples_leaf = 2, n_estimators=
    'XG': XGBClassifier(n_jobs = -1),
    'AB': AdaBoostClassifier(learning_rate=0.1, n_estimators=70),
}

# Create an empty DataFrame to store the results
results_df = pd.DataFrame(columns=['Classifier', 'Training Accuracy', 'Test Accuracy

# Train each classifier and evaluate its performance
for key, classifier in classifiers.items():
    classifier.fit(X_train, y_train)

    # Calculate training accuracy
    y_train_pred = classifier.predict(X_train)
    train_accuracy = accuracy_score(y_train, y_train_pred) * 100

    # Calculate test metrics
    y_pred = classifier.predict(X_test)
    test_accuracy, fpr, fnr, fmeasure = calculate_performance_metrics(y_test, y_pred
    print(f"{key} classifier: Training accuracy = {train_accuracy:.2f}%, Test accura

    # Save the classifier
```

```
joblib.dump(classifier, f"D:/Attack Defense/Multiple classifiers/New_Integration
print(f"{key}_classifier is dumped")

# Append the results to the results_df
results_df = results_df.append({
    'Classifier': key,
    'Training Accuracy': train_accuracy,
    'Test Accuracy': test_accuracy,
    'FPR': fpr,
    'FNR': fnr,
    'F-measure': fmeasure},
    ignore_index=True)

# Save the results to a CSV file
results_df.to_csv('D:/Attack Defense/Multiple classifiers/New_Integration/classi
print("Results of classifier are printed")
```