AUTOMATIC DETECTION OF INJECTON ATTACK IN HTTP REQUESTS

A Thesis Presented to the Department of Computer Science
African University of Science and Technology

In Partial Fulfilment of the Requirements for the Degree of

MASTER of Science

By

SODIQ IDOWU IBRAHIM
40577

Abuja, Nigeria

June, 2019.

CERTIFICATION

This is to certify that the thesis titled ‘Automatic detection of injection attack in HTTP requests’ submitted to the school of postgraduate studies, African University of Science and Technology (AUST), Abuja, Nigeria for the award of the Master's degree is a record of original research carried out by Sodiq Idowu I. in the Department of Computer Science.

African University of Science and Technology [AUST]
Knowledge is Freedom

APPROVAL BY

Supervisor

Surname: ODUMUYIWA

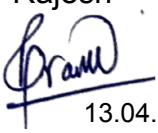
First name: Victor

Signature: 

The Head of Department

Surname: Prasad

First name: Rajesh

Signature: 
13.04.2020

ABSTRACT

Malicious user requests pose a vicious threat to backend devices which execute them; more so, could result in the compromise of other user accounts, exposing them to theft and blackmail. It becomes imperative to sanitize such requests before they are treated by the servers as access to a single malicious request is enough to cause a disaster. A number of authors suggest that sanitizing models built on support vector machines guarantee optimum classification of malicious from non-malicious requests. In this work, we have been able to establish that the use of ensemble learner provides a better performance, especially when associated with a strong classifying tool like decision tree.

DEDICATION

I dedicate this work to God Almighty for his immense grace and mercies.

ACKNOWLEDGEMENT

I want to express my sincere appreciation to my supervisor, Dr. Victor Odumuyiwa for his immense support and guidance over the period of the research. I understand that you make out time of your busy schedule, at grave inconvenience just to nudge in the right direction.

My sincere appreciation to Prof. Amos David and Dr. Rajesh Prasad for your very many sacrifices during the period of this program. I am indeed indebted to the many faculty members, some of whom crossed the high seas just to deliver knowledge. You are held in high esteem.

My profound gratitude goes to my lovely mother, Mrs. Adenike Sodiq, to my grandmother, Mrs Agbeyegbe Joseph, my siblings – Yewande, Taiwo, Tosin - and the amazing “super cousin”, Olasoju Olajumoke. Thanks for the very many sacrifices made up to this point.

To the intellectuals in the Department of Computer Science and Engineering, I appreciate the lovely memories, kind gestures, challenges tackled, and simply being the best classmates ever. I wish you success in your future endeavours.

TABLE OF CONTENTS

CERTIFICATION ii
ABSTRACT v
DEDICATION vi
ACKNOWLEDGEMENT vii
TABLE OF CONTENTS viii
LIST OF FIGURES x
LIST OF TABLES xi
LIST OF EQUATIONS xii
LIST OF ABBREVIATIONS xiii
CHAPTER ONE 1
INTRODUCTION 1
1.1 Research Background 1
1.2 Problem Statement 3
1.3 Research Aim and Objectives 4
1.4 Limitation of Study 4
1.5 Chapterization 4
CHAPTER TWO 6
LITERATURE REVIEW 6
2.1 Basic Concepts and Terminologies 6
2.2 Neural Network 7
2.2.1 Deep Neural Network 7
2.2.2 Artificial Neural Network 8
2.2.2 Propagation Function 9
2.2.3 Bias 9
2.2.4 Activation Function 10
2.4 Support Vector Machine 14
2.5 Existing Work 15
CHAPTER THREE 19
METHODOLOGY 19
3.1 Concept of Classification Technique 19
3.2 Software Design Phase 19
3.3 Proposed Framework 20
3.4 Intrusion Detection Framework 21
3.5 Dataset Description 22

3.6 Data Transformation 23
3.6.1 Tokenizaton 23
3.6.2 Vectorization 23
3.6.3 Feature Extraction 23
3.7 Ensemble Learning 23
3.7.1 The ensemble Framework 24
3.7.2 Training Algorithm for Ensemble Learning 25
3.8 Performance Evaluation Metrics 30
3.8.1 Confusion Matrix 30
3.8.2 Measures 31
CHAPTER FOUR 33
RESULTS AND DISCUSSIONS 33
4.1 Presentation of Results 33
4.1.1 Reading Text File 33
4.1.2 Data Preprocessing 35
4.2 Classification 38
4.2.1 Support Vector Machine 38
4.2.2 Ensemble using Support Vector Machine as Base Classifier 38
4.2.3 DecisionTree Classifier 39
4.2.4 Ensemble using DecisionTree as Base Classifier 40
4.3 Result of Performance 41
4.3.1 Support Vector Machine 42
4.3.2 Decision Tree 43
4.3.3 Ensemble with Decision Tree base classifier 44
4.3.4 Ensemble with SVM base classifier 45
CHAPTER FIVE 46
SUMMARY, CONCLUSION AND FUTURE WORK 46
5.1 Summary 46
5.2 Conclusion 47
5.3 Future Work 47
REFERENCE 49

LIST OF FIGURES

Figure 2. 1 Single Layer Perceptron 9
Figure 2. 2 Sigmoid Function 11
Figure 2. 3 Hyperbolic Tangent Function 12
Figure 2. 4 Recursive Neural Tensor Networks 13
Figure 2. 5 A Model of Recursive Neural Network 14
Figure 2. 6 Support Vector Machine Model 15
Figure 3. 1 Machine Learning Paradigm 20
Figure 3. 2 Intrusion Detection Framework 21
Figure 3. 3 Sample of Dataset 22
Figure 3. 4 Ensemble Learning Model 24
Figure 3. 5 Model for Dependent Ensemble Method 26
Figure 3. 6 Model for Independent Ensemble Method 28
Figure 3. 7 Stacking Ensemble Model 30
Figure 4.1 Code to read Data into Model 33
Figure 4.2 Pandas Formatted Data Sample 34
Figure 4.3 Code to Transform Label into Binary form 34
Figure 4.4 Transformed Label 35
Figure 4.5 Code for Tokenization 36
Figure 4.6 Code for Vectorization 36
Figure 4.7 Code for Feature Selection 37
Figure 4.8 Numerical Equivalent of Dataset 37
Figure 4.9 Code for Linear SVC Classifier 38
Figure 4.10 Code for Ensemble with SVC Base 39
Figure 4.11 Accuracy Report for Ensemble with SVC 39
Figure 4.12 Function Call for Decision Tree Classifier 40
Figure 4.13 Accuracy Report for Decision Tree 40
Figure 4.14 Code for Ensemble with Decision Tree classifier 41
Figure 4.15 Accuracy Report for Ensemble with DecisionTree 41

LIST OF TABLES

Table 3. 1 Confusion Matrix 31
Table 4. 1 Confusion Matrix for Support Vector Machine……42
Table 4. 2 Confusion Matrix for Decision Tree 43
Table 4. 3 Confusion Matrix for Ensemble with DecisionTree 44
Table 4. 4 Confusion Matrix for Ensemble with Support Vector Machine..... 45

LIST OF EQUATIONS

Equation 1 9
Equation 2 9
Equation 3 10
Equation 4 10
Equation 5 11
Equation 6 12
Equation 7 12
Equation 8 23

LIST OF ABBREVIATIONS

HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
SVM	Support Vector Machine
RNTN	Recursive Neural Tensor Network
CNN	Convoluted Neural Network
ReLU	Rectified Linear Unit
OCPAD	One Class Naïve Bayes Classifier for Payload-based Anomaly Detection
AP	Affinity Propagation
CBR	Case-Based Reasoning
SQL	Structured Query Language
AI	Artificial Intelligence
TF-IDF	Term Frequency – Inverse Document Frequency
NIDS	Network Intrusion Detection System
AIIDA-SQL	Adaptive Intelligent Intrusion Detector Agent for SQL injection attacks

CHAPTER ONE

INTRODUCTION

1.1 Research Background

The web is considered as the future of businesses. In the 21st century and beyond, it is believed that our lives will gravitate around the Internet. From international trade, internet banking, financial markets, academia, medicine to social interaction (Nagpal, Chauhan, & Singh, 2017), the world would further shrink into a global Web. The Web has simply come to stay.

The sheer size of the internet's potential has resulted in an industry of unscrupulous activities aimed at compromising integrity of unsuspecting users, making them victims of theft to personal data (Caballero, Grier, Kreibich, Paxson, & Berkeley, 2011) and to blackmail (Threats & Young, 1996; Young & Yung, 2017). In recent time, critical national infrastructures and control systems have also been either compromised or targeted (Cherepanov, 2017; Karnouskos, 2011). Several methods are being employed by such pilferers and targeted at varying devices in the chain of transmission. Malicious URL's, through intrusion tops the list of attack methodology and have since become the primary mechanism used to perpetuate such cyber-crimes (Le, Pham, Sahoo, & Hoi, 2018).

Over the years, scholars, with the use of machine learning tools have made significant progress with solutions intended at addressing such lapses; most notably machine learning tools have been adopted for predicting (Shen, Mariconti, Vervier, & Stringhini, 2018) and preventing (Haas, 1953; K. Wang & Stolfo, 2007) network intrusion, reverse engineering (Chua, Shen, Saxena, & Liang, 2017) and detecting malicious codes (Arp,

Spreitzenbarth, Malte, Gascon, & Rieck, 2013; Huang & Stokes, 2016; Jordaney et al., 2017). This resulted in two major paradigms in intrusion detection systems (IDS). First being Signature-based approaches which are effective in detecting known attacks with a low false positive (FP) rate. They, however, fail in the detection of unknown attacks – firewall and antivirus applications fall under this category. In contrast, Anomaly based approach detects unknown attack, but with high False Positive rates (Dong et al., 2018). This work focuses on a comparative analysis of machine learning tools to better understand which is likely to produce best performing model capable of detecting injection attacks.

Steps taken to extract meaningful information from a dataset are as follows:

i. Preprocessing: It involves cleaning, feature extraction, feature selection and dimensionality reduction.
ii. Clustering: It is an unsupervised learning technique that involves grouping a set of related data into cluster and classes.
iii. Classification: It is a supervised machine learning technique where decisions and predictions are made based on past data; which also involves both the provided input and output data.

In this research work, focus is on classification technique as datasets used have been initially labeled.

1.2 Problem Statement

Malicious URLs is one of the most potent ways by which cyber-crime is perpetuated.

They are capable of hosting unsolicited content and success is achieved when un-sanitized and unvalidated user inputs are permitted to interact with backend devices (Le et al., 2018; Uwagbole, Buchanan, & Fan, 2017).

Of more pressing concern is the fact that while developers of software infrastructure understand the challenges of security threats and make efforts to address such, some developers of web-based applications either have little of such skills or are outright ignorant of the challenges. These developers basically focus on functionality while unwittingly ignoring the security of their clients. Furthermore, a revealed report suggested that up to 97 percent of websites and web applications were vulnerable to one form of attack or another (Halfond & Orso, 2005). Techniques to safely classify malicious URLs from non-malicious ones have since been proposed by several scholars. However, many of such techniques are either inefficient, impracticable or not dynamic enough to identify and handle the ever-changing malicious threats (W. Wang et al., 2014). Moreover, traditional methods used to detect attacks, such as signature matching, no longer provide as much protection in the face of changing vectors. Unknown threats which have no familiar existing signatures remain at large. The problem is made worse with obscure codes (in the body of codes) that obstructs static signature matching (Moser, Kruegel, & Kirda, 2007; Wressnegger, 2018).

This thesis is aimed at proposing a model that automatically detects injection attacks on http requests using machine learning tools. A comparative analysis of various ML tools is done to determine the probable tool capable of achieving the best performing model.

For machine learning classification, Dataset containing malicious and non-malicious URLs were fetched, tokenized to remove unnecessary and special characters, vectorized to transform the corpus into numerical format in matrix shape, after which support vector machine, logistic regression and decision tree classifications were carried out.

1.3 Research Aim and Objectives

Objective of the study is to identify machine learning tools that would guarantee optimum classification where malicious URLs compromised with payloads can be easily identified. The research work makes use of two different datasets composed of malicious and non-malicious URLs and were fetched from firewall of various organizations. The main objectives of the research are:

i. Comparative analysis of various tools to determine which would best achieve an intrusion detection model.
ii. Compare results with those got using regular machine learning classifications.

1.4 Limitation of Study

This study involves proposing a model that will efficiently detect attacks in http requests. The study employed the use of support vector machines, ensemble learning and decision tree classifiers. Performance of the system was confirmed using the confusion matrix.

1.5 Chapterization

The work is distributed over five chapters with each highlighting different topics and subtopics.

Chapter 1 introduces the challenges of http intrusion, problem statement and objective of the thesis.

Chapter 2 discusses the basic concept of machine learning, deep neural networks and a critical review of literature.

Chapter 3 introduces the methodology involved in the model set-up, hardware requirements, processes in pre-processing/ data transformation and introduction to classifier performance metrics.

Chapter 4 presents code snippets, discuss results and makes comparison with performance of relevant machine learning tools.

Chapter 5 brings the work to conclusion and makes recommendations for future work.

CHAPTER TWO

LITERATURE REVIEW

This chapter introduces basic concepts and terminologies employed in this work: methods, preprocessing, feature selection, data classification and lastly literature review of previous published articles.

2.1 Basic Concepts and Terminologies

Machine learning is a technology that allows computers to learn from past experiences in the form of data passed across to them (H. Wang, Ma, & Zhou, 2009). It is perceived as the domain of Artificial Intelligence (AI) and finds application in such innovation as email filtering, robotics, voice recognition and computer vision. ML focuses on the development of computer programs that can access data, learn, unlearn and re-learn. Its performance is dependent on the quality of information/dataset provided to the system. The goal of machine learning algorithm is to construct a model such that new data can be labeled effortlessly. An inducer constructs an algorithm for a particular model and a classifier is an instance of such model (Rokach, 2010). Methods involved in Machine Learning are: Supervised, Unsupervised, Semi-supervised and Reinforcement learning methods. Supervised method applies what has been learned in the past from labeled examples to predict future events. Unsupervised method however is employed when the training data is neither classified nor labeled. It attempts to propose how a system can draw a function to describe a hidden structure for an unlabeled data – the algorithm is left to itself to discover structures in the data. Semi-supervised method lies in-between the regions of supervised and unsupervised domain since they use both labeled and unlabeled data for training. Lastly, Reinforcement

method learns when it interacts with its environment by producing actions that maximize rewards and penalty for committed errors.

Furthermore, Machine Learning can be categorized based on the desired output. Important output categories are: Classification, Regression and Clustering. In classification, inputs are divided into classes, and the learning model must be able to assign test inputs into any of the available classes. It is typically employed in a supervised learning environment and finds application in email classification of messages as either spam or otherwise. Regression also operates in a supervised learning domain and is applied where the outputs are continuous rather than discrete, whereas, Clustering operates in the unsupervised learning domain and is used by the model to classify inputs into groups.

2.2 Neural Network

Neural nets are algorithms which were inspired by the physical structure of the mammalian brain. Like the brain, the interpret data through some kind of perceptron, labeling and clustering. Their inputs use numerical values alongside some complex mathematical techniques, unlike the brain which fires neurons based on some chemical reactions. Neural nets have shown promise in performance and are continually implemented in image, voice and pattern recognition.

2.2.1 Deep Neural Network

Deep Neural learning is a machine learning technique that attempts to teach computers those activities that naturally appear to be in human domain. It is the technology behind self-driving cars, enabling them to recognize the stop signal and distinguish a pedestrian from a random object. It also finds application in consumer devices like

phones, tablets and television. In deep neural networks, a computer model attempts to perform classification tasks on images, text and sound. They have since grown to achieve state-of-the-art performance, even outperforming human on occasions. Deep learning methods have their roots in neural networks and have an extended hidden layer; a justification for the term “deep”. Traditional neural networks have only as much as 3 hidden layers, however, deep networks can have as many as 150.

2.2.2 Artificial Neural Network

Artificial Neural Network (ANN) is a software implementation of the neuronal structure of the mammalian brain. The mammalian brain is composed of several million neurons which can be likened to switches whose states can be changed depending on the strength of their electrical or chemical input. The neurons can be likened to a large interconnected complex network, where the output of any of the neurons may be input to thousands of others. Learning occurs by repeatedly activating certain neural connections over others, thereby reinforcing those connections. It makes them more likely to produce a desired output given a certain input (Thomas, 1995). Technically, a fraction of neural network consists of simple processing units – analogous to nodes – and directed weighted connections between them. It consists of a sorted two set inputs (N, V, w). The strength of a connection between two neurons *i* and *j* can be represented as w_{ij}. Neural nets consist of a couple of Neurons identified as N, and V is a set of $(i, j)|i, j \in N$ whose elements are connections between neurons *I* and another *j* represented as w_{ij}. The Function $w : V \rightarrow R$ (R is Real Numbers) defines weight, where w(i, j), is the weight of the connection between neurons *I* and *j*. (Kriesel, 2005).

2.2.2 Propagation Function

For a neuron *j,* several other neurons deliver their computations (transfer their output to j). The propagation function of the neuron j receives the output $o_1,\ldots,o_n$ of other neurons $i_1,\ldots\ldots i_n$ (connected to j), and transforms them in relation to the weights w_{ij} into a network net that can be further processed by the activation function. It can be represented as such:

$$\text{NETJ} = \sum_{i\ \epsilon\ i}^{n} (Oi\,.\,wij) \ldots\ldots\ldots\ldots (1)$$

2.2.3 Bias

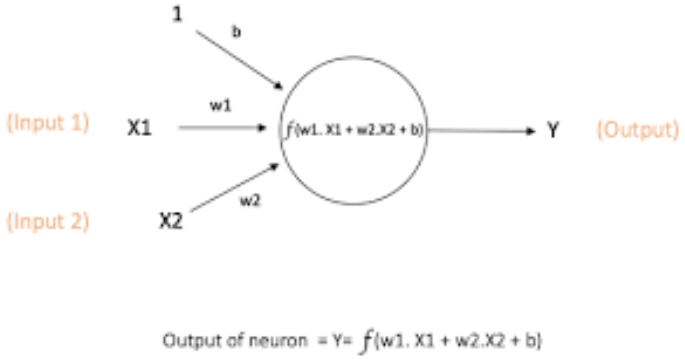


Figure 2.1 Single Layer Perceptron

As shown in Figure 2.1, the circle represents a node and it is the seat of the activation function. It takes as input, the weights and result of previous computations, sums them and inputs them to the activation function. The inclusion of the bias “b” gives the neuron flexibility to adjust its output.

For a linear model as this, results are achieved by mapping input functions to an output (performed in the hidden layers). The vector is given by

$$F(x) = w^T x + b \ldots\ldots\ldots\ldots (2)$$

2.2.4 Activation Function

The activation function indicates the extent of a neuron's activity. Its main purpose is to convert an input signal of a node in a neural network to an output signal.

Essentially, in a neural network, sum of products of inputs and their corresponding weights is taken, bias is added and then subjected to activation function to get the output of that layer which is then fed as input to another neuron. Neurons get activated when the network input exceeds their threshold value.

The threshold value can be explained as the position of the maximum gradient value of the activation function (Kriesel, 2005). Thus, the activation function can be rewritten:

$$A_J(T) = F_{ACT}\left(NET_J(T), A_J(T-1), \Theta_J\right) \quad \ldots\ldots\ldots (3)$$

where net_j is the network input, a_j (t − 1) is the previous activation state, Θ_j is the threshold value and a_j (t) represents the new activation state.

Naturally, neural networks produce linear results from a single node network, and so for a network with several nodes, the need to convert to non-linear in order to further manipulate and understand patterns arises. This births the need for an activation function. The following equation further explains:

$$Y = \alpha(W_1X_1 + W_2X_2 + \ldots\ldots + W_NX_N + B) \quad \ldots\ldots\ldots\ldots (4)$$

where α is the activation function (Agostinelli, Hoffman, Sadowski, & Baldi, 2014).

The need for the activation function is to convert the linear input signals and models into non-linear signals such that higher order polynomials beyond degree one can be investigated. The common activation functions are discussed further.

2.2.4.1 Sigmoid Function

The sigmoid, logistic or squashing function is a non-linear Activation Function used mostly in Feedforward Neural Networks. It can be represented by the equation

$$f(x) = \left(\frac{1}{(1+\text{EXP}-x)} \right) \ldots\ldots\ldots\ldots (5)$$

The sigmoid function, as shown in Figure 2.2, is used for the prediction of probability-based output and has found application in the modelling of logistic regression tasks. It has mostly been used in shallow networks and generally maps functions to between the range of 0 and 1. It however suffers some drawback which include sharp damp gradients during backpropagation, gradient saturation, slow convergence and non-zero centred output, thereby causing the gradient updates to propagate in different directions (Nwankpa, Ijomah, Gachagan, & Marshall, 2018). Consequence of which is that models which utilize the sigmoid function will be slow learners in the experimentation phase and will eventually generate prediction values with low accuracy (Lai, Pan, Liu, & Yan, 2015).

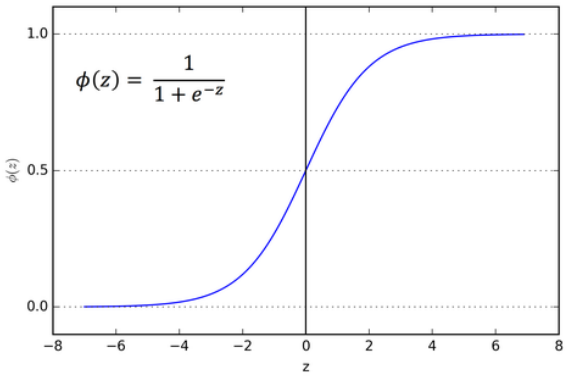


Figure 2.2 Sigmoid Function

2.2.4.2 Hyperbolic Tangent Function (Tanh)

The tanh function is a scaled-up version of the sigmoid function with a bound between the values of -1 and 1. As shown in Figure 2.3, Its output is represented by:

$$f(x) = \left(\frac{\text{EXP}\, x - \text{EXP}\, -x}{\text{EXP}\, x + \text{EXP}\, -x} \right) \ldots\ldots\ldots\ldots (6)$$

The tanh function is preferred to the sigmoid because it gives a better training performance for multilayer networks. It however suffers from the vanishing gradient problem like the sigmoid function(Hu, Lu, & Tan, 2014).

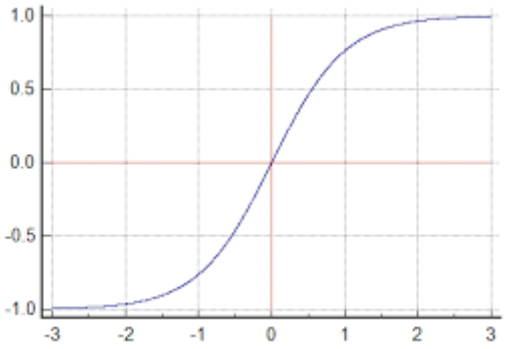


Figure 2.3 Hyperbolic Tangent Function

2.2.4.3 Rectified Linear Unit (ReLU)

ReLU has been a better activation function with a faster learning capability, generalization and performance. It rectifies the values of the inputs less than zero, thereby forcing them to zero and eliminating the vanishing gradient function. It can be represented as:

$$f(x) = \text{MAX}(0, x) = x, if\ x \geq 0 \mid 0, if\ x < 0 \ldots\ldots\ldots\ldots (7)$$

ReLU, shown in Figure 2.4, guarantees faster computation and works by “squishing” values between zero and maximum (Maas, Hannun, & Ng, 2013; Zeiler et al., 2013).

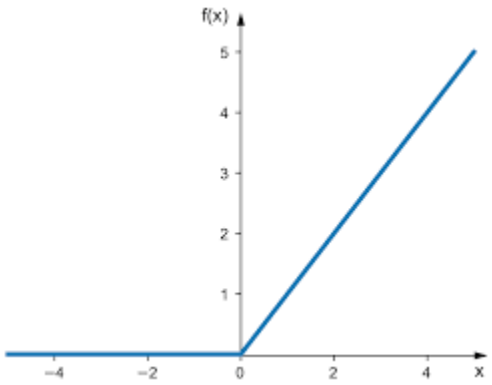


Figure 2.4 Recursive Neural Tensor Networks

The RNTN, a brainchild of Richard Socher was proposed as part of his doctorate thesis. Its purpose was to analyse data via a hierarchal structure. They were initially designed for sentiment analysis with the assumption that the sentiment of a sentence structure depends on the individual component words and their syntactic grouping. The RNTN is composed of two groups: the parent (root) and the child (leaf); with each group being a collection of neurons where the neuron number depends on the complexity of the input data.

The leaves are connected to the root without being connected themselves, forming a binary tree. Essentially, the leaf group receives input and the root group uses a classifier to output a class and a score (Bowman, 2013; Bowman, Potts, & Manning, 2014; Irsoy & Cardie, 2014; Socher, Chen, Manning, Chen, & Ng, 2013).

RNTN framework, shown in Figure 2.5 works by applying the same set of weights recursively over a structured input to produce a structured prediction. They have been successful in learning sequence of patterns and tree structures in natural language processing. They are mostly trained with backpropagation. Below is a representation of a simple RNTN:

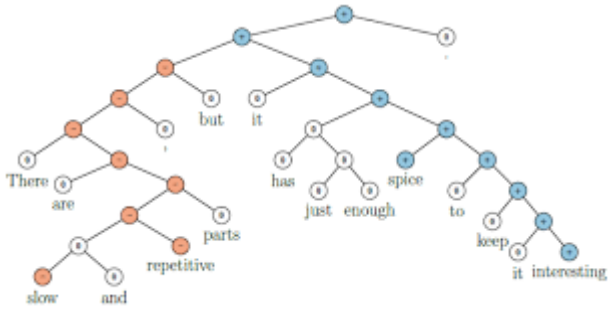


Figure 2.5 A Model of Recursive Neural Network

2.4 Support Vector Machine

Support Vector Machine (SVM) is an algorithm that learns by example by assigning labels to objects based on previous knowledge. It has found application in fraud detection by examining pattern in transactions involving fraudulent and non-fraudulent credit card activities. SVM has also been used in the examination of images. It is a supervised learning model and works by mapping vectors in separate categories such that each is as wide as possible. New vectors are then attempted to be mapped into the same space based on prediction made by the model (Cortes & Vapnik, 1995; Gholami & Fakhari, 2017; Noble, 2006).

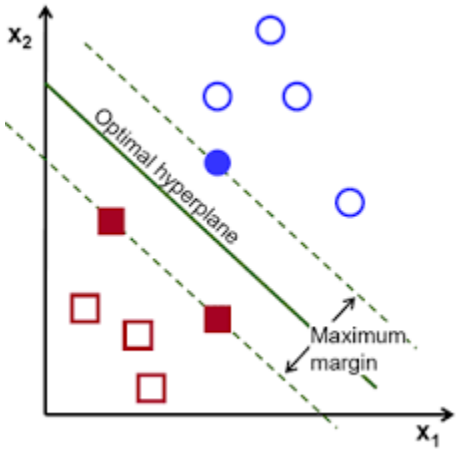


Figure 2.6 Support Vector Machine Model

Essentially, SVM algorithm is the identification and selection of the right hyperplane(s) which maximally segregates classes best (as shown in Figure 2.6). From the learned classification, further test sets can then be classified.

2.5 Existing Work

Several methods have been proposed by researchers in a bid to safeguard internet devices and its users from all forms of attacks, and machine learning has had a long-standing history in that respect. Efforts date back to the 1990s in this regard (Forrest, Hofmeyr, Somayaji, & Longstaff, 2002; Lane & Brodley, 2002; Lee, Stolfo, & Mok, 1999; Sinclair, Pierce, & Matzner, 1999; Warrender, Forrest, & Pearlmutter, 1999). Over the last few decades, these approaches have been improved upon and applied to several other sectors including industrial and national grids (Feng, Li, & Chana, 2017). Ever since its introduction, machine learning and its variants have grown in popularity and application to detection of malicious activities in desktop systems (Huang & Stokes, 2016), mobile devices (Olejnik et al., 2017) and web applications (Canali, Cova, Vigna, & Kruegel, 2011; Kapravelos, Shoshitaishvili, Cova, & others, 2013).

The available malicious activity detection methods can be divided into Adaptive and Constant detection techniques. Adaptive detection techniques always incorporate new traffic data into their learned repository to induce an up-to-date detection model - that way, they are abreast of new and emerging methods employed by attackers, while constant detection technique use a collected data to build a model once and make further predictions based on that - it is analogous to what is achieved in the firewall and antivirus software.

Intrusion detection models can be further classified as either Signature based or Anomaly based. Signature based schemes seek defined patterns in the body of a request in order to raise an alarm, while anomaly-based detectors define a normal behavior for a system to be protected. A deviation from the standard beyond a threshold is termed malicious and thus results in a flag.

(Swarnkar & Hubballi, 2016) proposed a content anomaly detection model to identify network packets with suspicious payload. OCPAD (One Class Naïve Bayes classifier for payload-based anomaly detection) matches the occurrence of each sequence in a payload against a known non-malicious packet as a measure to derive the degree of maliciousness in such packet. In the training phase, the classifier is made to generate the likelihood range of such sequence's occurrence from read packets. The model treats a sequence as anomalous if its signature is not found in the database or if its likelihood of occurrence in a packet is not found in the training set range.

In the work done by (Robertson, Vigna, Kruegel, & Kemmerer, 2005), they proposed a model where unknown attacks were scrutinized by some custom built software, characterized and grouped in order to forestall further attacks. Their model comprises of

an event collector, an anomaly detection component, anomaly aggregation component, anomaly signature generation component and an attack class inference component. The event collector, having retrieved the request logs from the repository, normalizes them. The normalized events are then passed to the anomaly detector which decides whether they are anomalous or not. If an event is normal, no flags are raised, however, if one is suspected to contain payloads, it is passed across to the anomaly aggregation component which tries to compare with a set of signatures to determine whether such event is similar to a previous one. If an event can be matched with a signature, a flag is generated and it is grouped with instances of similar flags. Otherwise, the event is passed to the anomaly signature generation process. Their work relates to that done by (Kruegel & Vigna, 2003) who used web server logs to detect attacks aimed at servers and its applications. Client queries were investigated for potential attacks using six methods (length and structure of parameter). Their model assigned a probability value to either a query or one of its attributes. The value is matched against a standard, with the assumption that abnormal values indicate a potential attack.

While constant detection models tried to manage the http request intrusion situation, they did not seem to address cases of dynamic changes in attack modes. Adaptive intrusion detection models however go a step further by constantly updating their learned repository so new attack modes can be captured. Such is the case with (Meng & Kwok, 2013) who deployed a blacklist filter. The system consists of a blacklist packet filter and a custom monitoring engine. Their model was designed to consolidate on the activities of a regular intrusion detection system. The blacklist packet filter is responsible for filtering out packets according to a blacklist template. On the other hand, a monitor

engine is responsible for monitoring the System (NIDS) and collecting statistical data regarding the network traffic. Furthermore, it periodically updates the blacklist packet filter thereby enhancing the packet filtration process. Guyet and Quiniou (W. Wang et al., 2014) in their work proposed an adaptive framework of autonomic intrusion detection over unlabeled HTTP traffic streams. They employed Affinity Propagation (AP) algorithm to learn a subject's behaviors through dynamic clustering of the streaming data. An initial template is first built as clusters, thereafter, subsequent payload is compared against such clusters. If payload data strays from a cluster, then it is flagged as anomalous. If, however, it is not largely strayed, then it is considered suspicious.

(Pinzón, De Paz, Bajo, Herrero, & Corchado, 2010) introduced an Adaptive Intelligent Intrusion Detector Agent for SQL injection attacks. The AIIDA-SQL engine incorporated a Case-Based Reasoning (CBR) engine which had been equipped with reasoning and learning capabilities for classifying queries and detecting malicious requests. A classification model based on the mixture of an Artificial Neural Network and Support Vector Machine is then applied in the subsequent usage of CBR. (Dong et al., 2018) proposed an Adaptive System for Detecting Malicious queries in web attacks (AMOD). They also presented an enhanced Support Vector Machine called SVM-Hybrid, designed to leverage on their model. AMOD consists of an ensemble classifier with 3 base classifiers and SVM as its meta-classifier. The SVM-Hybrid is a custom kernel of SVM and relies on the position of queries in the classifier to decide labeling position. SVM-Hybrid classifies queries into either Suspicious Selection (queries which lie in a confusion region where the classifier cannot readily aggregate) and Exemplar Selection (representation of queries to be aggregated).

CHAPTER THREE

METHODOLOGY

In this chapter, we shall discuss the framework, algorithm used and the explanation of the various stages in the framework.

3.1 Concept of Classification Technique

Classification is one of the ways by which machines learn. The concept of machine learning focuses on the development of automated systems that are able to process large amount of data in order to extract meaningful and potentially useful information.

Classification comprises information extraction and their eventual utilization. Such activity is aimed at developing tools that having studied a large pool of datasets, are able to automatically label never seen data.

Several work have been conducted on classification; from neural nets, deep learning and vector machines. In this work, we try to establish the effectiveness of Ensemble Learning in comparison to other learning algorithm.

3.2 Software Design Phase

The proposed model was implemented using Python Programming language 3 with relevant dependencies and libraries designed for machine learning. Python has the advantage of flexibility and a large community of support for machine learning and deep learning over regular programming languages.

3.3 Proposed Framework

The proposed machine learning framework is as shown in Figure 3.1.

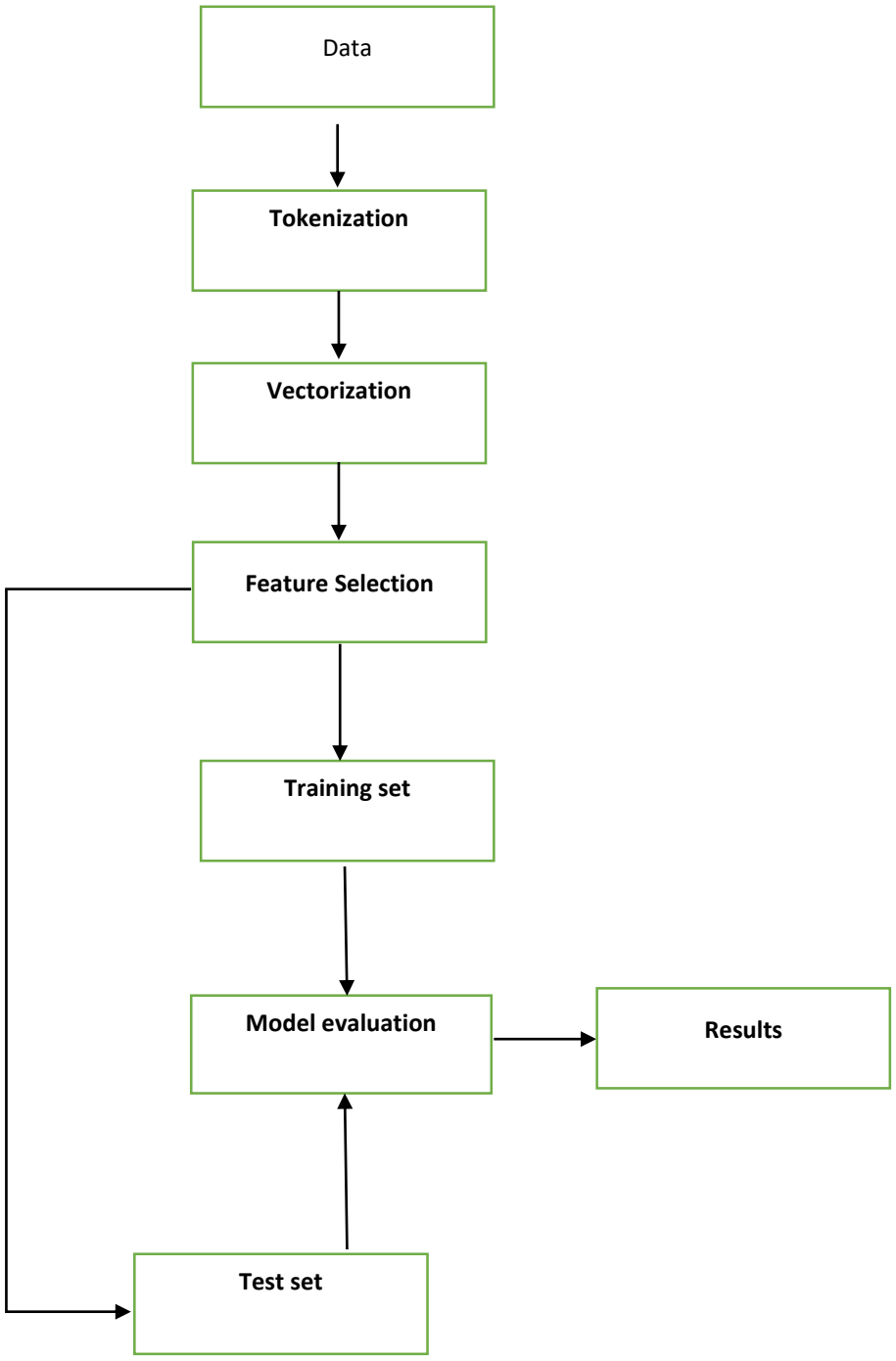


Figure 3.1 Machine Learning Paradigm

3.4 Intrusion Detection Framework

The proposed framework is expected to act as an interface that verifies the authenticity of the HTTP requests emanating from the user. It is expected that requests that are perceived to be malicious are dropped, with their signature used to retrain the machine learning model. As shown in Figure 3.2, non-malicious requests are however allowed passage to be processed by the application server.

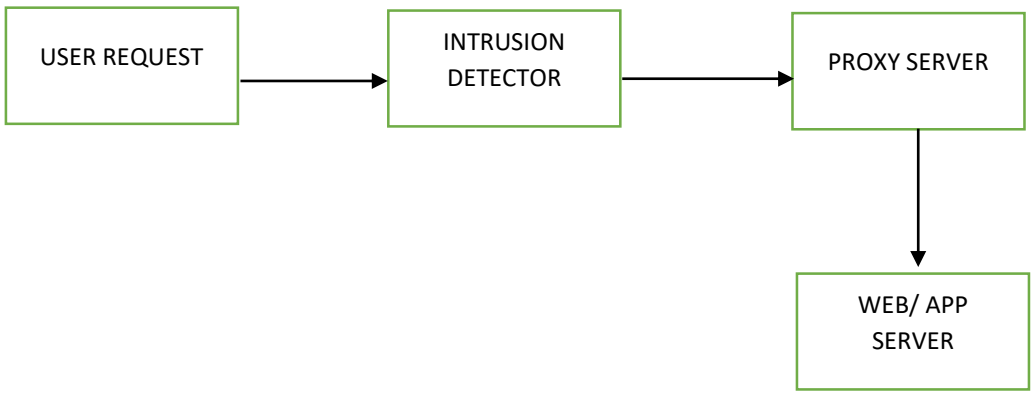


Figure 3.2 Intrusion Detection Framework

Machine learning model in the above Figure is expected to be a request sanitizer, safely distinguishing requests that could potentially result in the compromise of the backend servers from harmless requests. The model could be implemented in a dedicated hardware (comparable to what's obtainable with Sophos or Cisco ASA threat protection) or made to run on a computer much like a regular endpoint protection program.

3.5 Dataset Description

The dataset was sourced online and are fetched from log reports of backend servers. It is available at https://github.com/cozpii/Malicious-URL-detection

As shown in Figure 3.3, the dataset is pre-labeled and alphabetic in nature, consisting of two columns – URL on one side and malignant status on the other. An entry labelled “bad” signifies a sample URL that is malicious and “good” signifies a non-malicious entry. There were 420,464 entries in total with 344,821 being non-malicious and 75,643 malicious elements.

	A	B
1	url	label
2	diaryofagameaddict.com	bad
3	espdesign.com.au	bad
4	iamagameaddict.com	bad
5	kalantzis.net	bad
6	slightlyoffcenter.net	bad
7	toddscarwash.com	bad
8	tubemoviez.com	bad
9	ipl.hk	bad
10	crackspider.us/toolbar/install.php?pack=exe	bad
11	pos-kupang.com/	bad
12	rupor.info	bad
13	svision-online.de/mgfi/administrator/components/com_babackup/classes/fx29id1.txt	bad
14	officeon.ch.ma/office.js?google_ad_format=728x90_as	bad
15	sn-gzzx.com	bad
16	sunlux.net/company/about.html	bad
17	outporn.com	bad
18	timothycopus.aimoo.com	bad
19	xindalawyer.com	bad
20	freeserials.spb.ru/key/68703.htm	bad
21	deletespyware-adware.com	bad
22	orbowlada.strefa.pl/text396.htm	bad
23	ruiyangcn.com	bad
24	zkic.com	bad
25	adserving.favorit-network.com/eas?camp=19320;cre=mu&grpid=1738&tag_id=618&nums=FGApbjFAAA	bad
26	cracks.vg/d1.php	bad
27	juicypussyclips.com	bad
28	nuptialimages.com	bad
29	andysgame.com	bad
30	bezproudoff.cz	bad
31	ceskarepublika.net	bad
32	hotspot.cz	bad
33	gmcjjh.org/DHL	bad
34	nerez-schodiste-zabradli.com	bad
35	nordiccountry.cz	bad

Figure 3.3 Sample of Dataset

3.6 Data Transformation

3.6.1 Tokenizaton

Tokenization in Python is done by the NLTK library's word_tokenize() function. It helps to convert group of sentences into token by splitting data into chunk of words thereby eliminating punctuations, stop words, uppercase and lower-case characters.

3.6.2 Vectorization

Term Frequency – Inverse Document Frequency (TF-IDF) is a method of evaluating the importance of a word in a document. Term Frequency indicates the number of times such word appears in such document. Inverse Document Frequency however, measures the weight of the word in the document. It follows the assumption that terms that appear frequently in a document are less important than terms that are rare (Drucker, Member, Wu, Member, & Vapnik, 1999; Zhang, Yoshida, & Tang, 2011).

$$Wi,j = \text{TFI,J X LOG}\left(\frac{N}{dfi}\right) \ldots\ldots\ldots\ldots (8)$$

3.6.3 Feature Extraction

For this work, the SelectKBest function was employed. It works by scoring the features using a suitable function (chi2) and then removes all but the k highest scoring features.

3.7 Ensemble Learning

Ensemble learning, shown in Figure 3.4, is a concept that combines multiple learning algorithms. Each algorithm solves the same initial task independently in order to obtain a better combined model with accurate and reliable decisions than could have been obtained from using a single model.

They contain learners referred to as base learners. It has the ability to generalize; thereby making weak learners become strong learners capable of making accurate predictions (Li, Gao, Li, & Fan, 2014; Oza, 2000). It works by constructing a model through which it classifies datasets by taking a vote among the predictions (Yoneda-Kato et al., 1996).

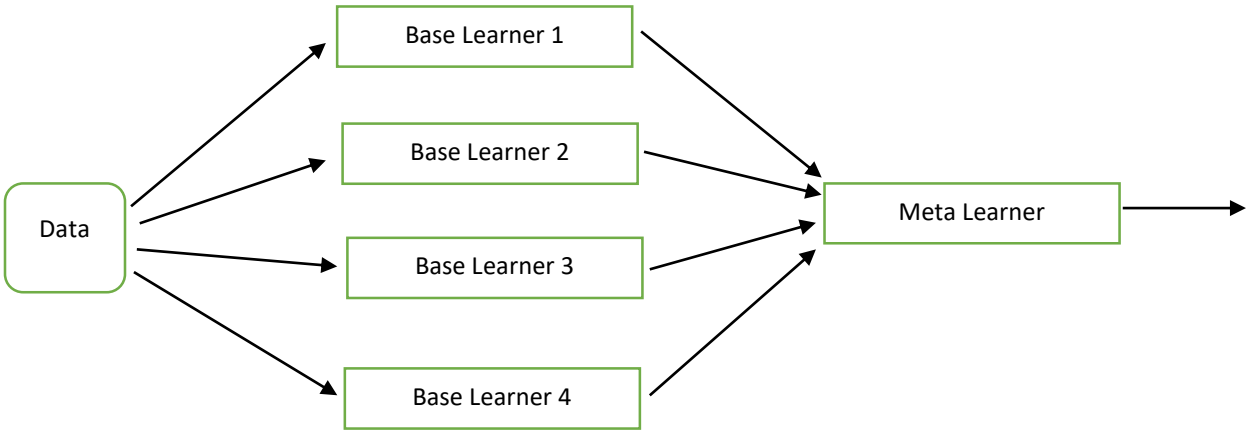


Figure 3.4 Ensemble Learning Model

In machine learning, training set usually has more of the instances than test set in the ratio 60:40. In this case, that is 336,371 train sets against 84,093 test set.

3.7.1 The ensemble Framework

An ensemble method algorithm contains the following building blocks:

3.7.1.1 Training set

This is mostly a labelled dataset used for training an ensemble algorithm. Dataset usually comes as a set of n attributes.

3.7.1.2 Base Inducers

The inducer forms a classifier which is a representation of the relationship between the input attributes and the target attributes

3.7.1.3 Diversity Generator

The diversity generator produces the array of classifiers.

3.7.1.4 Combiner

This combines the classification of the various classifiers and applies the various voting techniques.

3.7.2 Training Algorithm for Ensemble Learning

There are several algorithms for training data in the Ensemble model. As earlier stated, Ensembles use several learners to complement the weaknesses of an individual learning algorithm. Most ensemble methods use a single base learning algorithm to produce homogeneous learner. Others, however, combine multiple learning algorithms to produce heterogeneous learners. In building ensembles, there are dependent frameworks as well as independent frameworks. In a dependent framework, shown in Figure 3.5, the result of a classifier is used in the construction of another. Such models take advantage of knowledge generated from previous classifications to inform next aggregation.

3.7.2.1 Dependent method

Also known as sequential ensemble method functions by generating base learners in a sequential manner. Essentially, sequential method attempts to exploit dependence between base learners, thereby providing better performance by weighing in on previously mislabeled output. Further methodologies are underlisted

- Incremental Batch Learning: Result of aggregation produced in one iteration is given as prior knowledge in order to influence the learning of the subsequent iteration.

- Model-guided instance selection: Previous classification is also used in the manipulation of subsequent training set; however, this method ignores instances in which classifier result is correct and only learns from misclassified instances.

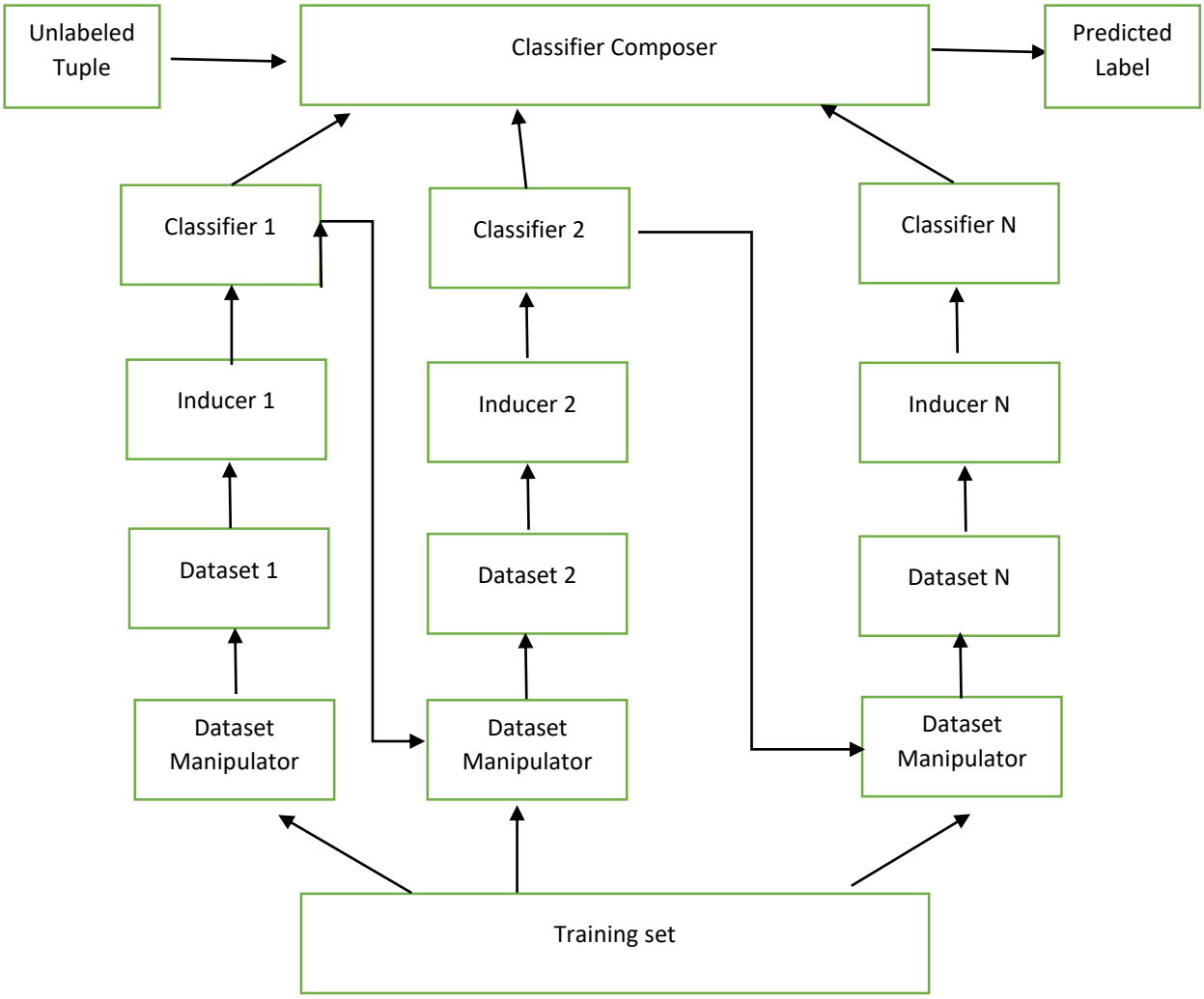


Figure 3.5 Model for Dependent Ensemble Method

Popular dependent ensemble algorithms are:

- **Boosting**

Also known as Adaptive resampling and combining. Additional models are added to the overall sample in a way that allows creation of a new model and the base learner is trained from the errors of the previous learners (Quinlan, 2006). It essentially works by repeatedly running a weak learner over various distributed training data. The resulting classifier is thereafter combined to achieve a better performance.

- **AdaBoost**

AdaBoost, also referred to as Adaptive Boosting is a model that improves the boosting process. The base learner takes all the distributions and assign equal weight to every pattern in the training set. Over every iteration, weight of all misclassified instance is increased while weight of every correctly classified instance is decreased. As a result, the model is forced to focus on difficult dataset instances (Freund & Schapire, 1996).

3.7.2.2 Independent method

Independent framework does not necessarily generalize based on result of past classification. The dataset is first transformed into various data units from which several classifiers can be trained. As shown in Figure 3.6, a combination or voting method is thereafter applied in order to conclude the final aggregation.

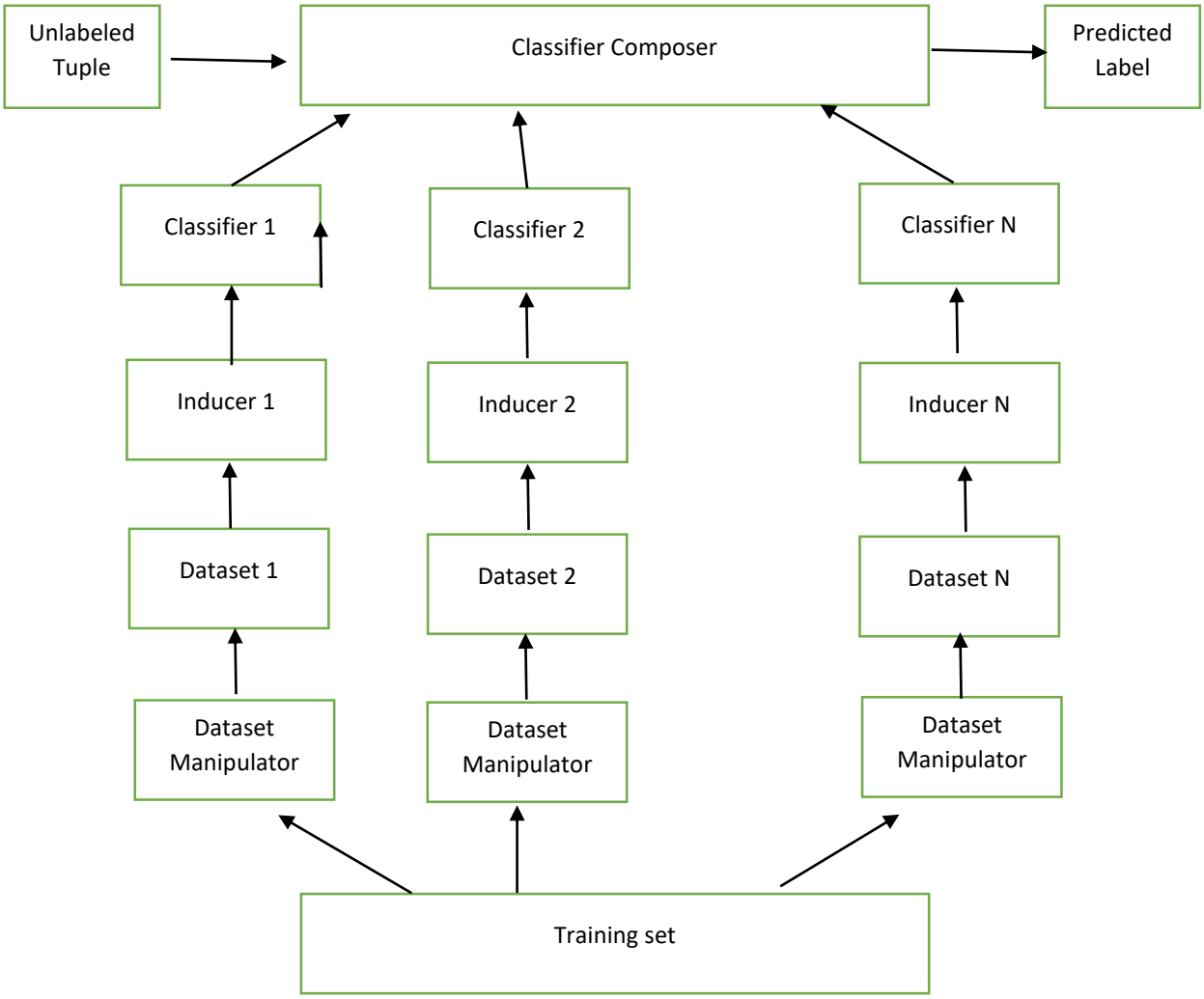


Figure 3.6 Model for Independent Ensemble Method

Common independent methods are:

- **Bagging**

It is a combination of Bootstrapping and Aggregation and is also known as Bootstrap Aggregation. The method achieves accurate classification by combining output of various classifications into a single prediction.

Bootstrapping helps decrease variance of a classifier and thus reduce overfitting by resampling data from training set provided (Leo, 1996; Tan & Gilbert, 2003).

To classify a new instance of a dataset, each of the classifier provides a classification result for the new instance. The bagging algorithm then uses a voting method to return a result that has been predicted most often. Essentially, Bagging excels with unstable and limited data by aggregating scores over many samples.

- **Stacking**

A new model is formed from the combined predictions of two or more other models. It involves different models built by different inducers with a meta-classifier being made to do a final classification. As seen in Figure 3.7, dataset instance are first classified by each of the base classifiers, after which the results are fed into a meta-level training set from which a meta-classifier is generated. The meta-classifier is tasked with the responsibility of combining the different predictions into a final one (Džeroski & Ženko, 2004).

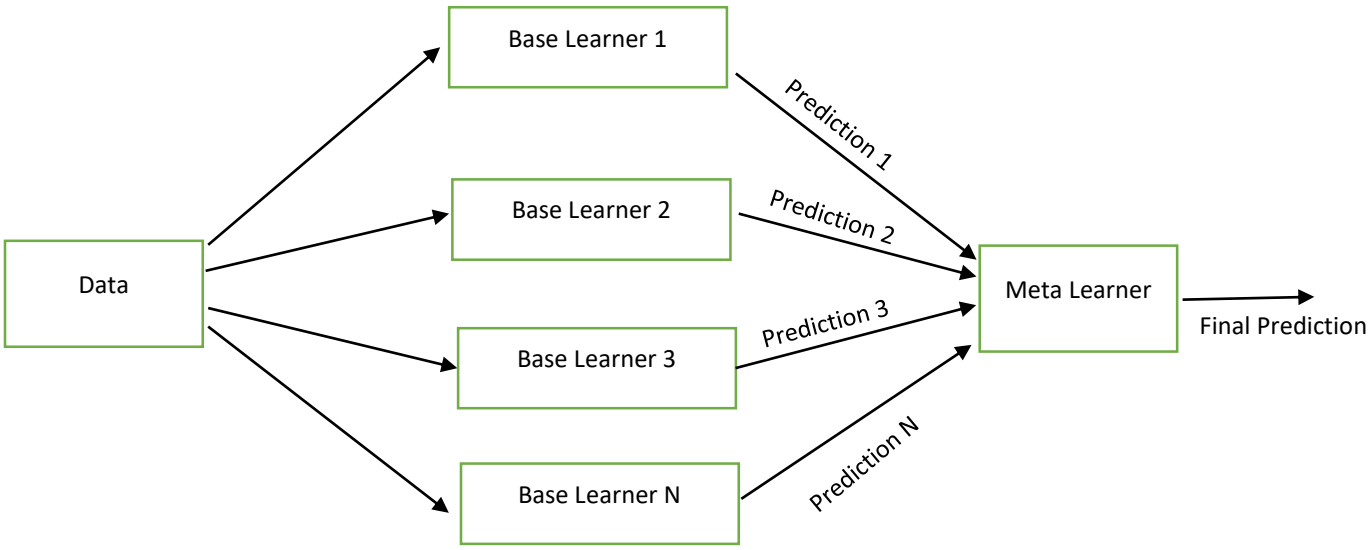


Figure 3.7 Stacking Ensemble Model

3.8 Performance Evaluation Metrics

3.8.1 Confusion Matrix

In order to check the accuracy of our results, we employ the confusion matrix to evaluate performance of the models. Confusion matrix is a table that is used to describe the performance of a model. It compares relationship between the predicted and actual classifications in a graphical format as shown in table 3.1. Data is thereafter presented in a N x M (N number of rows against M number of columns) table format.

i. True-Positive indicative of correctly classified patterns. A Predicted ‘YES’ is actually ‘YES’.

ii. False-Positive indicative of incorrectly classified patterns. A Predicted ‘YES’ is actually ‘NO’.

iii. False-Negative indicative of incorrectly classified patterns. A Predicted ‘NO is actually ‘YES’.

iv. True-Negative indicative of correctly classified patterns. A Predicted ‘NO is actually ‘NO’.

Table 3.1 Confusion Matrix

		Actual Class	
		Positive	**Negative**
Predicted Class	**Positive**	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

3.8.2 Measures

i. **Accuracy** implies the number of correct predictions made by the model over all possible predictions made.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

ii. **Recall/ Sensitivity** corresponds to the number of positive predictions that are correctly predicted as positive.

$$Sensitivity = \frac{TP}{TP + FN}$$

iii. **Specificity** corresponds to the number of negative predictions which are wrongly considered as positive.

$$Specificity = \frac{TN}{TN + FP}$$

iv. **Precision** determines how many of the selected items were actually correct.

$$Precision = \frac{TP}{TP + FP}$$

CHAPTER FOUR

RESULTS AND DISCUSSIONS

The implemented work discussed in chapter 3 was initiated using Python programming environment, version 3.6.4. Screenshots of results along with code snippet are presented. This chapter captures preprocessing from tokenization, vectorization down to classification with Ensemble learning.

4.1 Presentation of Results

This work presents a model for malicious URL detection using machine learning tools and already labelled dataset. The various stages of preprocessing, vectorization, feature selection, data split into training/ testing sets, and measures of accuracy are implemented using Python programming language

4.1.1 Reading Text File

The data was downloaded from github repository and through the directory https://github.com/cozpii/Malicious-URL-detection with the filename data.csv. The data is uploaded and read into the Python notebook (as seen in Figure 4.1), after which it is further manipulated. The dataset can be further viewed in pandas library format as shown in Figure 4.2.

```
1 #read from a file
2 data = pd.read_csv("../data/data.csv",',',error_bad_lines=False)    #reading file
3 data['url'].values
```

Figure 4.1 Code to read Data into Model

	url	label
0	diaryofagameaddict.com	bad
1	espdesign.com.au	bad
2	iamagameaddict.com	bad
3	kalantzis.net	bad
4	slightlyoffcenter.net	bad
5	toddscarwash.com	bad
6	tubemoviez.com	bad
7	ipl.hk	bad
8	crackspider.us/toolbar/install.php?pack=exe	bad
9	pos-kupang.com/	bad
10	rupor.info	bad
11	svision-online.de/mgfi/administrator/component...	bad
12	officeon.ch.ma/office.js?google_ad_format=728x...	bad
13	sn-gzzx.com	bad
14	sunlux.net/company/about.html	bad

Figure 4.2 Pandas Formatted Data Sample

The data label is further transformed into binary, from the initial good – bad labels. Code snippet shown in Figure 4.3 converts ‘bad’ to 0 and ‘good’ to 1, where its result can be seen in Figure 4.4.

```
data.replace('bad',0, inplace=True)
data.replace('good',1, inplace=True)
```

Figure 4.3 Code to Transform Label into Binary form

	url	label
0	diaryofagameaddict.com	0
1	espdesign.com.au	0
2	iamagameaddict.com	0
3	kalantzis.net	0
4	slightlyoffcenter.net	0
5	toddscarwash.com	0
6	tubemoviez.com	0
7	ipl.hk	0
8	crackspider.us/toolbar/install.php?pack=exe	0
9	pos-kupang.com/	0
10	rupor.info	0
11	svision-online.de/mgfi/administrator/component...	0
12	officeon.ch.ma/office.js?google_ad_format=728x...	0
13	sn-gzzx.com	0
14	sunlux.net/company/about.html	0
15	outporn.com	0
16	timothycopus.aimoo.com	0
17	xindalawyer.com	0
18	freeserials.spb.ru/key/68703.htm	0

Figure 4.4 Transformed Label

4.1.2 Data Preprocessing

The dataset is first subjected to Tokenization in order to split sentences or string of words into chunks. The activity strips such sentences of punctuations, stop words, uppercase and lowercase characters. Common words are equally eliminated while regular words are replaced with their root words. The process is achieved with a Token function and the knowledge of regular expressions as shown in Figure 4.5.

```python
#this function is taken from https://github.com/faizann24/Using-machine-learning-to-detect-malicious-URLs
def getTokens(input):
    tokensBySlash = str(input.encode('utf-8')).split('/')
    allTokens = []
    for i in tokensBySlash:
        tokens = str(i).split('-')
        tokensByDot = []
        for j in range(0,len(tokens)):
            tempTokens = str(tokens[j]).split('.')
            tokensByDot = tokensByDot + tempTokens
        allTokens = allTokens + tokens + tokensByDot
    allTokens = list(set(allTokens))
    if 'com' in allTokens:
        allTokens.remove('com')
    return allTokens

#function to remove "http://" from URL
def trim(url):
    return re.match(r'(?:\w*://)?(?:.*\.)?([a-zA-Z-1-9]*\.[a-zA-Z]{1,}).*', url).groups()[0]
```

Figure 4.5 Code for Tokenization

Subsequently, Term Frequency - Inverse Document Frequency (TF-IDF) feature extraction is applied to extract important features in a matrix numerical format from the word corpus.

TF-IDF, shown in Figure 4.6, uses vector space modelling technique for text document representation with the assumption that terms that appear frequently in a document are less important than terms that are rare.

```python
#convert text data into numerical data for machine learning models
y = [d[1] for d in data]
corpus = [d[0] for d in data]
vectorizer = TfidfVectorizer(tokenizer=getTokens)
X = vectorizer.fit_transform(corpus)
```

Figure 4.6 Code for Vectorization

SelectKBest function, shown in Figure 4.7, is further applied to extract important values from the generated numerical matrix with the result shown Figure 4.8.

```
# Import SelectKBest, chi2(score function for classification), f_regression (score function for regression)
from sklearn.feature_selection import SelectKBest, chi2, f_regression
```

```
X_clf_new=SelectKBest(score_func=chi2,k=7).fit_transform(X,y)
```

Figure 4.7 Code for Feature Selection

```
(25, 7)       0.15052719410410223
(77, 7)       0.15052719410410223
(120, 7)      0.14546399736478194
(176, 2)      0.18072997057754225
(183, 7)      0.14546399736478194
(185, 2)      0.18072997057754225
(210, 7)      0.14546399736478194
(248, 7)      0.15169181369439338
(275, 7)      0.15169181369439338
(282, 5)      0.2316921807177718
(282, 4)      0.17806069700671998
(282, 7)      0.1780340398079766
(285, 7)      0.14546399736478194
(286, 7)      0.17541191668621592
(290, 7)      0.17541191668621592
(306, 2)      0.1628411036713788
(343, 7)      0.15551210044073774
(358, 4)      0.14430323665945494
(365, 5)      0.23508316784342517
(365, 4)      0.18066674754007897
(365, 7)      0.18063970019343562
(368, 7)      0.1759906598612285
(379, 5)      0.2551332220451028
(379, 4)      0.1960756690458162
(379, 7)      0.19604631485274407
:     :
(420332, 9)   0.1036785014240712
(420359, 4)   0.17626959264392206
(420362, 5)   0.2294719825045806
(420362, 4)   0.17635442431288462
(420362, 7)   0.17632802255765614
(420363, 8)   0.24726501884384866
(420363, 1)   0.25189349442184267
(420364, 8)   0.2609990981058653
(420364, 1)   0.26588465756393137
(420365, 7)   0.14381579336778322
```

Figure 4.8 Numerical Equivalent of Dataset

4.2 Classification

The pre-processed data is fed into the desired machine learning algorithm. For this work, we compared the performance of Ensemble learning with DecisionTree as its base classifier against the performance of Ensemble learning with Support Vector Machine as its base classifier and yet Support Vector Machine with Linear kernel. The training data is about 60 percent of the entire dataset while the test set is of the remaining 40 percent.

4.2.1 Support Vector Machine

The data set was subjected to support vector machine with a linear kernel and penalty of 3 as shown in Figure 4.9.

```
1  svcModel = LinearSVC(C=3)
```

```
1  svcModel.fit(X_train, y_train)
```

```
LinearSVC(C=3, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0)
```

```
1  svcModel.score(X_test, y_test)
```

```
0.7872355606293032
```

Figure 4.9 Code for Linear SVC Classifier

It can be observed that Support Vector Machine has a performance score of 78 percent with the stated parameters.

4.2.2 Ensemble using Support Vector Machine as Base Classifier

The dataset was also subjected to Ensemble Bagging classifier with SVM. The active code component and performance are shown in Figure 4.10 and Figure 4.11.

By applying Bagging Ensemble method, it is expected that a better performance is achieved as errors in base classifier are gradually cancelled out.

```
[ ] BaggingClassifier(base_estimator=LinearSVC(C=3, class_weight=None, dual=True,
                                               fit_intercept=True,
                                               intercept_scaling=1,
                                               loss='squared_hinge', max_iter=1000,
                                               multi_class='ovr', penalty='l2',
                                               random_state=None, tol=0.0001,
                                               verbose=0),
                      bootstrap=True, bootstrap_features=False, max_features=1.0,
                      max_samples=1.0, n_estimators=100, n_jobs=None,
                      oob_score=False, random_state=7, verbose=0, warm_start=False)

[ ] model2predict = model.predict(X_test)
```

Figure 4.10 Code for Ensemble with SVC Base

```
print(accuracy_score(y_test, modelpredict))

0.8017670911966513
```

Figure 4.11 Accuracy Report for Ensemble with SVC

Result of the Ensemble classifier indicates a better performance in relation to the Support Vector Machine with a score of 80.17 percent.

4.2.3 DecisionTree Classifier

Dataset was equally subjected to Decision tree classifier in order to achieve a second opinion. Decision tree uses a tree system to build a model from which further data can be classified. It naturally performs well and so produced a great result with the dataset.

The following Figure 4.12 and Figure 4.13 show the active code snippet and performance result respectively.

```
[ ]  # Create Decision Tree classifer object
     clf = DecisionTreeClassifier()

     # Train Decision Tree Classifer
     clf = clf.fit(X_train,y_train)

     #Predict the response for test dataset
     y_pred = clf.predict(X_test)
```

Figure 4.12 Function Call for Decision Tree Classifier

```
[ ]  # Model Accuracy, how often is the classifier correct?
     print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

     Accuracy: 0.8001379425160239
```

Figure 4.13 Accuracy Report for Decision Tree

4.2.4 Ensemble using DecisionTree as Base Classifier

The dataset was finally applied to Ensemble Bagging classifier with DecisionTree. Decision tree applies a model with a tree like graph, and thus, generalizes reliably. The active code and performance results are shown in Figures 4.14 and 4.15. By applying Ensemble alongside DecisionTree, it is expected that a much better performance is achieved.

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None,
                                                        criterion='gini',
                                                        max_depth=None,
                                                        max_features=None,
                                                        max_leaf_nodes=None,
                                                        min_impurity_decrease=0.0,
                                                        min_impurity_split=None,
                                                        min_samples_leaf=1,
                                                        min_samples_split=2,
                                                        min_weight_fraction_leaf=0.0,
                                                        presort=False,
                                                        random_state=None,
                                                        splitter='best'),
                  bootstrap=True, bootstrap_features=False, max_features=1.0,
                  max_samples=1.0, n_estimators=100, n_jobs=None,
                  oob_score=False, random_state=7, verbose=0, warm_start=False)
```

```
modelpredict = model.predict(X_test)
```

Figure 4.14 Code for Ensemble with Decision Tree classifier

```
[ ]  print(accuracy_score(y_test, model2predict))

     0.8019514109378902
```

Figure 4.15 Accuracy Report for Ensemble with DecisionTree

Result of the accuracy scores indicate that Ensemble classifiers with Decision Tree gave a better performance by a fraction over Ensemble classifiers with SVM. Although, the aforementioned only slightly edges classification produced by Decision Tree.

4.3 Result of Performance

In order to demonstrate accuracy of the models having tested with 40 percent of the overall data, we generate a confusion matrix to understand the results.

4.3.1 Support Vector Machine

Table 4.1 Confusion Matrix for Support Vector Machine

ACTUAL			
PREDICTED		Negative	Positive
	Negative	24281	32458
	Positive	3539	107908

Support Vector Machine provides True Positive and True Negative values of 107908 and 24281 consecutively as shown in table 4.1; which are approximately 65 and 14 percent of the data.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Accuracy = (107908 + 24281) / (168186)

Accuracy = 78.59

$$Sensitivity = \frac{TP}{TP + FN}$$

Sensitivity = 107908 / (107908 + 32458)

Sensitivity = 76.87

$$Precision = \frac{TP}{TP + FP}$$

Precision = 107908 / (107908 + 3539)

Precision = 96.82

4.3.2 Decision Tree

Table 4.2 Confusion Matrix for Decision Tree

ACTUAL			
PREDICTED		Negative	Positive
	Negative	23939	32800
	Positive	814	110633

Decision Tree provides True Positive and True Negative values of 110633 and 23939 consecutively as seen in table 4.2.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Accuracy = (23939 + 110633) / (168186)

Accuracy = 80.01

$$Sensitivity = \frac{TP}{TP + FN}$$

Sensitivity = 110633/ (110633+ 32800)

Sensitivity = 77.13

$$Precision = \frac{TP}{TP + FP}$$

Precision = 110633/ (110633+ 814)

Precision = 99.26

4.3.3 Ensemble with Decision Tree base classifier

Table 4.3 Confusion Matrix for Ensemble with DecisionTree

ACTUAL			
PREDICTED		Negative	Positive
	Negative	23877	32505
	Positive	804	111000

Ensemble classifier proves to aggregate better with decision tree as its base classifier in comparison to Support Vector Machine as seen from table 4.3. Confusion matrix shows that it predicts better into the true positive and true negative range.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Accuracy = (111000+ 23877) / (168186)

Accuracy = 80.19

$$Sensitivity = \frac{TP}{TP + FN}$$

Sensitivity = 111000/ (111000+ 32505)

Sensitivity = 77.34

$$Precision = \frac{TP}{TP + FP}$$

Precision = 111000/ (111000+ 804)

Precision = 99.28

4.3.4 Ensemble with SVM base classifier

Table 4.4 Confusion Matrix for Ensemble with Support Vector Machine

ACTUAL			
PREDICTED		Negative	Positive
	Negative	23877	32505
	Positive	804	111000

As computed in table 4.4, Ensemble classifier with SVM shares similar confusion matrix result as Ensemble classifier with Decision Tree. This can be attributed to the negligible difference in performance between individual base learners.

Conclusively, a model built on Ensemble method with reliable base classifiers such as as DecisionTree and SVM is likely to produce a strong performance.

CHAPTER FIVE

SUMMARY, CONCLUSION AND FUTURE WORK

5.1 Summary

Automatic intrusion detection of malicious URL has become an important research area considering the spate of hack, cyber-theft, espionage and blackmail. With the proliferation of cheap and efficient devices, it is expected that a lot more people will be connected in the years to come. With the huge number of probable victims comes inspiration for potential hackers to device novel means of attacking unsuspecting users through their web servers.

Web servers are at perpetual risk of being compromised, which may have far reaching consequences on its users. It is therefore important to have a query device as a sanitizer between web servers/ proxy servers and the user requests. Machine learning tools have been applied to these problems due to their potential to classify objects correctly. However, the problems of selecting the best machine learning tool amongst the available plethora becomes the challenge as even a wrongly classified request could result in problems for all connected devices.

Ensemble is a machine learning technique that combines several base models in order to produce a model with optimal classification capability. DecisionTree on the other hand repetitively divides the data into a tree representation after which decisions are made. Ensemble learning with DecisionTree base performs perfectly and is most suitable for a device to distinguish malicious from non-malicious URL.

5.2 Conclusion

According to literature, attempts have been made by different authors to safely classify URLs. Some have resulted in the use of custom Support Vector Machine kernels. Other models, however, have been reported to suffer major setbacks from time inefficiency, inability to observe abnormalities or misinterpretation of abnormalities respectively. In this work, the motive is to discover the classifier that stands to produce the best generalization of malicious from non-malicious URLs. Our implementation suggests an appreciable performance improvement in comparison to other selected classifiers.

The model accepts URL which have been pre-labelled as either malicious or non-malicious. Data processing is thereafter executed on the data in order to make it accessible to the machine learning algorithm. Tokenization, Vectorization and feature selection are carried out sequentially in the preprocessing phase. The data, which now has been transformed into a matrix form is then partition into training and testing sets to the ratio 60:40. The model is trained with the training set and performance tested with the test set. Performance accuracy of the technique was thereafter validated using the confusion matrix. The experiment produced a better and accurate result in comparison to SVM which has been tipped by other authors.

5.3 Future Work

In this work, we emphasized the generalization capability of the Ensemble classifier with Decision Tree as its base classifier in the safe classification of malicious URLs from non-malicious ones. A device built on this model is expected to perform maximally.

For the future work, we propose the use of deep learning tools to not only classify the URLs but to also analyze the entire body of request. Deep learning tools like

Convoluted Neural Networks try to imitate the intelligence in human and thus, suitable for cases of intrusion where attack vectors are constantly changed in order to evade detection.

REFERENCE

Agostinelli, F., Hoffman, M., Sadowski, P., & Baldi, P. (2014). *Learning Activation Functions to Improve Deep Neural Networks*. (2013), 1–9. Retrieved from http://arxiv.org/abs/1412.6830

Arp, D., Spreitzenbarth, M., Malte, H., Gascon, H., & Rieck, K. (2013). Forest of pressure: Ogawa Shinsuke and postwar Japanese documentary. *Choice Reviews Online*, *45*(02), 45-0765-45–0765. https://doi.org/10.5860/choice.45-0765

Bowman, S. R. (2013). *Can recursive neural tensor networks learn logical reasoning?* 1–10. Retrieved from http://arxiv.org/abs/1312.6192

Bowman, S. R., Potts, C., & Manning, C. D. (2014). *Recursive Neural Networks Can Learn Logical Semantics*. Retrieved from http://arxiv.org/abs/1406.1827

Caballero, J., Grier, C., Kreibich, C., Paxson, V., & Berkeley, U. C. (2011). Measuring Pay-per-Install : The Commoditization of Malware Distribution. *Proceedings of the 20th USENIX Conference on Security SEC '11*, (May), 13–13. Retrieved from http://www.icir.org/vern/papers/ppi-usesec11.pdf

Canali, D., Cova, M., Vigna, G., & Kruegel, C. (2011). Prophiler : A Fast Filter for the Large-Scale Detection of Malicious Web Pages Categories and Subject Descriptors. *WWW '11 Proceedings of the 20th International Conference on World Wide Web*, 197–206. https://doi.org/10.1145/1963405.1963436

Cherepanov, A. (2017). WIN32/INDUSTROYER: A new threat for industrial control systems. *Eset*. Retrieved from https://www.welivesecurity.com/wp-content/uploads/2017/06/Win32_Industroyer.pdf

Chua, Z. L., Shen, S. Q., Saxena, P., & Liang, Z. K. (2017). Neural Nets Can Learn Function Type Signatures From Binaries. *Proceedings of the 26th Usenix Security Symposium (Usenix Security '17)*, 99–116.

Cortes, C., & Vapnik, V. (1995). In Silico Log P Prediction for a Large Data Set with Support Vector Machines, Radial Basis Neural Networks and Multiple Linear Regression. *Chemical Biology & Drug Design*, *20*, 273–297. https://doi.org/10.1111/j.1747-0285.2009.00840.x

Dong, Y., Zhang, Y., Ma, H., Wu, Q., Liu, Q., Wang, K., & Wang, W. (2018). An adaptive system for detecting malicious queries in web attacks. *Science China Information Sciences*, *61*(3), 1–16. https://doi.org/10.1007/s11432-017-9288-4

Drucker, H., Member, S., Wu, D., Member, S., & Vapnik, V. N. (1999). Support vector machines for spam categorization. - Drucker, Wu, Vapnik - 1999.pdf. *Support Vector Machines for Spam Categorization*, *10*(5), 1048–1054.

Džeroski, S., & Ženko, B. (2004). Is combining classifiers with stacking better than selecting the best one? *Machine Learning*, *54*(3), 255–273. https://doi.org/10.1023/B:MACH.0000015881.36452.6e

Feng, C., Li, T., & Chana, D. (2017). Multi-level Anomaly Detection in Industrial Control Systems via Package Signatures and LSTM Networks. *Proceedings - 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2017*, 261–272. https://doi.org/10.1109/DSN.2017.34

Forrest, S., Hofmeyr, S. A., Somayaji, A., & Longstaff, T. A. (2002). *A sense of self for Unix processes*. 120–128. https://doi.org/10.1109/secpri.1996.502675

Freund, Y., & Schapire, R. R. E. (1996). Experiments with a New Boosting Algorithm. *International Conference on Machine Learning*, 148–156. https://doi.org/10.1.1.133.1040

Gholami, R., & Fakhari, N. (2017). Support Vector Machine: Principles, Parameters, and Applications. In *Handbook of Neural Computation* (1st ed.). https://doi.org/10.1016/B978-0-12-811318-9.00027-2

Haas, E. (1953). Beitrag zum Morbus Hodgkin und seiner Studieneinteilung. *Klinische Wochenschrift*, *31*(29–30), 694–697. https://doi.org/10.1007/BF01473650

Halfond, W. G. J., & Orso, A. (2005). AMNESIA: Analysis and Monitoring for NEutralizing SQL-injection Attacks. *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, 174–183. https://doi.org/10.1145/1101908.1101935

Hu, J., Lu, J., & Tan, Y. P. (2014). Discriminative deep metric learning for face verification in the wild. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1875–1882. https://doi.org/10.1109/CVPR.2014.242

Huang, W., & Stokes, J. W. (2016). MtNet: A multi-task neural network for dynamic malware classification. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9721*, 399–418. https://doi.org/10.1007/978-3-319-40667-1_20

Irsoy, O., & Cardie, C. (2014). Deep recursive neural networks for compositionality in language. *Proceedings of The Neural Information Processing Systems*, 2096–

2104. Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-84937828128&partnerID=tZOtx3y1

Jordaney, R., Sharad, K., Dash, S. K., Wang, Z., Papini, D., Nouretdinov, I., & Cavallaro, L. (2017). Transcend: Detecting Concept Drift in Malware Classification Models. *Proceedings of the 26th Usenix Security Symposium (Usenix Security '17)*, 625–642. Retrieved from https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jordaney

Kapravelos, A., Shoshitaishvili, Y., Cova, M., & others. (2013). Revolver: An Automated Approach to the Detection of Evasive Web-based Malware. *Proceedings of the 22nd USENIX Security Symposium (USENIX Security)*, 637–652. Retrieved from http://0b4af6cdc2f0c5998459-c0245c5c937c5dedcca3f1764ecc9b2f.r43.cf2.rackcdn.com/12330-sec13-paper_kapravelos.pdf

Karnouskos, S. (2011). Stuxnet worm impact on industrial cyber-physical system security. *IECON Proceedings (Industrial Electronics Conference)*, 4490–4494. https://doi.org/10.1109/IECON.2011.6120048

Kriesel, D. (2005). A Brief Introduction toNeural Networks. *University of Bonn Seminar Proceedings*. Retrieved from http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf

Kruegel, C., & Vigna, G. (2003). Anomaly detection of web-based attacks. *Proceedings*

of the 10th ACM Conference on Computer and Communication Security - CCS '03, 251. https://doi.org/10.1145/948143.948144

Lai, H., Pan, Y., Liu, Y., & Yan, S. (2015). Simultaneous feature learning and hash coding with deep neural networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *07-12-June*, 3270–3278. https://doi.org/10.1109/CVPR.2015.7298947

Lane, T., & Brodley, C. E. (2002). Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, *2*(3), 295–331. https://doi.org/10.1145/322510.322526

Le, H., Pham, Q., Sahoo, D., & Hoi, S. C. H. (2018). *URLNet: Learning a URL Representation with Deep Learning for Malicious URL Detection*. (i). Retrieved from http://arxiv.org/abs/1802.03162

Lee, W., Stolfo, S. J., & Mok, K. W. (1999). A data mining framework for building intrusion. *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium On*, 120–132.

Leo, B. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140. Retrieved from http://dx.doi.org/10.1007/BF00058655

Li, Y., Gao, J., Li, Q., & Fan, W. (2014). Ensemble learning. *Data Classification: Algorithms and Applications*, 483–509. https://doi.org/10.1201/b17320

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. *Icml '13*, *28*, 6. Retrieved from http://www.stanford.edu/~awni/papers/relu_hybrid_icml2013_final.pdf

Meng, Y., & Kwok, L. (2013). Journal of Network and Computer Applications Adaptive blacklist-based packet fi lter with a statistic-based approach in network intrusion detection $. *Journal of Network and Computer Applications*, 1–10. https://doi.org/10.1016/j.jnca.2013.05.009

Moser, A., Kruegel, C., & Kirda, E. (2007). Limits of static analysis for malware detection. *Proceedings - Annual Computer Security Applications Conference, ACSAC*, 421–430. https://doi.org/10.1109/ACSAC.2007.21

Nagpal, B., Chauhan, N., & Singh, N. (2017). SECSIX: security engine for CSRF, SQL injection and XSS attacks. *International Journal of Systems Assurance Engineering and Management*, *8*, 631–644. https://doi.org/10.1007/s13198-016-0489-0

Noble, W. S. (2006). What is a support vector machine? *Nature Biotechnology*, *24*(12), 1565–1567. https://doi.org/10.1038/nbt1206-1565

Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*. 1–20. Retrieved from http://arxiv.org/abs/1811.03378

Olejnik, K., Dacosta, I., Machado, J. S., Huguenin, K., Olejnik, K., Dacosta, I., … Machado, J. S. (2017). *SmarPer : Context-Aware and Automatic Runtime-Permissions for Mobile Devices To cite this version : HAL Id : hal-01489684 SmarPer : Context-Aware and Automatic Runtime-Permissions for Mobile Devices*.

Oza, N. C. (2000). Online ensemble learning. *Aaai/Iaai*, *6837*, 1109–1109. https://doi.org/10.1007/978-3-642-22763-9_7

Pinzón, C., De Paz, J. F., Bajo, J., Herrero, Á., & Corchado, E. (2010). AIIDA-SQL: An

Adaptive Intelligent Intrusion Detector Agent for detecting SQL injection attacks. *2010 10th International Conference on Hybrid Intelligent Systems, HIS 2010*, (September), 73–78. https://doi.org/10.1109/HIS.2010.5600026

Quinlan, J. R. (2006). *Bagging? Boosting? and C??? 5*(Quinlan 1993).

Robertson, W., Vigna, G., Kruegel, C., & Kemmerer, R. A. (2005). *Using Generalization and Characterization Techniques in the Anomaly-based Detection of Web Attacks*.

Rokach, L. (2010). Ensemble-based classifiers. *Artificial Intelligence Review*, *33*(1–2), 1–39. https://doi.org/10.1007/s10462-009-9124-7

Shen, Y., Mariconti, E., Vervier, P. A., & Stringhini, G. (2018). Tiresias: Predicting Security Events Through Deep Learning. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security - CCS '18*, *14*, 592–605. https://doi.org/10.1145/3243734.3243811

Sinclair, C., Pierce, L., & Matzner, S. (1999). An application of machine learning to network intrusion detection. *Proceedings - Annual Computer Security Applications Conference, ACSAC*, *Part F1334*(0293), 371–377. https://doi.org/10.1109/CSAC.1999.816048

Socher, R., Chen, D., Manning, C., Chen, D., & Ng, A. (2013). NTN:Reasoning With Neural Tensor Networks for Knowledge Base Completion. *Neural Information Processing Systems (2003)*, 926–934. https://doi.org/10.1023/A:1004554217069

Swarnkar, M., & Hubballi, N. (2016). *OCPAD : One class Naive Bayes classifier for payload based anomaly detection*. *64*, 330–339. https://doi.org/10.1016/j.eswa.2016.07.036

Tan, A. C., & Gilbert, D. (2003). Data for Cancer Classification. *Applied Bioinformatics*, *2*, 1–10. https://doi.org/10.1103/PhysRevLett.111.215303

Thomas, A. (1995). An introduction to neural networks for beginners. *Clinical EEG and Neuroscience*, *26*(3), 180–183. https://doi.org/10.1177/155005949502600310

Threats, E. S., & Young, A. (1996). Cryptovirology : 1 Introduction 2 Background. *Organization*.

Unknown. (2007). *An inquiry into the nature and causes of the wealth of internet miscreants*. 375. https://doi.org/10.1145/1315245.1315292

Uwagbole, S. O., Buchanan, W. J., & Fan, L. (2017). Applied Machine Learning predictive analytics to SQL Injection Attack detection and prevention. *Proceedings of the IM 2017 - 2017 IFIP/IEEE International Symposium on Integrated Network and Service Management*, 1087–1090. https://doi.org/10.23919/INM.2017.7987433

Wang, H., Ma, C., & Zhou, L. (2009). A brief review of machine learning and its application. *Proceedings - 2009 International Conference on Information Engineering and Computer Science, ICIECS 2009*. https://doi.org/10.1109/ICIECS.2009.5362936

Wang, K., & Stolfo, S. J. (2007). *Anagram : A Content Anomaly Detector Resistant to*.

Wang, W., Guyet, T., Quiniou, R., Cordier, M. O., Masseglia, F., & Zhang, X. (2014). Autonomic intrusion detection: Adaptively detecting anomalies over unlabeled audit data streams in computer networks. *Knowledge-Based Systems*, *70*, 103–117. https://doi.org/10.1016/j.knosys.2014.06.018

Warrender, C., Forrest, S., & Pearlmutter, B. (1999). Detecting intrusions using system

calls: Alternative data models. *Proceedings - IEEE Symposium on Security and Privacy*, *1999-Janua*, 133–145. https://doi.org/10.1109/SECPRI.1999.766910

Wressnegger, C. (2018). *Efficient Machine Learning for Attack Detection*. (November).

Yoneda-Kato, N., Look, A. T., Kirstein, M. N., Valentine, M. B., Raimondi, S. C., Cohen, K. J., … Morris, S. W. (1996). The t(3;5)(q25.1;q34) of myelodysplastic syndrome and acute myeloid leukemia produces a novel fusion gene, NPM-MLF1. *Oncogene*, *12*(2), 265–275. https://doi.org/10.1007/3-540-45014-9_1

Young, A. L., & Yung, M. (2017). Cryptovirology. *Communications of the ACM*, *60*(7), 24–26. https://doi.org/10.1145/3097347

Zeiler, M. D., Ranzato, M., Monga, R., Mao, M., Yang, K., Le, Q. V, … Hinton, G. E. (2013). On rectified linear units for speech recognition. *New York*, 3517–3521.

Zhang, W., Yoshida, T., & Tang, X. (2011). A comparative study of TF*IDF, LSI and multi-words for text classification. *Expert Systems with Applications*, *38*(3), 2758–2765. https://doi.org/10.1016/j.eswa.2010.08.066