VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

☆ ☆ ☆

GRADUATION THESIS

BUILDING A MACHINE LEARNING-BASED WEB APPLICATION FIREWALL

COMMITTEE: COMPUTER SCIENCE 3

Supervisors:	Dr. Nguyen An Khuong
	Mr. Le Dinh Thuan, M. Sc.
	Mr. Le Nguyen Minh Khoi, B. Eng.
	Mr. Nguyen Van Hoa, B. Eng.
Reviewer:	Mr. Tran Ngoc Bao Duy, M. Sc.

Student 1:	Trinh Cong Vu	1852882
Student 2:	Le Thien Thanh	1814009

Ho Chi Minh City, June 2023

Commitment

We vouch for the fact that the work in this dissertation was completed following the guidelines established by the university and was not completed for submission to any other academic bodies. The works are our own unless otherwise noted by a particular citation in the text.

Ho Chi Minh City, June 2023

Acknowledgement

First and foremost, we want to express our appreciation to our supervisors, Professor Nguyen An Khuong and Le Dinh Thuan, for their patience, inspiration, and extensive expertise, all of which were crucial in assisting us with the thesis. Throughout the entire process of working on and producing the thesis, their advice was really helpful.

We also want to express our thankfulness to Le Nguyen Minh Khoi and Nguyen Van Hoa for their particular assistance and important information sharing. We cannot complete a fantastic thesis without their assistance.

In addition, we want to express our gratitude to our families and friends who have helped and supported us greatly during this thesis and our time at the university.

Thesis group members

Preface

With the increased exchange of information and other activity on the World Wide Web, the Web has become the primary platform for attackers to cause havoc. Effective methods for detecting Web threats are crucial for ensuring Web security. With the explosion of data, it is becoming more challenging to manually detect and prevent cyber threats; however, the use of machine learning-based ways to fight cybersecurity is on the rise, with broad uses of machine learning in cybersecurity providers.

In this thesis, we investigated and evaluated the approaches of machine learning in WAF, evaluating the benefits and drawbacks of each related work in recent times. Then we proposed a machine learning-based WAF to defend web applications from cyber attacks. Typically, rule-based WAFs have been widely employed. They do, however, have a significant false-positive rate. As a consequence, we are building a machine learning-based WAF to validate requests that have been labeled suspicious by WAFs to enhance the WAFs and provide better surveillance.

The malicious request validator is based on the assumption that genuine requests to a website generally fit into the same category. The module uses a Convolutional Neural Network to classify the suspected request and evaluates whether it falls into the same category as the standard requests observed. The ultimate decision on whether to block a request is made using this result and combining it with the Logistic Regression request content analyzer.

The false positive rate (or the false alarm rate) is the most crucial requirement for modules in experimentation. Precision and accuracy are other key criteria. We also accounted for processing time with the suspicious request detector, as WAFs are supposed to be instant. The best experimental results for our machine learning-based WAF tested on the CSIC 2010 request dataset is almost a zero false positive rate with an accuracy of 85.23%, compared to only 66.98% when we combined the machine learning models with rule-based WAF, in this case, ModSecurity. However, our WAF is still not competitive enough with other studies in the same WAF approach, nor does it meet the response time criteria. And it is clear from the aforementioned experiments that the machine learning-based WAF approach has certain advantages over the other techniques and can be applied adaptable in a variety of circumstances.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

The following list shows the abbreviations that will be used in the text of this thesis.

Adam	Adaptive Moment Estimation
CNN	Convolutional Neural Network
IDS	Intrusion Detection System
JSON	JavaScript Object Notation
LSTM	Long Short-Term Memory
ML	Machine Learning
NLP	Natural Language Processing
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SQLi	SQL Injection
SVM	Support Vector Machine
TF-IDF	Term Frequency - Inverse Document Frequency
WAF	Web Application Firewall
XSS	Cross-site Scripting

1

Introduction

In this chapter, we introduce issues in the field of cyber security and clarify the necessity of building a firewall system. Next, we present an overview of the goals, challenges, and structure of the thesis.

Table of Contents

1.1 Motivation and problem statement

Websites have become necessary for every business, brand, institution, organization, and individual. Web-based applications offer the general public and enterprises fast and simple services, they may be used for social media, email, banking, online shopping, education, or entertainment. Web applications are popular for a variety of reasons[1]. For starters, web-based applications are convenient for users because they can be accessed from any location that has an internet connection. Second, a web application is a low-cost method for corporations because it does not require any special hardware or software and can be easily scaled up or down as needed. Furthermore, web-based apps frequently offer subscription-based models, which can be more cost-effective for businesses in the long run. Third, because web apps do not need to be downloaded or installed on a device, they are generally faster and more responsive than native apps. These are instead run on remote servers and accessed through a web browser. As a result, web apps can fully utilize the most recent advances in server-side technology, which helps in performance and speed.

Why would someone want to attack web applications? Web applications are an attractive target for remote attackers because they can access the web app anywhere with an internet connection. Besides, they frequently handle sensitive information, such as login credentials and financial data. In short, web applications are targeted because they are visible, accessible, and offer a multitude of potential payoffs for an attacker. Cybercriminals can attack web applications for many reasons, including system flaws caused by incorrect coding, misconfigured web servers, application design flaws, or failure to validate forms. These flaws and vulnerabilities enable attackers to gain access to databases containing sensitive information.

The action conducted by attackers, which can do harm to the web application can be called cyberattacks. Cyberattacks are malicious attempts to access a person's or an organization's computer systems, networks, or data without authorization. The vulnerability to cyberattacks and our reliance on technology and connectivity are growing in tandem. Unlike viruses that would shut down a system for a few hours a few years ago, the consequences of cyberattacks today can include stolen data, destroyed networks, and thousands, if not millions, of dollars in recovery efforts. Cyberattacks can harm businesses, governments, and society[2]. Among them, businesses and companies are more prone to being attacked by cybercriminals because they have more holes and gaps in their security that make them vulnerable to attacks. Cyberattacks affect a company's productivity, revenue, and reputation. Regarding productivity, nearly every business that suffers a cyberattack must suspend part or all of its operations until the attack is resolved, whether by paying a ransom, removing the malware from the device, network, or system, or restoring a backed-up version of its system. In terms of revenue, the costs of a cyberattack can wreck the economy of a company. The average cost of a data breach for a small to medium-sized business is massive, whether it has to shut down operations for several days, pay a ransom, lose data, replace devices, or pay a security expert to clean all malware out of the system or network. The most serious effect of a cyberattack is the loss of reputation. The most critical consequence of a cyberattack on a business is a loss of reputation. Consider the recent data breaches at Equifax, Target, and J.P. Morgan Chase, which led to the loss of

[1] Amy Bros. Sysprobs. *Why are Web Applications Becoming Popular?*. July 2022. https://www.sysprobs.com/web-applications-becoming-popular

[2] CEI-The Digital Office. *The Consequences Of Cyber Attacks And Their Impact On Cybersecurity*. https://www.copycei.com/consequences-of-cyber-attacks

customer data such as social security numbers, account details, and credit card numbers. Despite having the resources to recover, most businesses can not recover from security breaches because they lose their clients' trust and thus business. The biggest concern of a cyberattack on a government entity is the enormous volume of stolen data. This data could include everything from military and national security information to private data about civilians, which could be sold on the dark web and misused by terrorist groups. When cyberattacks happen, they badly affect practically every part of society, whether it's a large corporation or a small business. Consumers pay the price when enterprises, companies, and even nonprofit service providers like hospitals are forced to cover the costs of a cyberattack. There will be shortages that the customer will have to suffer when a company is restricted from providing its service as a result of cyberattacks or data breaches.

Let's take a look at Parachute's statistics[3]. Healthcare, throughout the past 12 years, this sector has experienced the most costly data breaches, the costs have even increased by 41.6% from 2020 until 2022. At least 849 healthcare cybersecurity incidents and 571 data breaches were reported in 2022. The average financial loss due to data breaches in healthcare has skyrocketed from around USD 9 million to USD 10.10 million (2022). In the Finance industry, phishing attacks against banks and other financial institutions held the largest share, accounting for 23.2% of all cyberattacks targeting the financial sector. In the first quarter of 2022, ransomware assaults increased by 35% in the financial sectors. On average, financial organizations bore the second-highest data breach costs, at USD 5.97 million, just behind healthcare institutions (2022).

From the above situation, we can see that tools like Web Application Firewalls play a crucial role in organizations' security systems. They are a powerful tool for preventing malicious traffic from entering or leaving an organization's systems. However, they also have some limitations[4]. The first drawback is false positives, a WAF may block legitimate traffic or requests if it mistakes them for malicious activity. Second, a WAF only protects against web-based attacks and may not be effective against other types of attacks. Next, a WAF can add overhead to the application and may affect its performance, which can be a concern for critical applications with high traffic volumes or strict performance requirements. We've found some studies about using machine learning to improve WAF, or building a machine learning-based WAF. Most of them have very positive results. The necessity of WAF in modern days and these studies have inspired us, therefore in this thesis, our group wants to make some efforts to strengthen WAFs' cyber security, and we decided to choose this topic - Building a Machine Learning-Based Web Application Firewall. This thesis will compare the existing methods of applying machine learning to WAF and build a WAF with machine learning based on the more effective approach.

[3]Parachute. *Cyber Attack Statistics to Know in 2023*. https://parachute.cloud/cyber-attack-statistics-data-and-trends/

[4]Sourcedefense. *Limitations of WAFs*. https://sourcedefense.com/glossary/limitations-of-waf/

1.2 Scope and objectives

In this thesis, we will investigate recent studies about WAF, then divide them into main approaches according to the method used in the research. We will try to compare and analyze the main directions of machine learning in WAF. Based on the experiments with minimal models, false positive rate, and processing form of model size, we will try to prove which approach is more effective.

Cybersecurity needs to be enhanced because of the potential repercussions that cybercriminals could have. We aim to improve the capabilities of web application firewalls given the current need for cybersecurity. So, we decide to build a machine learning-based web application firewall. This thesis will focus on recognizing malicious requests that get past the firewall. We want to use machine learning to create a WAF processing a massive volume of requests per second quickly and effectively. The created module ought to be able to validate the request and determine whether it's an attack or not.

We also aim to eliminate the high false positive (FP) rate, an inherent weakness of rule-based WAF, and assure the speed of the WAF, which must react nearly instantly to deliver a consistent customer experience, the module will use two fast and simple machine learning models. Each model will run independently to generate an output, then combined the result to achieve an accuracy of at least 95% as well as low latency of 5 milliseconds[5] for the WAF.

1.3 Challenges and solutions

During this thesis, we have encountered challenge with the datasets. The datasets of requests tend to be confidential because they might include personal user information. Therefore, there are not many freely accessible datasets. Additionally, labeling these datasets is an expensive job that calls for specialized knowledge. The datasets of requests are randomly generated, thus may not be practical and may not display the complexity of network requests.

About the datasets challenge, to supplement and update new HTTP access data which is suitable for practical applications, we tried to combine and come up with a way to utilize other URLs datasets which are abundant, and proposed an approach that uses structural languages classification as a part of the request validator to shift the weight of the lacking and impractical request dataset to other datasets.

[5]Abdalslam. *Web Application Firewalls (WAF) Statistics, Trends And Facts 2023*. https://abdalslam.com/web-application-firewalls-waf-statistics

1.4 Tentative structure of the thesis

The structure of the thesis is divided into eight chapters, a summary of the content of each chapter is as follows.

Chapter 1 introduces issues in the field of cyber security, and clarifies the necessity of building a firewall system. Next, we present an overview of the goals, challenges, and structure of the thesis.

Chapter 2 introduces the background knowledge of this thesis, including information about Web Application Firewalls, Machine Learning Models, HTTP requests, Natural Language Processing, and Deep Neural Networks.

Chapter 3 mentions some related studies on the use of machine learning techniques to detect malicious requests, their strengths and weaknesses, and evaluate an appropriate approach for this thesis.

Chapter 4 displays the problems as well as the malicious request validator's input and output. Then we offer the design and architecture of the problem's solutions.

Chapter 5 describes the implementation steps of the systems including system frameworks.

Chapter 6 describes the datasets which we use to train and evaluate the Logistic Regression module and CNN categorize module, with the problems of datasets and pre-processing details.

Chapter 7 discusses the division of the data set before training, assessment techniques, and experimental results. Compare the outcomes of the experiments we suggest to the reference experiments from relevant works.

Chapter 8 summarizes the results for the thesis. Finally, we want to present future improvements.

2

Background

This chapter introduces the background knowledge of this thesis, including information about Web Application Firewall, Machine Learning Models, HTTP request, Natural Language Processing, and Deep Neural Networks.

Table of Contents

2.1 Web Application Firewall

A Web Application Firewall helps protect web applications by filtering and monitoring HTTP traffic between a web application and the Internet. It typically protects web applications from attacks such as cross-site forgery, cross-site-scripting (XSS), file inclusion, and SQL injection, among others.

2.1.1 Definition

WAF stands for **Web Application Firewall**. This firewall solution commonly monitors data packets and filters them for the presence of malware or viruses. It performs the data monitoring/filtering for to and from data packets.

The WAF tool can be distributed using network-based, cloud-based, or host-based architectures. It needs a reverse proxy to make sure that one or more web apps are in front of it while facing forward. It can be utilized either alone or in conjunction with other applications. WAF may function at a lower level or a higher level depending on the requirement[1].

2.1.2 The functionality of WAF

As previously stated, WAF is deployed at the application layer and serves as a two-way firewall. At work, WAF monitors HTTP or HTTPS traffic entering or exiting a specific web app. When WAF detects a malicious object in the traffic, it activates and destroys it. Figure 2.1 demonstrates how WAF works, legitimate users (top-left and bottom-left) are permitted access to the server with a WAF enabled, but attackers (middle-left) are prevented from doing so.

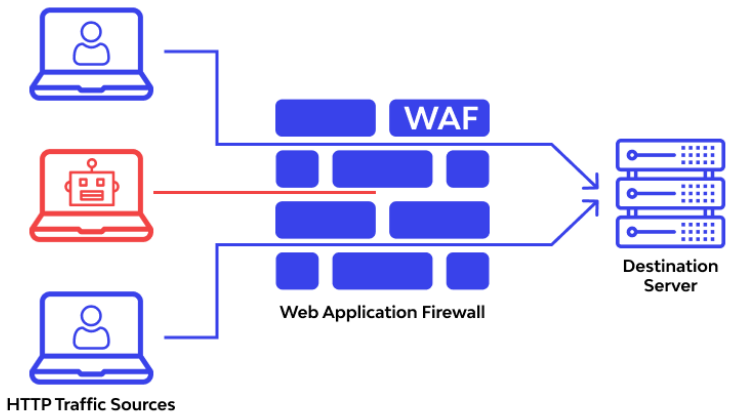


Figure 2.1: How does WAF work (source: https://www.wallarm.com/what/waf-meaning)

[1] Wallarm. *WAF Meaning*. https://www.wallarm.com/what/waf-meaning

WAF predefined what is malicious and what is not to make the process easier. WAF adheres to these guidelines throughout the process. WAF primarily analyzes the GET and POST portions of HTTP traffic. GET retrieves data from the server, whereas POST directs data to the server to change its original state[2].

2.1.3 Differences between WAF and firewall

In the modern era of sophisticated cyberattacks and digital innovation, it is essential for organizations to understand the threats they face and what their security measures protect them from. Understanding the value of and distinctions between WAF security and network firewall security is vital for preventing online attacks and other types of network intrusions[3].

A web application firewall (WAF) protects web applications by intercepting Hypertext Transfer Protocol (HTTP) traffic. This is distinct from a traditional firewall, which acts as a wall between external and internal network traffic.

A WAF stands between external users and web applications to track all HTTP traffic. It then identifies and stops harmful requests from entering users or apps on the web. As a result, WAFs protect key company online applications and servers from zero-day threats and other application-layer attacks. This becomes highly critical as firms invest in new digital efforts, which could expose new web apps and APIs to attacks.

A network firewall guards a secure local-area network against unwanted access to reduce the risk of assaults. Its goal is to distinguish a safe zone from a less secure zone and to control communication between the two. Without it, every device that has a public IP address is exposed to the outside network and potentially vulnerable to attack.

The layer of security that WAF and network-level firewalls operate on is the primary technological distinction between them. Attacks at OSI model Layer 7, or the application level, are protected by WAFs. This covers URL assaults, cookie manipulation, SQL injection, and attacks against JavaScript, ActiveX, and Ajax applications. They also target HTTP and HTTPS, the web application protocols that link web browsers and web servers. Network firewalls secure data transfer and network traffic at OSI model Layers 3 and 4. This covers assaults on the Domain Name System (DNS) and File Transfer Protocol (FTP), as well as Telnet, Secure Shell (SSH), and Simple Mail Transfer Protocol (SMTP). The following figure (Figure 2.2) displays the attacks which can be blocked by network firewall and WAF.

[2]Wallarm. *WAF Meaning*. https://www.wallarm.com/what/waf-meaning

[3]Fortinet. *WAF vs. Firewall: Web Application and Network Firewalls*. https://www.fortinet.com/resources/cyberglossary/waf-vs-firewall

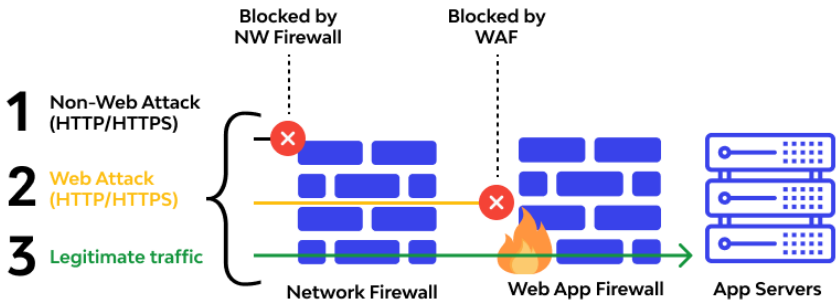


Figure 2.2: WAF and network firewall block attacks (source: https://www.wallarm.com/what/waf-meaning)

It's critical to pick a suitable network firewall or WAF to protect against all of the risks that may be present. Businesses cannot be protected from web page attacks by a network firewall alone; WAF capabilities are the sole means of defense. Business organizations risk leaving their larger network vulnerable to attack due to web application vulnerabilities without an application firewall. A WAF cannot protect from attacks at the network layer, so it should supplement a network firewall rather than replace it. Network-based and web-based solutions operate at several layers and protect from multiple types of traffic. As a result, they perform best together rather than against one another. A network firewall usually protects a wider range of traffic types, but a WAF deals with a specific threat that a conventional strategy cannot handle. Having both options is therefore advisable, especially if a company's operating systems and the web interact frequently.

2.1.4 False positive

2.1.4.1 Definition

An alert produced when there is no real danger is known as a false positive (FP). In other words, it serves as a warning indicator but ultimately proves to be a false alarm. False positives may appear to be innocuous annoyances, but they can have harmful effects[4]. For instance, when it discovers unusual behavior on a network, an intrusion detection system (IDS) may send out an alert. However, additional research reveals that the behavior is harmless—a false positive. For IT security teams who have to waste time looking into them, they are frequently the result of wrong settings or overly vigilant security software.

[4]Hannah Brice. *The Dangers of False Positives in Cybersecurity and how to avoid them.* Oct 2022. https://www.lupovis.io/the-dangers-of-false-positives-in-cybersecurity-and-how-to-avoid-them/

Handling a large number of notifications is something that all WAF specialists have experience with. They're probably also wasting a ton of time sorting out false positives from these warnings. Attacks are to be stopped, while valid traffic is to be allowed to pass through the WAF. False positive incidents clog the alerts feed and, worse yet, block legitimate traffic. A few false positive incidents happen because of flaws or poor application design. A WAF rule that is too general or doesn't fit the site's operation may trigger additional events[5].

2.1.4.2 The risks of false positive

Any expanding company will worry about scalability, and expanding development processes presents several difficulties. Small-scale development still frequently uses ad hoc toolkits and manual procedures, while the former can still produce too many false positives.

The number of false positives can expand exponentially, and it is hard to handle them all manually, as upgrades, products, and workloads all continue to multiply.

There may be substantial financial repercussions as well. It can take too long to investigate reports that turn out to be false positives, which can result in delays and a possible loss of money and business possibilities.

Due to the overwhelming amount of false positives, staff members could grow accustomed to ignoring reports, increasing the likelihood that an actual vulnerability will go unnoticed and enter the production application, again with very costly results[6].

2.2 HTTP request

2.2.1 Normal request

HTTP request is information sent from the client to the server, to ask the server to find or process some information and data that the client wants. The HTTP request can be a text file in the form of XML or JSON that both can understand. Here is an HTTP request using the POST method (Figure 2.3).

Request URL:	https://www.facebook.com/api/graphql/
Request Method:	POST
Status Code:	200
Remote Address:	[2a03:2880:f15c:83:face:b00c:0:25de]:443
Referrer Policy:	strict-origin-when-cross-origin

Figure 2.3: Normal HTTP request using POST method

[5] A. Lerner, N. Avital, S. Margel. *Alert fatigue - introducing false positives in WAF*. Aug 2020. https://www.imperva.com/blog/avoid-alert-fatigue-how-to-automatically-get-rid-of-waf-false-positive/

[6] Ritika Singh. *The Risks Of False Positives With Web Application Firewalls*. Sep 2021. https://www.indusface.com/blog/the-risks-of-false-positives-with-web-application-firewalls/

An HTTP request consists of.

- **Request line.** It begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF.
- **Body request.** It can be plain text, HTML, XML, JSON, Javascript, or a set of form-data key-value pairs.

2.2.1.1 Request line

Request line is the first line of an HTTP request. It includes.

- **HTTP method.** There are many types, but the most common are GET and POST.
- **URI (Uniform Resource Identifier).** It helps identify the resources requested by the client.
- **HTTP version.** The version of the HTTP protocol.

2.2.1.2 Body request

Allows the client to send additional requests that the server needs to do, such as creating or updating data that cannot be passed on the Header Parameters. Request body is often used in POST, PUT, PATCH methods.

2.2.2 Malicious request

Malicious traffic or malicious network traffic is any suspicious link, file or connection that is being created or received over the network. Malicious traffic is a threat with an organization's security or may compromise personal computers.

An example of a request used to attack Facebook is a cross-site scripting (XSS) attack (Figure 2.4). In this attack, a hacker can use a malicious HTTP request to inject JavaScript code into Facebook websites. When a user visits Facebook's website, JavaScript code is executed and allows the hacker to obtain the user's logins, transactions, and personal information.

```
POST /update_status.php HTTP/1.1
Host: www.facebook.com
Content-Type: application/x-www-form-urlencoded

status=Hello+World+%3Cscript%3Ealert(%27you+have+been+hacked%27)%3C/script%3E&post=1
```

Figure 2.4: Malicious HTTP request used to attack Facebook

In it, the status field contains malicious HTML and JavaScript characters. When a user visits Facebook's website, this JavaScript code is executed and displays a malicious warning on the user's screen.

This can cause unsuspecting users to log into their accounts. Upon successful login, the user's credentials will be captured and used by the hacker to access their account and perform other malicious activities.

2.3 Natural processing language

The field of computer science known as *Natural Language Processing* (NLP) is more particularly the branch of artificial intelligence (AI) that is concerned with providing computers the ability to understand spoken and written language like that of humans. Its practical applications include spam detection, translation, and sentiment analysis.

NLP blends statistical, machine learning, and deep learning models with computational linguistics—rule-based modeling of human language. With the use of these technologies, computers are now able to process human language in the form of text or audio data and fully "understand" what is being said or written, including the speaker's or writer's intentions and sentiments.

2.3.1 Tokenization

Tokenization is used in natural language processing to split paragraphs and sentences into smaller units that can be more easily assigned meaning. The first step of the NLP process is gathering the data (a sentence) and breaking it into understandable parts (words).

There are 3 types of tokenization.

- **Word tokenization.** The most used type of tokenization is word tokenization. It employs delimiters (characters like ',' or ';' or ",") to divide the data into its corresponding words when there are natural breaks, like pauses in voice or spaces in the text. Although this is the simplest method for breaking up voice or text into component pieces, it has several disadvantages.
- **Character tokenization.** Tokenization of characters was developed to solve some of the problems associated with word tokenization. It entirely splits text into characters rather than dividing it up into words. In contrast to word tokenization, this enables the tokenization process to maintain information about OOV terms.
- **Sub-word tokenization.** Similar to word tokenization, sub-word tokenization uses specific linguistic rules to further dissect individual words. They often use cutting-off affixes as one of their main tools. Prefixes, suffixes, and infixes can aid programs in comprehending the function of a word because they alter the basic meaning of words. This can be particularly helpful for terms not in your lexicon because figuring out an affix can provide a program with more understanding of how unfamiliar words work.

2.3.2 Vectorization

Vectorization is a classic approach to converting input data from its raw format (i.e. text) into vectors of real numbers which is the format that ML models support. In machine learning, vectorization is a step in feature extraction. The idea is to get some distinct features out of the text for the model to train on, by converting text to numerical vectors.

Some vectorization techniques used in this thesis.

- **TF-IDF.** It is a numerical statistic that's intended to reflect how important a word is to a document.
- **Word2Vec.** In a nutshell, this approach uses the power of a simple Neural Network to generate word embeddings.

2.4 Machine learning models

In this thesis, we are using two machine learning algorithms to train the model: *Logistic Regression* and *Convolutional Neural Network*.

2.4.1 Logistic regression

One of the most used machine learning algorithms for binary classification is logistic regression. A logit function is used to forecast the likelihood that a binary outcome will occur. Given that it uses the log function to estimate outcome probabilities, it is a specific example of linear regression.

We turn the result into a categorical value using the activation function (sigmoid). Logistic regression has a wide range of applications, including the detection of fraud, spam, and other conditions.

Within machine learning, logistic regression belongs to the family of supervised machine learning models. It is also considered a discriminative model, which means that it attempts to distinguish between classes (or categories)[7].

2.4.1.1 Logistic regression model

Predictive output of Linear Regression is displayed in (2.1).

$$f(x) = w^T x \tag{2.1}$$

The predictive output of logistic regression is generally written as (2.2).

$$f(x) = \theta(w^T x) \tag{2.2}$$

where θ is called the logistic function. The logistic function in linear regression is a type of sigmoid, a class of functions with the same specific properties.

[7]Dinesh Kumawat. *Introduction to Logistic Regression - Sigmoid Function, Code Explanation.* Jan 2021. https://www.analyticssteps.com/blogs/introduction-logistic-regression-sigmoid-function-code-explanation

2.4.1.2 Sigmoid Function

Sigmoid is a mathematical function that takes any real number and maps it to a probability between 1 and 0. The formula of the sigmoid function is shown in (2.3).

$$f(x) = \frac{1}{1+e^{-x}} \triangleq \sigma(x) \tag{2.3}$$

As a result of the sigmoid function's S-shaped graph (displayed in Figure 2.5), the probability increases as x approaches infinity and decreases as x approaches negative infinity. The model establishes a threshold that determines which binary variable corresponds to which probability range[8].

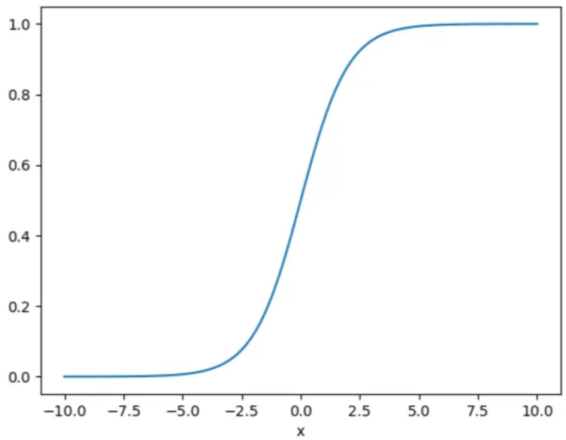


Figure 2.5: Sigmoid function forms an S-shaped graph (source: https://www.educative.io/answers/what-is-sigmoid-and-its-role-in-logistic-regression)

Suppose we have two possible outcomes, normal and abnormal, and have set the threshold as 0.5. A probability less than 0.5 would be mapped to the outcome abnormal, and a probability greater than or equal to 0.5 would be mapped to the outcome normal.

2.4.1.3 Optimizing loss functions

The Stochastic Gradient Descent (SGD) algorithm will be used here. Loss function with only one data point (Equation 2.4).

$$J(w; x_i, y_i) = -(y_i \log z_i + (1-y_i)\log(1-z_i)) \tag{2.4}$$

Using derivative, we have the expression shown in (2.5).

$$\frac{\partial J(w; x_i, y_i)}{\partial w} = \frac{z_i - y_i}{z_i(1-z_i)} \frac{\partial z_i}{\partial w} \tag{2.5}$$

To make this expression more compact and beautiful, we will find the function $z = f(w^T x)$ such that the denominator cancels out. If set $s = w^T x$, we have (2.6).

$$\frac{\partial z_i}{\partial w} = \frac{\partial z_i}{\partial s}\frac{\partial s}{\partial w} = \frac{\partial z_i}{\partial s} x \tag{2.6}$$

[8] Ayesha Naeem. *What is sigmoid and its role in logistic regression?*. https://www.educative.io/answers/what-is-sigmoid-and-its-role-in-logistic-regression

Most intuitively, we will find the function $z = f(s)$ such that (2.7) happens.

$$\frac{\partial z}{\partial s} = z(1 - z) \tag{2.7}$$

to suppress the denominator in the expression (2.5). Expression in (2.7) will be equivalent to.

$$z = \frac{e^s}{1 + e^s} = \frac{1}{1 + e^{-s}} = \sigma(s) \tag{2.8}$$

2.4.1.4 Updated math formula for logistic sigmoid regression

From above equations, we have (2.9).

$$\frac{\partial J(w; x_i, y_i)}{\partial w} = (z_i - y_i)x_i \tag{2.9}$$

The updated formula (according to the SGD algorithm) for logistic regression is displayed in (2.10).

$$w = w + \eta(y_i - z_i)x_i \tag{2.10}$$

2.5 Deep neural network

With their excellent results, broad applicability, and vast growth potential, neural networks are currently the most advanced advancement in artificial intelligence. *Feedforward Neural Networks* (FNN) and *Convolutional Neural Networks* (CNN) are widely used to make predictions with independent data input. In CNN (Figure 2.6), each input image is passed through convolutional layers (Filters, Pooling, and Fully-connected layers) to extract features before being classified using the Softmax function[9].

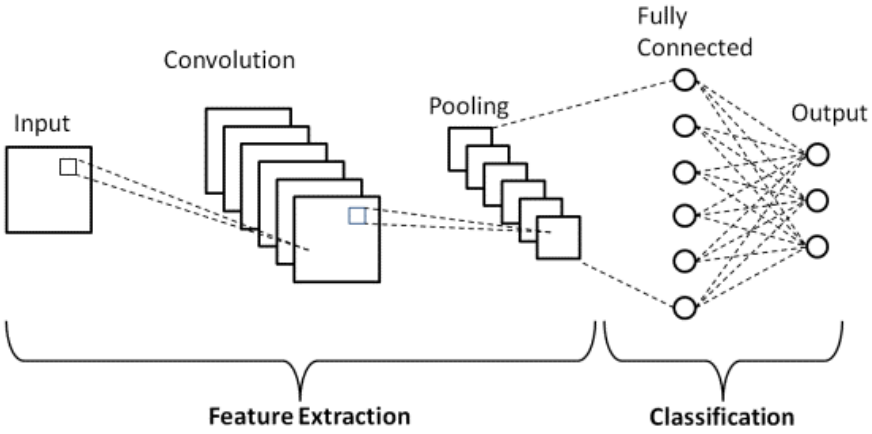


Figure 2.6: CNN architecture (source: https://www.upgrad.com/blog/basic-cnn-architecture/)

[9]Softmax function:$\sigma(\overrightarrow{z})_i = (e^{z_i})/\sum_{j=1}^{K} e_i^z$

Layers are the building components of deep neural networks such as CNN and others. A layer is a broad phrase that refers to a group of "nodes" that function together at a given level within a neural network. Layers are classified into three types: *input layer*, *hidden layers*, and *output layers*.

The *input layer* contains raw data (each variable as a "node"). In neural networks, black magic happens in the *hidden layer*(s). By minimizing an error/cost function, each layer attempts to learn different elements of the data. The context of "image recognition", such as a face, is the most intuitive way to understand these levels. The first layer may learn edge detection, the second eye, the third nose, and so on. This isn't exactly what's going on, but the idea is to split the problem down into components that different levels of abstraction can piece together, much to how our own brains work (hence the name neural networks).

The *output layer* is the simplest, usually consisting of a single output for classification problems. Even though it is a single "node", it is nevertheless regarded as a layer in a neural network because it might contain numerous nodes. We use six types of layers given by the TensorFlow framework in this thesis: **Embedding** layer, **Conv1D** layer, **MaxPooling1D** layer, **Dropout** layer, **Flatten** layer , and **Dense** layer.

2.5.1 Embedding layer

A class of methods for encoding words and documents using a dense vector representation is known as word embedding. It is an advance over the conventional bag-of-words encoding techniques, in which each word was represented by a big sparse vector, or a complete vocabulary was represented by scoring each word within the vector. Due to the enormous vocabulary and the fact that most words and documents were represented by large vectors largely made up of zero values, these representations were sparse. In contrast, words are represented in an embedding by dense vectors, where a vector is the word's projection into a continuous vector space.

On text data, neural networks can be applied using an *embedding layer*. The input data must be integer encoded for each word to be represented by a different number. All of the words in the training dataset will have an embedding learned by the embedding layer, which is initialized with random weights. It is a flexible layer that can be used in a variety of ways, such as.

- It can be used alone to learn a word embedding that can be saved and used in another model later.
- It can be used as part of a deep learning model where the embedding is learned along with the model itself.
- It can be used to load a pre-trained word embedding model, a type of transfer learning.

The embedding layer is defined as the first hidden layer of a network. It must specify 3 arguments.

- **input_dim.** This is the size of the vocabulary in the text data. For example, if your data is integer encoded to values between 0 and 10, then the size of the vocabulary would be 11 words.
- **output_dim.** This is the size of the vector space in which words will be embedded. It defines the size of the output vectors from this layer for each word.
- **input_length.** This is the length of input sequences. For example, if all of the input documents are comprised of 1000 words, this would be 1000.

Weights in the embedding layer are learned. If you save your model to file, this will include weights for the Embedding layer.

A 2D vector with one embedding for each word in the input word sequence (input document) is the result of the embedding layer.

2.5.2 Conv1D layer

CNN convolutional layers use learned filters to produce feature maps that represent the existence of certain features in the input.

A filter is created by concatenating several kernels, each of which is allocated to a different input channel. The difference between filters and kernels is always one dimension. For instance, filters in 1D convolutions are 2D matrices (basically, the kernels are a concatenation of 1D matrices). A kernel is a matrix that is slid across the input and multiplied by the input in order to enhance the output in a desired way.

Convolutional layers are particularly efficient, and stacking them in deep models enables the learning of high-order or more abstract characteristics, such as shapes or particular objects, by layers deeper in the model, while allowing layers near the input to learn low-level features, such as lines, at a faster rate.

Conv1D layer produces a tensor of outputs by creating a convolution kernel and combining it with the layer input over a single spatial (or temporal) dimension.In TensorFlow, the input shape of this layer is a 3+ D tensor with shape: $batch\_shape + (steps, input\_dim)$, while the output shape is a 3+ D tensor with shape: $batch\_shape + (new\_steps, filters)$, *steps* value might have changed due to padding or strides[10].

2.5.3 MaxPooling1D layer

The fact that convolutional layer feature maps preserve the exact location of input features is one of their limitations. This means that tiny movements in the position of the feature in the input image will result in a different feature map. This can happen with re-cropping, rotation, shifting, and other minor adjustments to the supplied image.

Downsampling is a popular strategy used in signal processing to solve this issue. In this case, a reduced resolution version of the input signal is produced, retaining the main structural features but excluding the small details that might not be as helpful.

[10]Keras. *Conv1D layer*. https://keras.io/api/layers/convolution_layers/convolution1d/

By altering the convolution's stride over the image, convolutional layers can be used to do downsampling. Using a pooling layer is a more reliable and popular strategy.

In order to apply a pooling operation on feature maps, similar to a filter, pooling is conducted. The size of the pooling operation or filter, which is typically 2×2 pixels applied with a stride of 2 pixels, is smaller than the size of the feature map. This means that each feature map will always be compressed by a factor of 2, i.e., each dimension is cut in half, making each feature map only contain a quarter as many pixels or values.

Instead of being taught, the pooling operation is defined. A typical function used in the process of pooling is Max pooling, which works by calculating the maximum value for each patch of the feature map, and MaxPooling1D is applied for 1D temporal data. The max pooling is demonstrated in Figure 2.7.

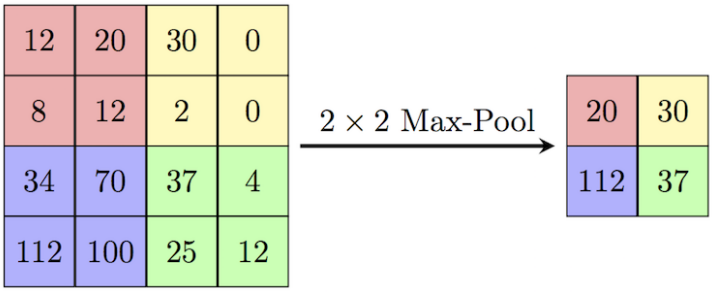
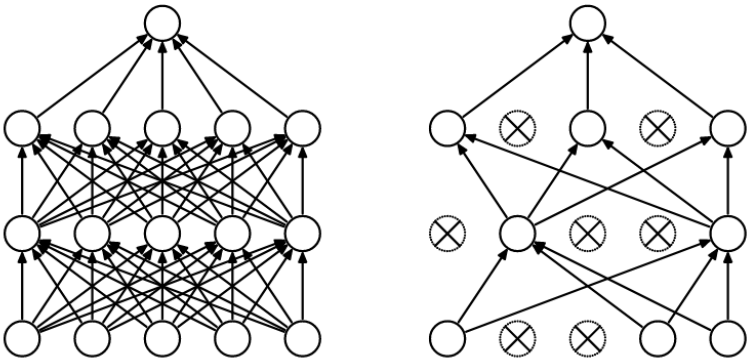


Figure 2.7: Max pooling demonstration (source: https://paperswithcode.com/method/max-pooling)

2.5.4 Dropout layer

Dropout is a training method in which some neurons are ignored at random. They are "dropped out" in a random manner. It means that on the forward pass, their contribution to the activation of downstream neurons is momentarily removed, and on the backward pass, no weight updates are made to the neuron.

If neurons are randomly removed from the network during training, the remaining neurons will have to step in and handle the representation needed to produce predictions for the missing neurons. The network is believed to learn numerous independent internal representations as a result. Dropout procedure is demonstrated in Figure 2.8.



(a) Standard Neural Net (b) After applying dropout.

Figure 2.8: Dropout demonstration (source: https://laid.delanover.com/dropout-explained-and-implementation-in-tensorflow/)

The result is a decrease in the network's sensitivity to the particular neuronal weights. As a result, the network is better able to generalize and is less prone to overfit the training set of data[11]. In Tensorflow, the *dropout layer* randomly sets input units to 0 with a frequency of rate at each step during training time. Inputs not set to 0 are scaled up by 1/(1 - rate) (with rate is the fraction of the input units to drop) such that the sum over all inputs is unchanged[12].

2.5.5 Flatten layer

To enter data into the following layer, flattening converts the data to a 1-dimensional array. To construct a single, lengthy feature vector, the output of the convolutional layers is flattened. The last classification model, referred to as a fully-connected layer, is connected to it as well. In other words, the final layer is connected to the flattened, single-line representation of all the pixel data. The flatten procedure is displayed in Figure 2.9.

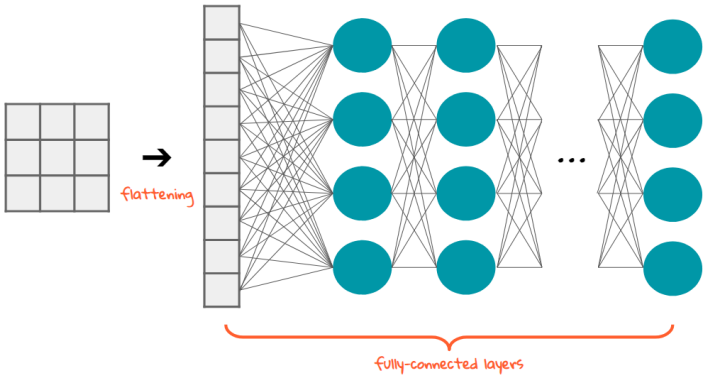


Figure 2.9: Flattening demonstration (source: https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480)

[11] Jason Brownlee. *Dropout Regularization in Deep Learning Models with Keras*. Jul 2022. https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

[12] Keras. *Dropout layer*. https://keras.io/api/layers/regularization_layers/dropout/

2.5.6 Dense layer

Dense layer is a neural network that has deep connection, meaning that each neuron in dense layer receives input from all neurons of its previous layer. The dense layer is found to be the most commonly used layer in the models.

The values utilized in the matrix are parameters that may be trained and updated with the aid of backpropagation, and the dense layer multiplies matrices and vectors. Dense Layer implements the following operation: *output = activation(dot(input, kernel) + bias)*. The dense layer on the output performs *dot product* of *input tensor* (input) and *weight kernel matrix* (kernel). A *bias vector* (bias) is added and *element-wise activation* (activation) is performed on output values.

An 'n' dimensional vector is produced as the dense layer's output. The vector can have its dimensions, rotation, scaling, and translation changed using a dense layer.

2.5.7 Gradient descent

Gradient descent is an optimization algorithm that is used to minimize a function by iteratively traveling in the direction of the steepest descent as defined by the gradient's negative. Gradient descent is used in machine learning to update the parameters of our model, especially weights in neural networks. Gradient descent is demonstrated in Figure 2.10.

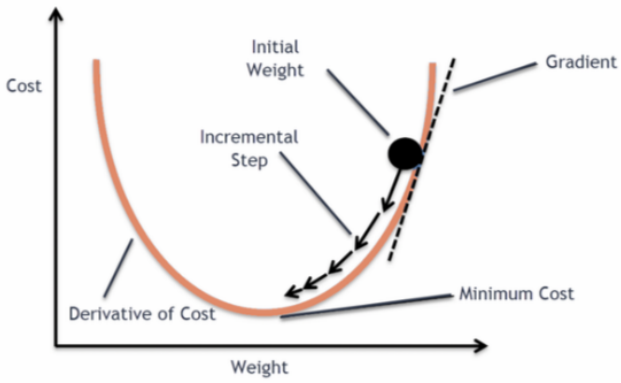


Figure 2.10: Gradient descent demonstration (source: https://www.analyticsvidhya.com/blog/2020/10/how-does-the-gradient-descent-algorithm-work-in-machine-learning/)

The objective of gradient descent is to minimize the cost function, or the error between predicted and actually, much like the purpose of determining the line of best fit in linear regression. It needs two data points to perform this: a direction and a learning rate. Future iterations' partial derivative calculations are determined by these variables, allowing the local or global minimum (i.e., point of convergence) to be reached gradually[13].

[13]IBM. *What is gradient descent?*.https://www.ibm.com/topics/gradient-descent

2.5.8 Learning Rate

The learning rate is the size of each step in each gradient descent cycle. We can cover more territory per step with a high learning rate, but we risk overshooting the lowest spot because the slope of the hill is continually changing. We may reliably go in the direction of the negative gradient with a very low learning rate because we are recalculating it so frequently. A low learning rate is more exact, but calculating the gradient takes time, so we will take a long time to reach the bottom. An example of the learning rate is in Figure 2.11.

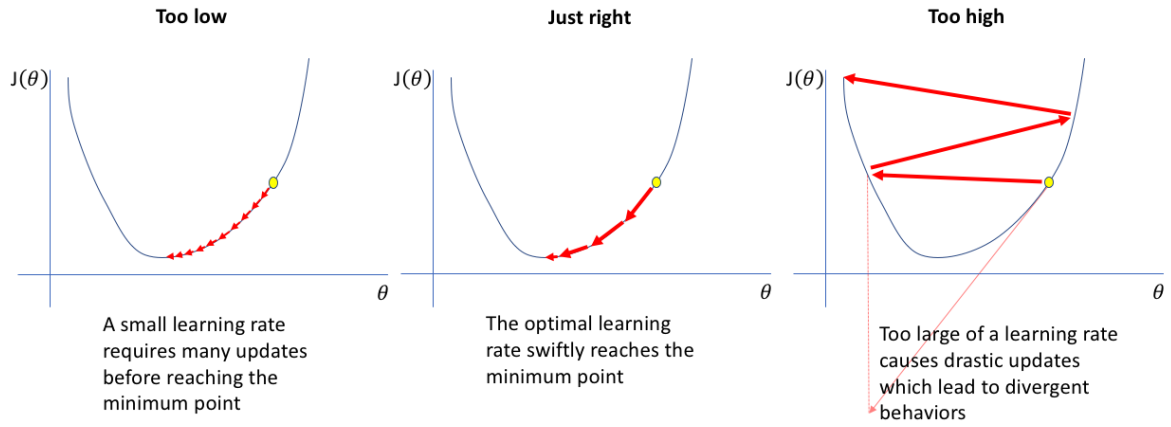


Figure 2.11: Learning rate (source: https://www.jeremyjordan.me/nn-learning-rate/)

How quickly the model adapts to the challenge is determined by the learning rate. Given the smaller changes to the weights made with each update, lower learning rates necessitate more training epochs, whereas higher learning rates produce quick changes and necessitate fewer training epochs.

When the learning rate is too high, the model may converge too rapidly to an unsatisfactory answer, whereas when it is too low, the process may become stuck.

Choosing the learning rate appropriately is a difficult aspect in training deep learning neural networks. For the model, it can be the most crucial hyperparameter[14].

[14]Jason Brownlee. *Understand the Impact of Learning Rate on Neural Network Performance*. Jan 2019. https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

2.5.9 Loss function

A loss function (or cost function) indicates how well the model predicts a given set of parameters. The loss function has its curve and gradients. The slope of this curve indicates how we should adjust our parameters to improve the model's accuracy. If the cost ever rises, we must reduce the value of the learning rate; if the cost falls slowly, we must increase the value of the learning rate. Figure 2.12 shows an example of loss function behavior based on the learning rate.

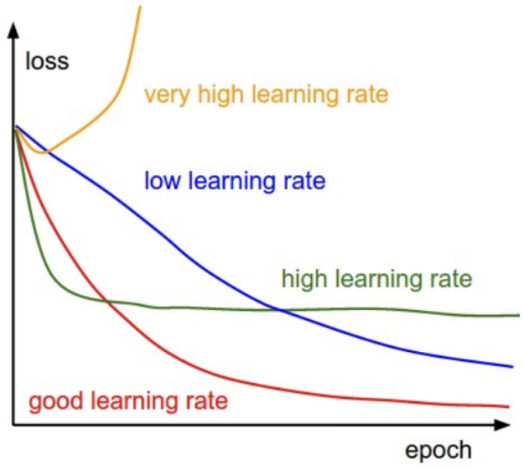


Figure 2.12: Loss function behaviour with different learning rate (source: https://www.researchgate.net/figure/Loss-function-is-changing-with-different-learning-rates-30_fig4_342657394)

2.5.10 Gradient descent optimizer

A method that computes adaptive learning rates for each parameter is *Adaptive Moment Estimation* (Adam). Adam, like Adadelta and RMSprop, preserves an exponentially decaying average of past squared gradients v_t in addition to an exponentially decaying average of past gradients m_t. Whereas momentum can be thought of as a ball rolling down a hill, Adam behaves more like a heavy ball with friction, preferring flat minima on the error surface. The decaying averages of past and past squared gradients, m_t and v_t, are computed as follows (2.11).

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{2.11}$$

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method. As m_t and v_t are initialized as vectors of 0's, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, especially when the decay rates are

small.

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \end{aligned} \tag{2.12}$$

They then use these to update the parameters.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{2.13}$$

3

Related works

This chapter mentions some related studies on the use of machine learning techniques to detect malicious requests, their strengths and weaknesses, and evaluate an appropriate approach for this thesis.

Table of Contents

3.1 Main approaches of machine learning to WAF

Malicious request detection has been a focus of recent studies, with several malicious request detection systems developed. They propose solutions for particular or different types of harmful requests. Existing malicious website detection approaches can be mainly divided into two categories based on their architecture: *A machine learning-based WAF* using pure machine learning to detect abnormal requests and *Using machine learning to improve existing WAF*.

3.2 Machine learning-based WAF

WAF based on machine learning is a type of web application firewall built on machine learning techniques to detect and prevent attacks on web applications. It uses machine learning algorithms to analyze data from incoming requests and responses from web applications, thereby creating rules to identify and prevent attacks. With continuous learning, the machine learning-based WAF can automatically update new rules to detect new and more advanced attacks. This makes it possible for WAF to protect web applications against unknown attacks and ensure the safety of user and business data. Here are some significant articles having an approach by using machine learning to create a WAF.

A. Shaheed et al. [1] present a proposed model for a web application firewall that uses machine learning and features engineering to detect common web attacks. Their proposed model uses Naive Bayes with cross-validation (100 Folds) analyzes incoming requests to the web server, parses these requests to extract four features that describe completely HTTP request parts (URL, payload, and headers), and classifies whether a request is normal or an anomaly. Final features are used by WAF to check if the request is normal or anomaly, these features are calculated and extracted based on the basic features. Features extracted in this model are general and can work with any web application. Final features describe all parts of the HTTP request including headers and files. Their proposed model achieved a classification accuracy of 99.6% with datasets used in research studies in this field and 98.8% with datasets of real web servers. The authors have performed tests on real data sets and the results show that the proposed method is capable of detecting web attacks with higher accuracy than conventional methods. However, the current implementation supports Linux distributions only.

I. Jemal et al. [2] build a smart web application firewall (SWAF) based on a convolutional neural network that learns from a massive amount of data and can detect malicious HTTP requests based on its learning phase and reduce the attack detection overhead time. They propose two solutions. The first one compresses the HTTP request. It keeps the necessary parts in the detection of attacks and eliminates the remaining parts which are useless in attack detection. The second one reduces the number of CNN neurons and eliminates the unnecessary one that increases the needed time. In this study, they used 5 fold cross validation to train and test the CNN model. They investigated a method to preprocess the HTTP requests called ASCII embedding, it proved experimentally that by using ASCII embedding, CNN model achieved better accuracy. It exceeds 98%, while, using word and character approaches, the accuracy did not overtake 97.6% and 96.12%, respectively. However, the ASCII embedding preprocessing method presents an extra analysis time (7.6ms) compared to the word and character embedding methods (3.85ms and

6.2ms, respectively). To solve this problem, they filtered CNN neurons and achieved high attack detection rate (98.1%) with overhead time does not exceed 2.3ms.

B. Dawadi et al. [3] make a comparative analysis between normal HTTP traffic and attack traffic that identifies attack-indicating parameters and features. This study introduces a layer architecture of WAF. Their proposed WAF consists of two modules in the layered architecture using Long short-term memory(LSTM), one for DDoS attack detection in the first layer and another for SQL injection and XSS detection in the second layer. The first detection layer was a DDoS attack detection model with an accuracy of 97.57%, and the second layer was for XSS and SQL injection attack detection with an accuracy of 89.34%. Rather than training the module with the single dataset, training the module with separate datasets would lead to better results, as the nature of the data and attacks are different. The result also proves that the analysis of features and parameters for attack detection helps to reduce false positives during traffic filtering, which is critical for the effectiveness of WAF.

S. Toprak et al. [4] focus on WAF systems with single and stacked LSTM layers architecture that reveal hyper-parameter values for the best performance using character sequences of user-supplied data. A semi-supervised approach is used and trained with PayloadAllTheThings dataset containing real attack payloads and only normal payloads of HTTP Dataset CSIC 2010 are used. F1-scores are used to calculate the success rate of the approach, which determines whether the user input is recognized as malicious or normal. High F1-scores and success in the detection and classification of the attacks were shown by the suggested model. To analyze character sequences of malicious web application payloads and implement various models with various hyper-parameters, the deep learning model with LSTM layers is analyzed and implemented throughout the work. This allows the authors to identify the best deep-learning structure for detection and to determine whether the supplied input words are benign or malicious. One may conclude that the resulting models, based on extensive experiments and analysis of various models with various hyper-parameters, are promising for the detection of malicious HTTP web requests for specific attack types using corresponding attack payloads with a relatively small attack payload dataset using a semi-supervised learning method. However, there is a genuine problem with the lack of a dataset that offers sample payloads for the relevant online assaults and web answers.

The goal of B. Gogoi et al. [5] is to assess the effectiveness of various machine learning (ML) methods in identifying XSS attacks in web apps and websites while employing Support Vector Machine (SVM) to detect XSS assaults. They developed a custom Apache webserver module that intercepts requests to the web application using hooks provided by Apache webserver. SVM has the advantage of being robust against overfitting problems, especially for text data due to high dimensional space. The authors also use 10-Fold cross-validation to prevent overfitting in SVM algorithm. The precision, recall and F1-score of both linear and nonlinear SVM indicates that the algorithms are able to successfully separate the XSS attack inputs from benign web application inputs. Though the non-linear SVM performs better in terms of accuracy and F1-score, it takes much longer time than the linear SVM for model fitting. The experimental results allow us to conclude that ways to detect XSS assaults have advantages over more conventional approaches. XSS attacks can be detected more accurately when ML technique is used in conjunction with conventional procedures.

D. T. A. M. Devi et al. [6] proposed a machine learning-based approach for a web application firewall. In this paper, one of the classification algorithms in machine learning is logistic regression used to classify good and bad queries. The queries were extracted and classified as legitimate and malicious using logistic regression classifier. All experiments were programmed and evaluated in Jupyter Notebooks, an interactive Python environment for data science. After converting data to an appropriate format using python scripts, they use the "bag-of-words" technique to convert data to numeric features. The N-Gram method is applied, and combined with TF-IDF vectorizer to turn the data into TF-IDF values to give n-grams weights, then apply logistic regression. The logistic regression classifier (where a 3-grams vectorizer) had 99.93% accuracy, a precision of 98.83% and F1-score of 99%. However, the dataset used to train and evaluate the proposed method is relatively small compared to the vast amount of web application content available on the internet, which may limit the generalizability of the proposed method.

3.3 Improving existing WAF by using machine learning

Combining machine learning and traditional methods in WAF can help improve the effectiveness of web application protection while minimizing errors and confusion in detecting and preventing web attacks. Conventional methods in WAF are believed to be correct. However, these methods may not be flexible enough and are difficult to adapt to new and advanced attacks. Therefore, combining traditional methods with machine learning enhances WAF's ability to detect and prevent attacks. Here are some significant articles having an approach by using machine learning to improve an existing WAF.

M. T. Nguyen et al. [7] aim to utilize machine learning methods to improve WAF performance. The authors discuss popular types of attacks on web applications and the survey of machine learning methods in the attack detection task to build an algorithm for automatic detection attacks based on the support vector machine (SVM) and analysis of HTTP requests. The scheme for classification process consists of 7 main stages: data collection to create a query base, preliminary data processing, payload comparison, checking regular expressions, calculation of request attributes, converting text data into vectors, and classification of queries based on the support vector machine. The researchers use 3-grams with cross-validation to verify the results of the approach. According to the experimental results, the proposed approach has both accuracy and F1-score at approximately 99.9%. The new approach is only effective for variable query length attacks (code injection, cross-site scripting) and requires high computational power as the dataset grows in size. The authors also suggest some methods to improve the performance of the model, such as using a combination of machine learning methods, increasing the number of quality attributes of queries or updating signature databases.

N. T. Tran et al. [8] build a WAF system based on ModSecurity with ModSecurity CRS, which focuses on lowering the false-positive rate of ModSecurity CRS based on machine learning techniques. This approach specifically blends ModSecurity CRS and machine learning. The authors suggested a strategy combining ModSecurity and ModSecurity CRS with Decision Tree and Random Forest machine learning models based on the created dataset and the accessible data. Based on the experimental results, they show that the proposed method has significantly improved the initial false-positive rate of the

ModSecurity CRS. However, in order to reduce the false-positive rate to this level (about 1.26%), the true-positive rate had to decrease to 69.96% (with Random Forest model). They also test their WAF system on real websites, also with Random Forest and only 0.29% of requests are wrongly blocked. With the trained model, the authors have been able to drastically lower the false-positive rate of ModSecurity CRS, making it more suited for the actual application. However, the proposed models do not increase the capacity to recognize any novel sorts of assaults; rather, they just reduce the false-positive rate in ModSecurity CRS's final decision formula.

The research of A. Alshammari et al. [9] suggests a detection architecture with several ML models for supplying IDS to find anomalous network traffic. This study proposes a system for detecting anomalies in network traffic by providing an IDS with an ML model. A dataset consisting of both malicious and legitimate traffic will be used. The data preprocessing phase significantly influences this work result. While others obtaining the accuracy of above 94%, with the D-tree, ANN, Random Forest and Decision Tree's result at 100%, the Naïve Bayes model and SVM is not suitable for this model with the accuracy of approximately 60%. Although the machine learning models utilized produced good results, the provided model had certain drawbacks. To extract the features of communication traffic and provide a real-time response, IDS security systems for computer networks must be extremely quick. The performance of fitting the system and evaluating it was affected by the large dataset that underlies the provided model and is regarded as a form of big data. In addition, the application of this approach in actual networks will slow down the necessary speed.

N. M. K. Le et al. [10] suggested an approach of extracting WAF rules and trained a machine learning with the decision model independent from the rules themselves. This makes the model more self-reliant and the overall result more neutral. The module has been tested, and though it is not completed yet, it shows potential. Eventhough this paper gains positive results, it also has some drawbacks. The dataset collected for phishing website detector is still small and skewed, the ML model is small and cannot cover all phishing cases. The malicious request validator has only been trained with three categories out of five and the authors only tested on one programming language in server-side script. This validator is still not field-tested on any production environment yet. Our module is inspired by this approach. We aim to finalize this module but combine it with another machine learning model to specialize it for the goal of determining incoming requests independent of the WAF rule.

4

Proposed approach

This chapter displays the problems as well as the malicious request validator's input and output. Then we offer the design and architecture of the problems' solutions.

Table of Contents

4.1 Cyber security problems

WAFs, as previously indicated, are frequently used, however, they suffer from high false positive rate. We aim to enhance the accuracy of WAFs, using the help of machine learning rather than rule-based approaches.

Because WAFs only cover the Application Layer, the network requests are the model's input (mostly HTTP requests). Our method produces the same result as WAFs: *whether the request is malicious or not.*

With its nature, WAF is obligated to have fast processing speed in deciding whether an incoming request is reliable or not. For user experiences, we can't examine the request's veracity for minutes before granting or denying access. Our system's time constraint must be in milliseconds.

4.2 Ratiocination

There several characteristics between the two strategies of machine learning in WAF. These distinctions are expected based on the approach of each direction.

WAF would be a deep learning program learned from a fixed data set or freshly added during operation under the supervision of machine learning-based WAF, however it will be limited to genuine machine learning. As a result, as compared to traditional WAF, machine learning-based WAF often achieves very high accuracy, typically exceeding 96%. [2], [4], [5], [6], and [9] attained accuracy greater than 99%. Furthermore, the response time of this direction's approaches is guaranteed to be very low, approximately 2.6ms at [2]. This could be explained by the fact that machine learning-based WAFs are often constructed on rather powerful hardware, and the data used to feed the machine learning models is preprocessed with much effort.

With researches on using machine learning to improve existing WAFs, it is common to aim to improve on one or a few existing WAF weaknesses. In [8], N.T. Tran et al. focus on reducing the false-positive rate of ModSecurity CRS based on machine learning methods by combining ModSecurity with a classification ML algorithm. [3] offers 2 separate layers of validation to ensure no malicious requests could "sneak" through the WAF, avoid DDOS and improve the precision for WAFs.

To summary, numerous strategies for identifying malicious URLs have been proposed. Most of these solutions utilize supervised-based machine learning techniques for classification. This technique can detect irregularities between requests, but it cannot extract these abnormalities into human-readable form in order to reconstruct the WAF. Relying solely on machine learning eventhough brought higher accuracy like in [11], WAFs need to be time-efficient. We can not compromise accuracy for speed, hence these methods don't work for WAFs.

Our approach is categorizing the incoming request and analyzing its structure, with moderate reliability, then combining the result of these two processes to achieve high precision yet not expend an excessive amount of time.

We have some theoretical points to prove why we decide to design in this way.

Firstly, in his 1996 paper, *The Lack of A Priori Distinctions Between Learning Algorithms*, D. Wolpert [12] introduced the No Free Lunch theorem for supervised machine learning and stated a quote at the beginning of his paper. The theorem states that given a noise-free dataset, "for any two machine learning algorithms A and B, the average performance of A and B will be the same across all possible problem instances drawn from a uniform probability distribution." This means "an algorithm may perform very well for one problem, but that gives us no reason to believe it will do just as well on a different problem where the same assumptions may not work."

Secondly, moreover, simpler models like logistic regression have more bias and tend to underfit, while more complex models like neural networks have more variance and tend to overfit.

- **Strengths and Weaknesses of logistic regression.** Let's take a look at some advantages provided by logistic regression. Firstly, it's easy to deploy, and it calculates accurately and fasts with big data. Secondly, it can handle both continuous and discrete data. Next, it's suitable for binary or multiclass classification problems. Finally, it allows us to determine the importance of input features. Besides the strengths, this model also has some weaknesses. First, this model cannot deal with data that is non-linear or has complex linear relationships. Second, it is sensitive to noise and outliers in data. Next, it can only be applied to classification problems, not to unsupervised learning algorithms. Lastly, it is prone to overfitting when the number of features is larger than the number of training samples.

- **Strengths and Weaknesses of CNN.** CNN model combined with word2vec is a widely used method in machine learning to perform natural language processing tasks such as text classification or machine translation. Below is a detailed analysis of the strengths and weaknesses of the CNN model combined with word2vec. Let's look at some of the CNN model's benefits. First, it can handle ordered data, including long data strings. Second, it can remember past information and apply it to future decisions. Next, it helps machine learning models understand the meaning of words and sentences in natural language. Another advantage, this model is capable of learning common features from texts, resulting in better prediction results with never-before-seen new text. Finally, CNN is faster by design compared to RNN - the more fitting model to categorize text. This machine learning model also has some weaknesses. The first one is that the CNN model requires a lot of computational resources to train due to many parameters. The following disadvantage is that sometimes it is not effective for problems with complex structured data such as images. Next, it is necessary to use appropriate dictionaries to represent words in vector space. Eventually, it is difficult for it to deal with words that are not in the user dictionary.

Finally, ensemble learning is an approach that combines diverse models to improve performance. The idea behind integrating different models is logical: each model has unique abilities, and when combined, they may excel in multiple tasks or subtasks. When these models are appropriately merged, they create a potent training algorithm capable of enhancing overall performance compared to using individual models alone. Ensemble learning reduces variance and bias, improving predictions of the developed model. Technically speaking, it helps avoid overfitting.

This problem classifies the input data as yes/no. It needs accuracy and the data set is not large, needs high accuracy. So we decided to combine logistic regression and CNN model.

4.3 Comparisons between ensemble learning and other machine learning approaches

To prove that ensemble learning is good enough for our project, we will make comparisons between ensemble learning and other machine learning approaches. First, let's discuss ensemble and rule-based learning. About the method, ensemble learning is a method of combining different machine learning models to create a better predictive model, while rule-based learning is a method of determining specific security rules based on information about known web attacks. About performance, ensemble learning can improve the performance of a machine learning model because it aggregates predictions from many different models. Meanwhile, new or unknown attacks can easily bypass rule-based learning. One of the most importance aspects is accuracy, ensemble learning can improve the accuracy of machine learning models by eliminating false predictions. However, the accuracy of the ensemble learning model also depends on the combination of submodels. Rule-based learning can achieve high accuracy if security rules are set up correctly. About generalizability, ensemble learning can improve the generalizability of a machine learning model by using a variety of models and avoiding overfitting. Rule-based learning may not generalize well if the security rules are not universal enough or easily circumvented. In summary, ensemble learning can improve the performance, accuracy, and generalizability of machine learning models for WAFs. Meanwhile, rule-based learning provides specific security rules and achieves high accuracy in some cases.

Another approach that comes into discuss is deep learning-based. Deep learning-based method has some advantages. First, deep learning uses complex models to analyze data, making it possible to learn and process data more accurately. Second, it is capable of handling and classifying thousands of attack types. Finally, it has the ability to automate the updating of security rules. Besides the strengths, it also has some limitations. The first limitation is that it requires large and varied training data to produce accurate results. Next, it requires powerful hardware to train and deploy the model. Lastly, it is difficult to explain how deep learning models work.

Thus, deep learning and ensemble learning both have their own advantages and disadvantages. However, because the special nature of WAF is often to protect against new attacks, ensemble learning may be the better choice due to its ability to detect new attacks based on a combination of models.

4.4 Designs

There are many efforts spent on this topic, as mentioned in **Chapter 3**. But there are two significant literatures about detecting malicious requests that inspired us.

M.Alsaedi et al.[11] proposed an approach to detect malicious URLs using ensemble learning. They extracted features using N-Gram and then use TF-IDF to represent them.

After that, the model applied Rain Forest Ensemble-Based for prediction and an artificial neural network (ANN) classifier was constructed for decision making. Results show that this approach significantly improved the detection performance, achieving 96.8% compared with the best 90.4% achieved by the URL-based features. The false-positive rate was significantly decreased to 3.1% compared with 12% performed by the URL-based model. This work is proven to be promising, but time is not the focus of this approach.

N. M. K. Le et al.[10] suggested an approach of extracting WAF rules and trained a machine learning with the decision model independent from the rules themselves. This makes the model more self-reliant and the overall result more neutral. The module has been tested, and though it is not completed yet, it shows potential. We aim to finalize this module but combine it with another machine learning model to specialize it for the goal of determining incoming requests independent of the WAF rule.

Our proposed approach is explained as follows.

It is already obvious that when attackers attempt to breach a system, they must try to execute or inject something executable onto the system. Malicious requests must therefore be script-based or written in computer languages. Most standard online applications and services, particularly API applications, appear to accept just plain text queries, i.e. requests in JSON or XML.

In the event of a website service that accepts scripted input, certain malicious requests may have the same category or language as the normal request; for example, the attacker may alter the SQL database account using SQL queries ('normal' database server category). However, the content of the malicious request seems to have a similar pattern. Moreover, to gain access to executing queries, attackers must try various methods to penetrate the server (reconnaissance scan, code injection, or command injection on the back-end servers and database servers). Then we only need to detect one malicious request to block all the attack session. Of course, WAF cannot protect systems from high-level methods, for example, attackers gain access by acquiring admin accounts from social engineering.

Our goal is to classify request supplied as inputs in order to determine whether they are harmful or inoffensive. The sample of request data consists of different categories including.

- **Plain text.** Request that contains data but does not trigger any machine execution, typically in the form of HTML, JSON, XML, and CSV, etc.
- **Client-side script.** Request in form of programming languages or scripts that can be executed on client's machine.
- **Server-side script.** Request in the form of programming languages (mostly back-end programming languages like PHP, JAVA, PYTHON, and so on) that can change the behavior of the application or web.
- **Shell script.** Request in the form of shell scripts that can run jobs on the server or change the server's or operating system's behavior.
- **SQL script.** Request that in form of SQL, can be used to query data from the database.

Then we introduce the first hypothesis: *normal requests to a server have the same category*. For example, a static web request may only contain plain text, API server requests are mostly in JSON format, incoming queries to a database are SQL or online compilers use programming language-format requests. A malicious request must be in different category with the normal requests, like a script request to a static web server or API server can be considered code injection or command injection, SQL requests to an API server may be SQLi, or script requests to a database is command injection or stored XSS attacks. We can determine if a request is 'normal' to a server by comparing the types of incoming suspicious requests to the average categories of regular requests. If the resemblance is low, we might presume that the origin of the inbound request is unusual. For instance, if a typical request to a secured server is in JSON format (plain text), but a suspicious request is in JavaScript (client-side script), we can deduce that the incoming request is malicious. If a suspicious request is in XML format (plain text), we may assume that the user made a mistake and the alarm was false.

The second hypothesis can be presented as *all regular requests will have a similar pattern, the same goes for malicious requests*. For example, a request with a URL ending in.exe or.php is frequently malicious, and a request with work like "SELECT", "DROP", or "TABLE" is unquestionably a SQLi. We can implement logistic regression to identify if an incoming request is normal or abnormal by analyzing its content.

From the hypothesises, we can create a CNN model to detect the type of request. Then we compare the incoming requests with the "normal" corresponding category and decide whether the requests is malicious or not, by using logistic regression.

4.5 Architecture

Our architecture (Figure 4.1) suggests a reasonable decision model for the combined result: *when two prediction is the same, the result is straight forward*. When the logistic

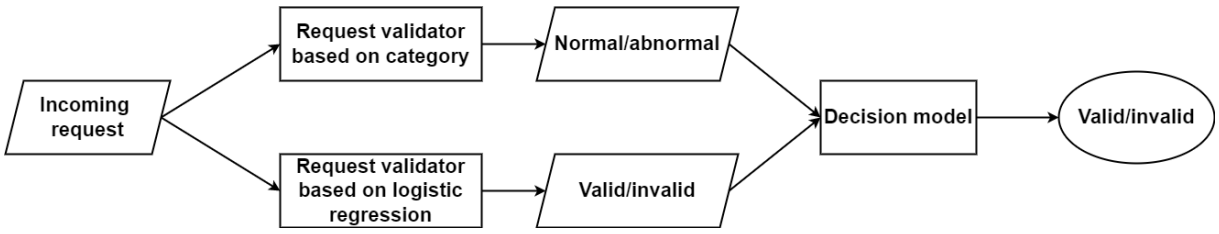


Figure 4.1: Malicious request validator architecture

regression model decided that the request is malicious, but the CNN model predicted the request is nornal, the CNN is favored. Otherwise, the Regression model classified the request as valid but the CNN predicted as abnormal, we'll favor the Regression. The decision model can be expressed in a decision table (Table 4.1).

Regression	CNN	Result
Valid	Normal	Valid
Valid	Abnormal	Valid
Invalid	Normal	Valid
Invalid	Abnormal	Invalid

Table 4.1: Decision table

The CNN validator will run for a set period of time (usually one or two weeks) to collect the familiar category of incoming requests and assign a threshold (which can be the mean or maximum (if we are optimistic) distance between each request vector in the observing stage and the sum vector). The same will also happened with the Regression model. After that training phase, the module will run on 'active phase', parallel with the WAF.

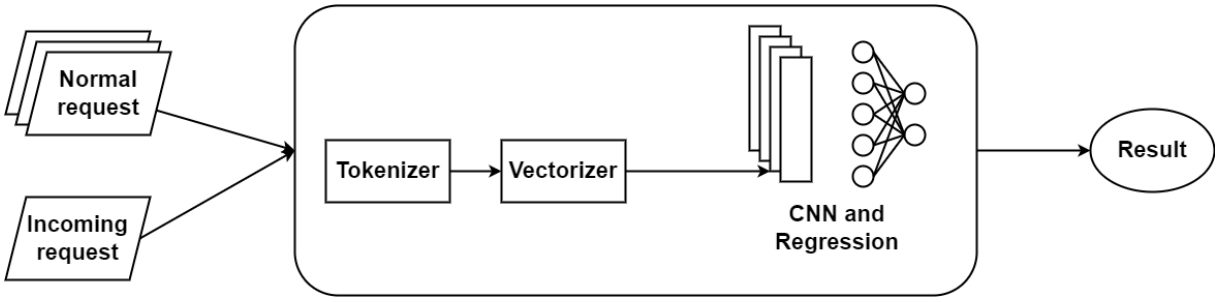


Figure 4.2: Decision model for the combination of CNN and the regression model

The request is routed through the module, which predicts the category. The category of suspicious request is then compared with the common category. Then the Regression model will determined whether the request is good or bad, combining with the normal or abnormal status to decide the result.

5

Implementation

This chapter describes the implementation steps of the systems including system frameworks.

Table of Contents

5.1 Framework

In this thesis, our module is built on Keras and Sklearn (python libraries). The framework is presented in the following figure (Figure 5.1).

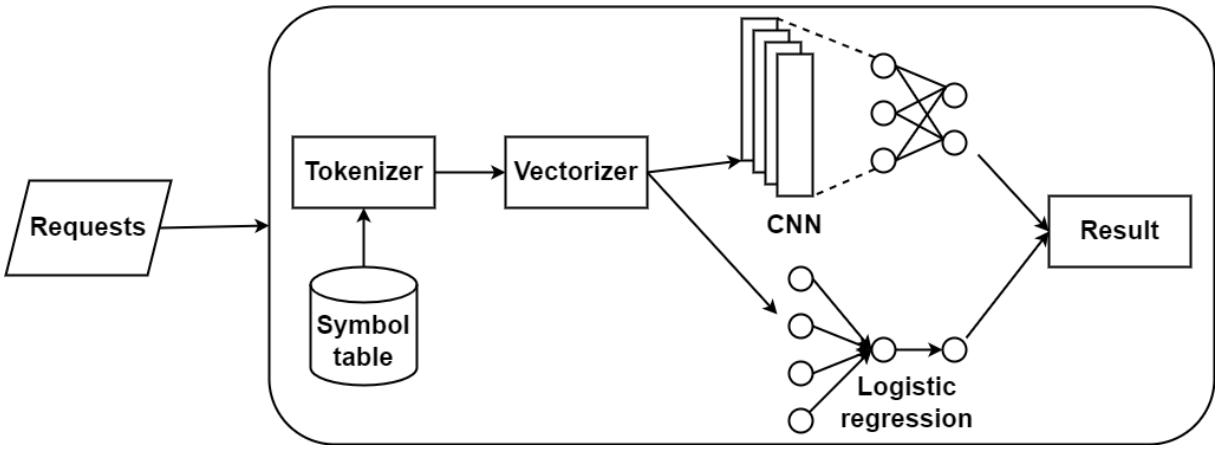


Figure 5.1: The framework of module

5.1.1 Logistic regression model

In the prediction process, we train a logistic regression model to determine whether an incoming request is malicious or not.

5.1.1.1 Data Processing

The data acquired, however, wasn't always in the format required for this project, i.e. just the input string. For instance, some of the data were in HTTP, GET, and POST queries. The data was converted into a format appropriate for this purpose to a CSV[1] file using Python scripts.

5.1.1.2 Feature engineering

The first part is tokenizing the data. Since our input do not only contain only words and punctuation like normal sentences, we must have an efficient way to tokenize our data (Figure 5.2).

[1]A comma-separated values (CSV) file is a delimited text file that uses a comma to separate values. Each line of the file is a data record.

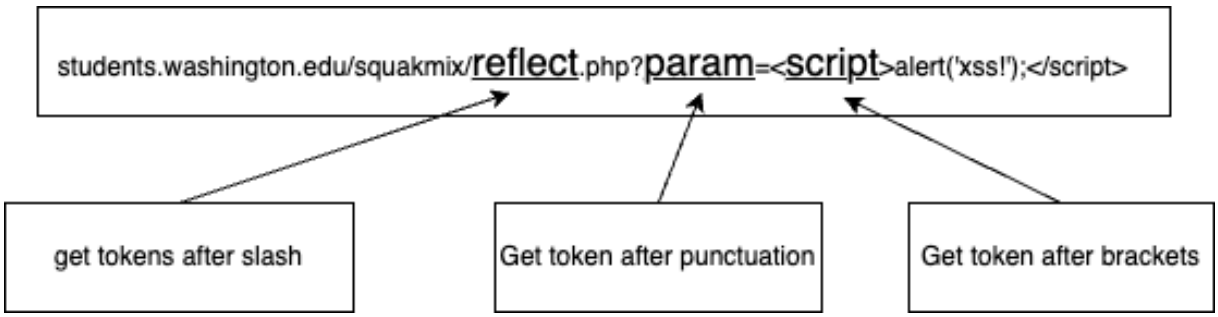


Figure 5.2: The method of tokenizing data

We also eliminate redundant tokens and remove frequently appearing words that are not relevant to the detection process of malicious requests, such as "localhost", ".com", etc.

Learning algorithms work with numerical features, so we need to convert words into numerical vectors. How does it work? The first entry technique mentioned is called "Bag-of-words."[2] A competent classifier in bag-of-words recognizes patterns in word distribution, which words appear, and how many times for each type of text. However, counting words is not always the best idea. As previously established, a request URL might be exceedingly long, adding bias to our predictions. We used TF-IDF scores instead of the "Bag-of-words" classification since there are words in URLs that are more important than other words e.g. 'DROP', 'script', 'param' etc. Then apply logistic regression.

5.2 CNN classifier

5.2.1 Framework

The framework of classifier is presented in Figure 5.3. The request classifier consists

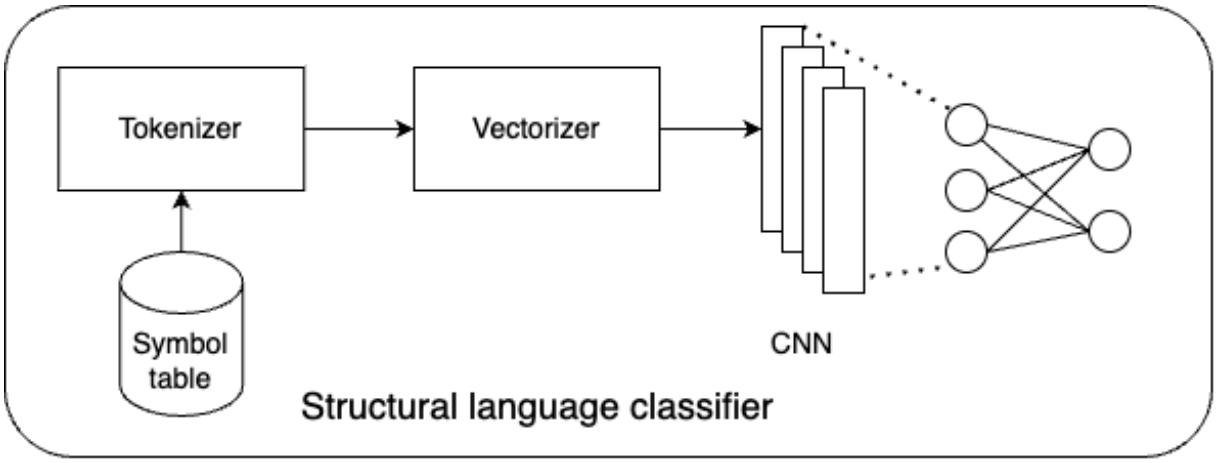


Figure 5.3: Framework of CNN classifier

of three parts: tokenizer, vectorizer, and CNN. The whole categorizer is built on Python.

The tokenizer formats the request by utilizing regular expressions to format white spaces and messages and a predetermined set of keywords (symbol table).

[2]Bag-of-words is a Natural Language Processing technique of text modelling.

The vectorizer vectorizes the request using a small word2vec model built by gensim[3]. The structured language dataset is used to train the model. The output of the vectorizer is a list of vectors (with height 150) representing each word of tokens.

The embedded vector is then sent into a trained CNN to forecast the request's categorization. The architecture is illustrated in Figure 5.4 and detailed as Table 5.1.

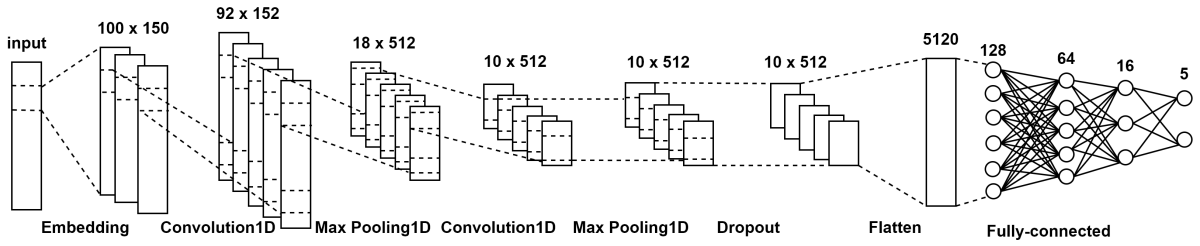


Figure 5.4: Architecture of malicious request validator model.

Layer (type)	Output Shape	Amount of parameters
embedding (Embedding)	(None, 100, 150)	172447200
conv1d (Conv1D)	(None, 92, 512)	691712
max_pooling1d (MaxPooling1D)	(None, 18, 512)	0
conv1d_1 (Conv1D)	(None, 10, 512)	2359808
max_pooling1d_1 (MaxPooling1D)	(None, 10, 512)	0
dropout (Dropout)	(None, 10, 512)	0
flatten (Flatten)	(None, 5120)	0
dense (Dense)	(None, 128)	655488
dense_1 (Dense)	(None, 64)	8256
dense_2 (Dense)	(None, 16)	1040
dense_3 (Dense)	(None, 5)	85
Total parameters: 176,163,589		
Trainable parameters: 3,716,389		
Non-trainable parameters: 172,447,200		

Table 5.1: Architecture specifiations of malicious request validator model.

The model can be explained as follows.

- **Input data**. The input data of the model is from the word2vec model. The data is then embedded using the previous vectorizer with height 150, making the output of the layer a 100×150 neuron matrix. The 3,716,319 parameters of this layer are the embeddings of the word2vec itself, then we have already trained these parameters in the previous vectorizer. This layer is frozen the CNN train starts with the next layer.

[3]Gensim is a Python library for topic modelling, document indexing and similarity retrieval with large corpora. Target audience is the natural language processing (NLP) and information retrieval (IR) community.

- **First 1D CNN layer**. The first layer defines a filter (also called feature detector) of height 9 (also called kernel size). Only defining one filter would allow the neural network to learn one single feature in the first layer. It might not be sufficient, so we will define 512 filters. It allows us to train 512 different features on the first layer of the network. The output of the first neural network layer is a 92×512 neuron matrix. Each column of the output matrix holds the weights of one single filter. With the defined kernel size and considering the length of the input matrix, each filter will contain 92 weights.

- **First Max pooling layer**. A pooling layer is often used after a CNN layer to reduce the complexity of the output and prevent overfitting of the data. In the thesis, we chose a size of five. It means that the size of the output matrix of this layer is only one-fifth of the input matrix.

- **Second 1D CNN layer**. Another sequence of 1D CNN layers follows to learn higher-level features. The output matrix after those two layers is an 18×512 matrix.

- **Second Max pooling layer**. One more pooling layer to further avoid overfitting. The output matrix has a size of 10×512 neurons. For each feature detector, ten weights remain in the neural network on this layer.

- **Dropout layer**. The dropout layer will randomly assign 0 weights to the neurons in the network. Since we chose a rate of 0.3, 30% of the neurons will receive a zero weight. With this operation, the network becomes less sensitive to react to smaller variations in the data. Therefore it should further increase our accuracy on unseen data. The output of this layer is still a 10×512 matrix of neurons.

- **Flatten layer**. This layer will compress the 10×512 matrix from the previous layer into one vector with height 5120, preparing for the classification phase.

- **Four fully-connected layers**. Four final layers will force the vector of height 5120 to a vector of 5 since we have five classes (plain text, client-side script, server-side script, shell script, and SQL). This reduction is made by matrix multiplications. Since the gap between the two vectors is too large (5120 and 5), we need five layers. The first three layers reduce the height from 5120 to 128, 128 to 64, and 64 to 16 respectively, using ReLU[4] as the activation function. The final layer reduces from size 16 to 5. Softmax is used as the activation function. It forces all three outputs of the neural network to sum up to one. The output value will therefore represent the probability for each of the three classes.

[4]Rectifier Linear Unit (ReLU) function: $g(z) = max(0, z)$

The CNN focuses on identifying typical requests (plain text) to decrease the FP of rule-based techniques. Then the model is converted to plain text using class weights. The weight for plain text is 5, whereas for others is 1. As a five-label loss function, the training method employs Mean squared error[5] loss function. Because others are programming languages, their predicted distributions are similar. When the loss function is Sparse Categorical Cross-Entropy[6], which has a harsh punishment for classification errors, the model may constantly alternate the prediction of these two labels. Adam is used as the model optimizer with learning rate of 0.005, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 1$. We are focused on the accuracy of the model.

[5]Mean Squared Error function: $MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y}_i)^2$

[6]Sparse categorical cross-entropy loss function: $-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$

6

Dataset

In this chapter, we want to describe the datasets which we use to train and evaluate the Logistic Regression module and CNN categorize module, with the problems of datasets and pre-processing details.

Table of Contents

6.1 Logistic regression dataset

6.1.1 CSIC 2010 dataset

The HTTP dataset CSIC 2010[1] contains web requests automatically generated, both normal and abnormal. It can be used for the testing of web attack protection systems. It was developed at the "Information Security Institute" of CSIC (Spanish Research National Council). The HTTP dataset CSIC 2010 contains the generated traffic targeted to an e-commerce web application developed at their department. In this web application, users can buy items using a shopping cart and register by providing some personal information. As it is a web application in Spanish, the data set contains some Latin characters. A request from the dataset is presented in the following figure (Figure 6.1).

```
GET http://localhost:8080/tienda1/publico/anadir.jsp?
id=2%2F&nombre=Jam%F3n+Ib%E9rico&precio=85&cantidad=49&B1=A%F1adir+al+carrito HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.8 (like Gecko)
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=F563B5262843F12ECAE41815ABDEEA54
Connection: close
```

Figure 6.1: A sample request of the dataset

The dataset consists of 36,000 legitimate requests and 25,063 anomalous requests. The HTTP requests are labeled as normal or abnormal; the dataset includes attacks such as SQL injection, buffer overflow, information gathering, file disclosure, CRLF injection, XSS, server-side includes injection, parameter tampering, and so on.

According to the authors, the dataset is generated using the following steps.

1. Real-life data are collected for all the parameters of the web application. All the data (names, surnames, addresses, etc.) are extracted from actual databases. These values are stored in two databases: one for the normal values and the other for the anomalous ones. Additionally, all the publicly available pages of the web application are listed.

2. For every web page, both normal and unusual requests are generated. When standard requests contain parameters, data randomly selected from the standard database is used to fill out the parameter values. The procedure is similar to anomalous requests in which the parameters' values are pulled from the anomalous database.

In this dataset, three types of anomalous requests were considered.

- Static attacks try to request hidden (or non-existent) resources. These requests include obsolete files, configuration files, default files, etc.
- Dynamic attacks modify valid request arguments: SQLi, CRLF injection, XSS, buffer overflow, etc.
- Unintentional illegal requests. These requests do not have malicious intentions, however, they do not follow the normal behavior of the web application and do not have

[1] Available at https://www.isi.csic.es/dataset/

the same structure as normal parameter values.

This dataset is quite outdated and randomly generated. It comprises a complete HTTP request with various components such as request URL, request method, pragma, and so on. During our research, we observed that the majority of the indicators of a malicious request may be located in the URL portion of the request. For example, an SQLi will have a route that includes a string such as: *2&nombre=Jam%F3n+Ib%E9rico&precio=85 &cantidad=%27%3B+ DROP +TABLE+usuarios%3B+SELECT+*+FROM+datos+ WHERE+nombre+LIKE+%27%25&B1=A%F1adir+al+carrito*.

Consequently, we will solely concentrate on the URL and Method parts of each HTTP request in order to detect fraudulent requests.

url	User-agent	Pragma	Cache-control	Accept	Accept-Encoding
GET http://localhost:8080	User-Agent: Mozilla/5.0 (co	Pragma: no-cache	Cache-control: no-cache	Accept: text/xml,applicatio	Accept-Encoding: x-gzip, x
POST http://localhost:808	User-Agent: Mozilla/5.0 (co	Pragma: no-cache	Cache-control: no-cache	Accept: text/xml,applicatio	Accept-Encoding: x-gzip, x
POST http://localhost:808	User-Agent: Mozilla/5.0 (co	Pragma: no-cache	Cache-control: no-cache	Accept: text/xml,applicatio	Accept-Encoding: x-gzip, x
POST http://localhost:808	User-Agent: Mozilla/5.0 (co	Pragma: no-cache	Cache-control: no-cache	Accept: text/xml,applicatio	Accept-Encoding: x-gzip, x
GET http://localhost:8080	User-Agent: Mozilla/5.0 (co	Pragma: no-cache	Cache-control: no-cache	Accept: text/xml,applicatio	Accept-Encoding: x-gzip, x
POST http://localhost:808	User-Agent: Mozilla/5.0 (co	Pragma: no-cache	Cache-control: no-cache	Accept: text/xml,applicatio	Accept-Encoding: x-gzip, x
GET http://localhost:8080	User-Agent: Mozilla/5.0 (co	Pragma: no-cache	Cache-control: no-cache	Accept: text/xml,applicatio	Accept-Encoding: x-gzip, x
POST http://localhost:808	User-Agent: Mozilla/5.0 (co	Pragma: no-cache	Cache-control: no-cache	Accept: text/xml,applicatio	Accept-Encoding: x-gzip, x
GET http://localhost:8080	User-Agent: Mozilla/5.0 (co	Pragma: no-cache	Cache-control: no-cache	Accept: text/xml,applicatio	Accept-Encoding: x-gzip, x
POST http://localhost:808	User-Agent: Mozilla/5.0 (co	Pragma: no-cache	Cache-control: no-cache	Accept: text/xml,applicatio	Accept-Encoding: x-gzip, x

Accept-Charset	Accept-Language	Host	Cookie	Connection
Accept-Charset: utf-8," utf-	Accept-Language: en	Host: localhost:8080	Cookie: JSESSIONID=8176	Connection: close
Accept-Charset: utf-8," utf-	Accept-Language: en	Host: localhost:8080	Cookie: JSESSIONID=9331	Content-Type: application/
Accept-Charset: utf-8," utf-	Accept-Language: en	Host: localhost:8080	Cookie: JSESSIONID=7104	Content-Type: application/
Accept-Charset: utf-8," utf-	Accept-Language: en	Host: localhost:8080	Cookie: JSESSIONID=417I	Content-Type: application/
Accept-Charset: utf-8," utf-	Accept-Language: en	Host: localhost:8080	Cookie: JSESSIONID=A651	Connection: close
Accept-Charset: utf-8," utf-	Accept-Language: en	Host: localhost:8080	Cookie: JSESSIONID=EE6/	Content-Type: application/
Accept-Charset: utf-8," utf-	Accept-Language: en	Host: localhost:8080	Cookie: JSESSIONID=5355	Connection: close
Accept-Charset: utf-8," utf-	Accept-Language: en	Host: localhost:8080	Cookie: JSESSIONID=5834	Content-Type: application/
Accept-Charset: utf-8," utf-	Accept-Language: en	Host: localhost:8080	Cookie: JSESSIONID=346/	Connection: close
Accept-Charset: utf-8," utf-	Accept-Language: en	Host: localhost:8080	Cookie: JSESSIONID=DF3I	Content-Type: application/

Figure 6.2: Requests dataset

6.1.2 Malicious and non-malicious URL

We opted to utilize this extra dataset from Kaggle, with an extremely basic structure, because using only data from the CSIC 2010 set is not dependable enough for our model to have a high level of reliability. Include just URLs and their labels. It consists of over 400,000 unique already labeled URLs.

url	label
diaryofagameaddict.com	bad
espdesign.com.au	bad
iamagameaddict.com	bad
kalantzis.net	bad
slightlyoffcenter.net	bad
toddscarwash.com	bad
tubemoviez.com	bad
ipl.hk	bad
crackspider.us/toolbar/install.php?pack=exe	bad
pos-kupang.com/	bad
rupor.info	bad
svision-online.de/mgfi/administrator/components/com_babackup/classes/fx29id1.txt	bad

Figure 6.3: A set of labeled URLs

Another thing to mention is the requests datasets generally are confidential, as they may contain sensitive data of users. There are not many open datasets available. Also, labeling these datasets are costly task, as it requires expertise in the field.

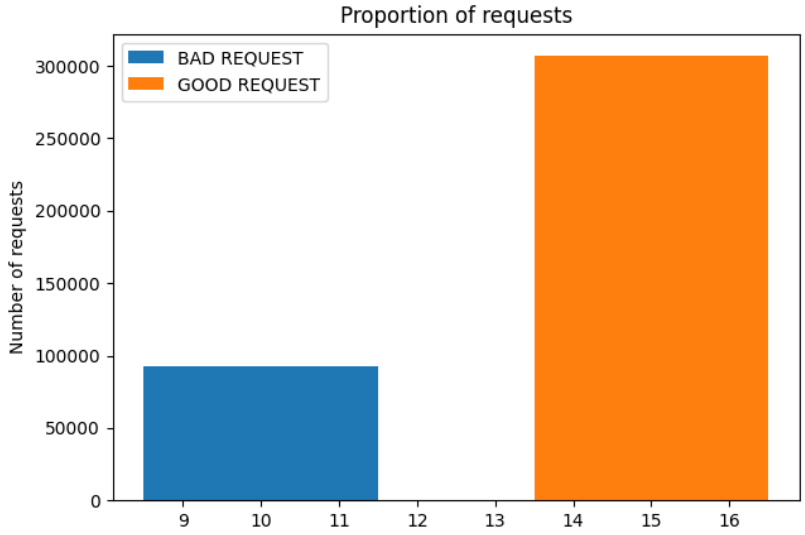


Figure 6.4: Proportion of requests

6.2 CNN classification module

The malicious request validator module detects the category of requests. As the categories are mostly structured languages (four out of five categories), we need to get a dataset of structured language and its category, as proposed in the proposed approach.

This dataset contains over 97,000,000 snippets of code from various GitHub repositories with more than 10,000 stars. The repositories included in this dataset were the results of searching for repositories with greater than 10,000 stars. For each repository, we created snippets from the default branch by going through each text file and extracting 5-line chunks of text every 5 lines. The dataset used file extensions to associate snippets with the programming language they most likely represent. For snippets for which it could not infer the language from the file extension, it uses the value UNKNOWN in the language column.

This dataset is very large, stored in an SQLite database, and includes many languages, so data preprocessing is very critical. We use a Python script to extract randomly 400,000 snippets of code from the database, and only from the languages relevant to our classification in the proposed approach.

The list consist of these categories.

- Plain text: JSON, HTML, YAML
- Client-side script: JavaScript
- Server-side script: PhP, Go, Python, C, C++
- Shell script: Shell
- SQL script: SQL

The shorten dataset is presented in Figure 6.5.

62610991	actualAttribute); assertNotEquals('Expected to find attribute with name id, in element ' + errorSuffix, '', actualAttribute.value); assertEquals(	JavaScript
62075630	<a href="ir__ <a href= <a href="ir__T <a href="c <a href="ir__Te	HTML

97257225	auto *returned = result.claimNext();
	if (ABIType != returned->getType())
	returned = coerceValue(returned, ABIType, IGM.DataLayout);
	
	Builder.CreateRet(returned);
		C++

78365817	enumerable: true,
	get: function () {
	return _index.objectTypeCallProperty;
	}
	});
		JavaScript

6126932	], "layout": { "width": 1200, "height": 800, "annotations": [	JSON

Figure 6.5: The processed dataset

The quantity of each programming language is shown in Figure 6.6.

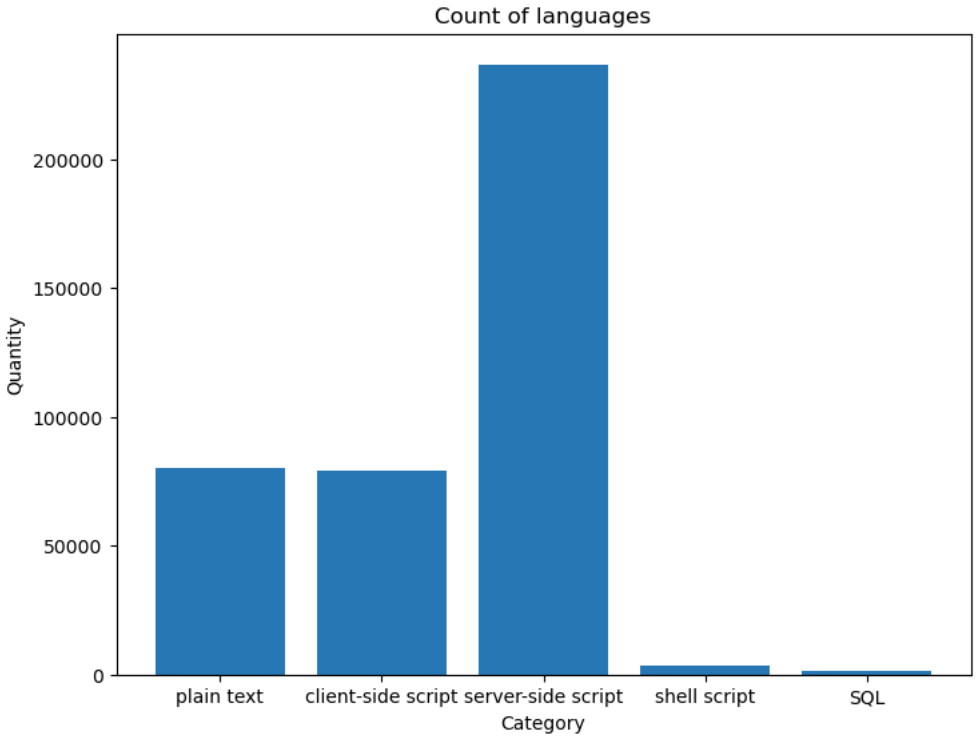


Figure 6.6: The quantity of every category in the dataset

7

Experiments

This chapter discusses the division of the dataset before training, assessment techniques, and experimental results. Compare the outcomes of the experiments we suggest to the reference experiments from relevant works.

Table of Contents

7.1 Classification evaluation metrics

The effectiveness of a statistical or machine learning model is assessed using evaluation metrics. These comprise classification accuracy, logarithmic loss, confusion matrix, and others.

It is necessary to analyze the model using a variety of measures. When utilizing one measurement from one assessment metric, a model may perform well, but when using another measurement from another evaluation metric, it may perform poorly. To make sure the model is working properly and ideally, evaluation metrics are essential.

7.1.1 Dataset preparation

Splitting is a technique for evaluating the performance of a machine learning algorithm. Train-test split and train-validation-test split are two common types of split.

In the train-test split technique, a dataset is split into two subsets. The training dataset is the first subset, used to fit the model. Instead of using the second subset to train the model, the input element of the dataset is given to the model, and predictions are then made and compared to the expected values. The testing dataset is the second dataset.

- **Training dataset.** It is used to fit the machine learning model.
- **Testing dataset.** It is used to evaluate the fit machine learning model.

The goal is to gauge how well the machine learning model performs on new data, i.e., data not used to train the model. We intend to put the model to use in this manner. In other words, to fit it to the data that is currently available with inputs and outputs that are known, and then to forecast outcomes for new cases in the future where we do not yet have the target values or expected outputs.

The validation dataset is a set that is occasionally necessary. It is known as the train-validation-test split approach. The dataset is divided into three subsets: training dataset, validation dataset, and testing dataset.

- **Training data.** The sample of data used to fit the model.
- **Validation dataset.** The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.
- **Testing data.** The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

7.1.2 Confusion matrix

The confusion matrix (error matrix), a tabular depiction of the model predictions versus the ground-truth labels, is one of the fundamental ideas in classification performance. The examples in a predicted class are represented in each row of the confusion matrix, while

the occurrences in an actual class are represented in each column. A confusion matrix is displayed in Table 7.1.

		Actual class	
		TRUE	**FALSE**
Predicted class	**TRUE**	True Positive (TP)	False Positive (FP)
	FALSE	False Negative (FN)	True Negative (TN)

Table 7.1: Confusion matrix

As we can see diagonal elements of this matrix denote the correct prediction for different classes, while the off-diagonal elements denote the samples which are misclassified. The confusion matrix can be normalized to obtain the rates. The normalized confusion matrix is presented in Table 7.2. Instead of TP, FN, FP, TN, normalized confusion matrix uses True Positive Rate (TPR), False Negative Rate (FNR), False Positive Rate (FPR), True Negative Rate (TNR). The most important figures here are FPR (also called false alarm rate) and FNR (miss detection rate).

		Actual class	
		TRUE	**FALSE**
Predicted class	**TRUE**	TPR = TP/(TP+FN)	FPR = FP/(FP+TN)
	FALSE	FNR = FN/(TP+FN)	TNR = TN/(FP+TN)

Table 7.2: Normalized confusion matrix

7.1.3 Classification accuracy

Classification accuracy is one of the most straightforward metrics one can think of. It is calculated by dividing the proportion of accurate predictions by all possible predictions and multiplying the result by 100. The equation of classification accuracy is.

$$accuracy = 100 \left(\frac{TP + TN}{TP + TN + FN + FP} \right)$$

7.1.4 Precision

Classification accuracy sometimes fails to serve as an accurate reflection of model performance. One of these cases is when there is an imbalance in the distribution of the classes, with one class being more prevalent than the others. In this scenario, even if every sample is expected to belong to the most common class, we would still have a high accuracy rate, which is completely illogical given that the model is not learning anything and is simply forecasting every sample to belong to the top class. Therefore we need to look at class-specific performance metrics too, such as precision, which is defined as.

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

7.1.5 Recall

Recall is another important metric, which is defined as the fraction of samples from a class which are correctly predicted by the model. It is shown in following equation.

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

7.1.6 F1-score

Recall or precision could be given a higher emphasis depending on the application. However, there are a lot of applications where both recall and precision are crucial. Therefore, it seems sensible to consider how to integrate these two metrics into a single one. The F1-score, which is the harmonic mean of precision and recall, is one well-known statistic that combines precision and recall.

$$F1\text{-}score = \frac{2 \times precision \times recall}{precision + recall}$$

The generalized version of the F1-score is.

$$F_\beta = \left(1 + \beta^2\right) \frac{2 \times precision \times recall}{\beta^2 \times precision + recall}$$

as we can see F1-score is a special case of F_β when $\beta = 1$.

7.2 Machine learning models evaluation

7.2.1 Logistic regression validator

7.2.1.1 Criteria and dataset preparation

This module is one of the two validators for the WAF, so not missing any suspicious requests is our top priority. However, this is only a part of our module, built with the criteria of fast and light, easy to run on many different configurations, so the accuracy may not be as high as other approaches. False positive is another major problem.

7.2.1.2 Experimental results

The experimental results of logistic regression are presented in Figure 7.1.

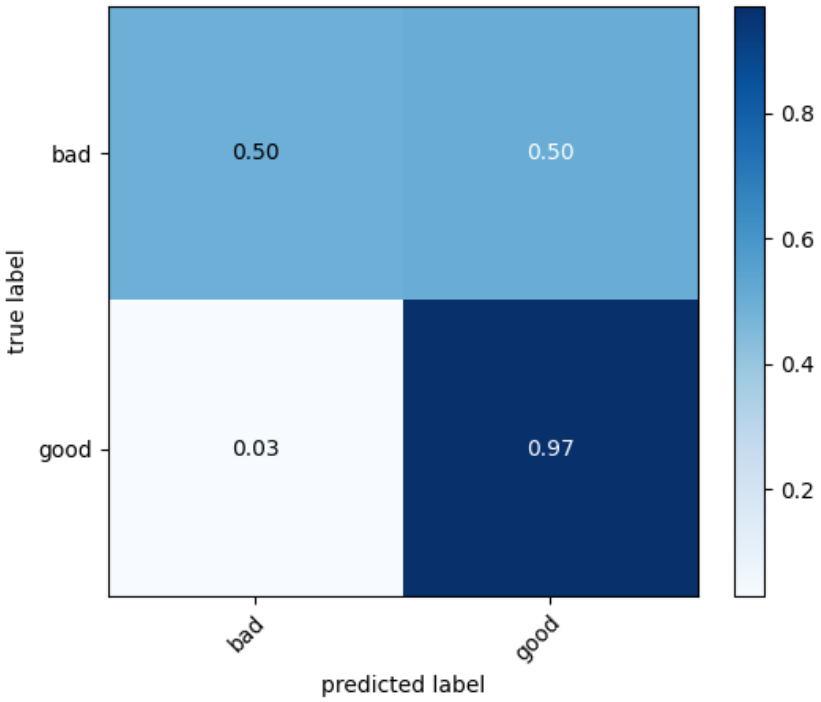


Figure 7.1: Confusion matrix of logistic regression model

The result of the logistic regression model is presented below.

- **Train accuracy:** 88.56%
- **Test accuracy:** 86,89%

	Precision	Recall	F1-score	Support
bad	**0.52**	0.86	0.65	1430
good	**0.97**	0.87	0.92	8568
accuracy			0.87	9998
macro avg	0.75	0.87	0.78	9998
weighted avg	0.91	0.87	0.88	9998

Table 7.3: Logistic regression model evaluation

The model has a very good chance of detecting normal requests with a precision of 97%, however, performs poorly in identifying malicious requests. Recall is not advised as a metric for evaluating malicious request detection since FP is more critical. This problem is not alarming, as we will discuss further in this thesis.

7.2.2 CNN Classifier

7.2.2.1 Criteria and dataset preparation

This module is a classifier for the incoming requests to the module. Then based on the proposed approach in **Chapter 4**, the validator will decide whether the request is normal or not. The model must have a low false alarm rate and no miss detection rate. It means the model must achieve almost zero FPR, and zero FNR, which leads to over 95% precision, especially in the category *plaintext*. Another vital criterion is time efficiency. A WAF must respond to block or not to block a request immediately. Therefore the time processing of each request in the model must be less than 10 milliseconds.

7.2.2.2 Experimental results

The CNN model is trained in 10 epoch with batch size 128. The precision metric and loss of the model during training process is plotted as Figure 7.2.

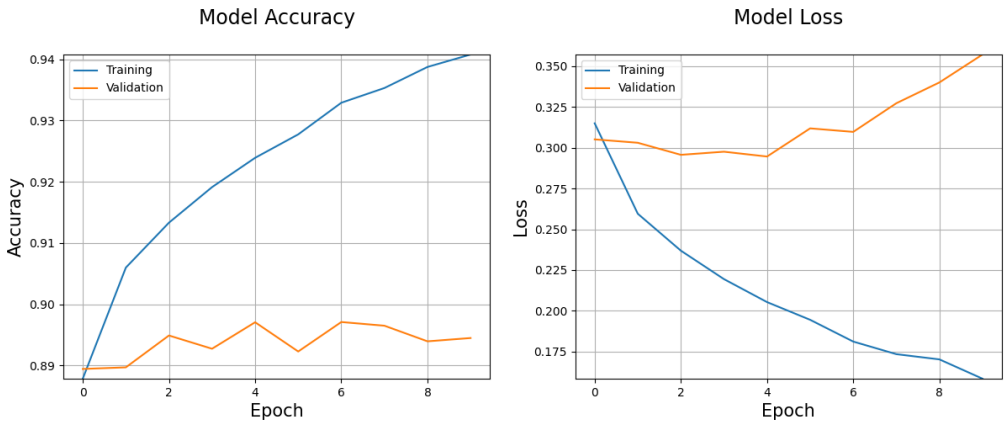


Figure 7.2: The precision metric and loss of the model

It appears that the model's training set and validation set differ fairly noticeably from one another. The loss of the training dataset decreases continuously, while the loss of the validation set decreases at early steps and starts increasing after a certain point. This demonstrates how "overfitting" our model was. The validation loss began below the training loss, then slightly decreased and increased after the fourth epoch. This leads to the stagnation of validation accuracy when the *val_accuracy* does not seem to increase. The confusion matrix is displayed in Figure 7.3.

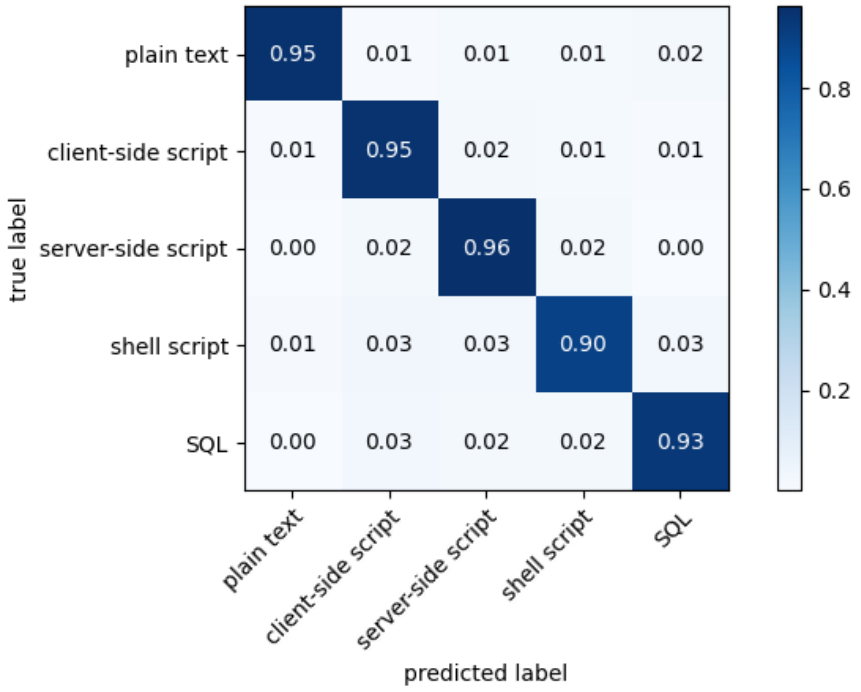


Figure 7.3: Structural language classifier model trained

While the values for other categories are only from 90% to 95% correspondingly, it appears that the model has a fair possibility of recognizing regular requests (plain text) with pretty good accuracy and a tiny chance of false alert or miss detection. As the following four categories are all programming languages, it is difficult for the model to separate them, and their weights are lower than plain text (one-fifth). Then the model has succeeded in reducing the blocking of legitimate requests i.e. reducing the false positive rate of WAF.

The evaluation metric of the model is presented in Table 7.4.

Label	Accuracy	Precision	Recall	F1-score
Plain text	**98.56%**	**0.95**	0.99	0.97
Client-side script	97.91%	0.95	0.95	0.95
Server-side script	97.29%	0.96	0.99	0.97
Shell script	98.4%	0.9	0.36	0.52
SQL	99.34%	0.93	0.63	0.75

Table 7.4: Evaluating the model by precision, recall and F1-score.

It is noticeable that the model does not have the best accuracy (only 95.75% on the test set). However, because there are just two convolutional layers in the model, it processes requests on average in 29ms (with system specifications shown in Table 7.5), meaning that the response is returned nowhere instantly. That time is still acceptable despite not reaching our goal yet (since our model requires the combination of two different models to produce the desired result).

Hardware	Specification
CPU	Apple M1 (8 cores)
GPU	Apple M1 8-core GPU
RAM	16GB

Table 7.5: System specifications.

7.3 End-to-end experiments

In these experiments, we are considering the default structural language category is plain text, showing that our module is defending an API application (which accepts just plain text queries, i.e. requests in JSON or XML).

We are using CSIC 2010 dataset (**Chapter 6**) to test our model and compare our work with the related works that also train on the mentioned datasets.

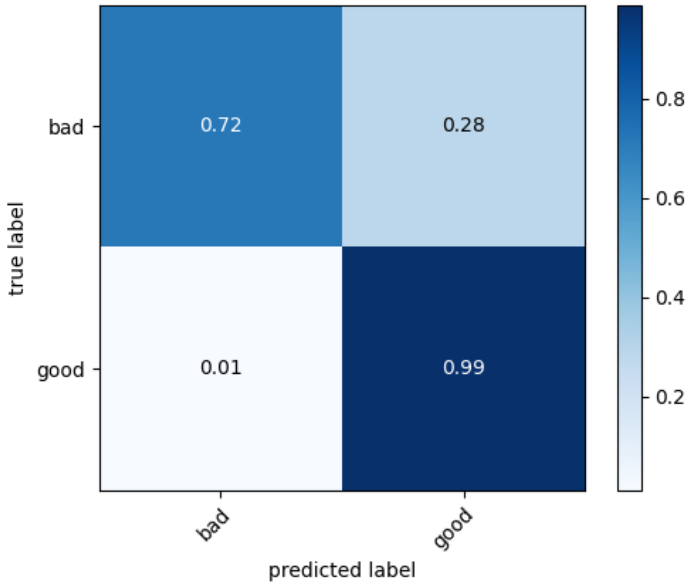


Figure 7.4: Confusion matrix of our WAF

	Precision	1-Precision	Recall	1-Recall	F1-score
Bad	0.7157	0.2843	0.9955	0.0045	0.8327
Good	0.9966	0.0112	0.7686	0.2314	0.8679
Accuracy			0.8523		
Misclassification rate			0.1477		
Macro F1			0.8503		
Weighted F1			0.8549		

Table 7.6: Evaluation of our WAF.

Through these experiments, we can see that the WAF model performs excellently in correctly detecting normal requests with a 99.66% chance. This implies that it is extremely uncommon for our WAF to unintentionally intercept a legitimate request. On the other hand, the WAF's performance on identifying imalicious request was found to be suboptimal, with precision values of 0.7157. This seems to be a problem. But attack detection should not be emphasized because when a hacker targets a web site, he must test out a wide range of attack techniques, from simple to complicated, and send a large amount of malicious request in a short amount of time. As a result, this issue is not concerning. Only around 80% of incoming attacks need to be detected for a model to alert administrators to take action to prevent further attacks. The extremely high precision in predicting normal request also means that we have succeed in reducing the FP rate in Traditional WAF. Therefore, the WAF have passed the FP rate criteria.

	CSIC	**Custom dataset**
Our machine learning-based WAF	85.23%	To be tested
Our machine learning-assisted WAF	66.98%	To be tested
ModSecurity	51.46%	44.21%
A. Shaheed et al.[1] (2022)	99.59%	98.8%
I. Jemal et al. [2] (2022)	98.1%	To be tested
S. Toprak et al. [4] (2022)	99.03%	To be tested
D. T. A. M. Devi et al. [6] (2022)	99.93%	To be tested

Table 7.7: Accuracy of our proposed model compared with related works.

Even though our WAF is still far from perfect, it has perfected the core elements of a WAF, such as accepting requests as input and returning whether or not the requests were denied.

When our machine learning-based WAF is compared to traditional WAF, in this case, ModSecurity, it is clear that our WAF performs significantly better, with 85.23% accuracy compared to 51.46% of ModSecurity, on the same dataset. We also attempt to apply our machine learning models to ModSecurity to determine whether a machine learning-aided WAF may outperform a WAF under the same circumstances. And in this instance, it is still insufficient to be compared to our WAF, thereby showing that, under the same conditions, machine learning-based WAF is better than the other two approaches. However, it is still not enough for our WAF to be considered superior to other studies. It still has weaknesses that can be overcome in the future. In reality, if you look at it from the standpoint of an application, 99.66% of blocking the request is sufficient and even considered average. Because if a model has a precision in predicting a legitimate request of 99%, that means out of every 100 legitimate requests that pass through the WAF system on average (just counting different requests), one of them will be mistakenly labeled as attacks. If the aforementioned WAF system is used to defend a customer website and even one request is unintentionally blocked, it may render the user useless. It will be even worse if customers are prevented from logging in or making transactions on e-commerce websites. The accuracy required to forecast the "normal" class must therefore be extremely high (above 99.9%), but the accuracy required to predict the "attack" class must be at a tolerable level (about 80%).

The weakness of this WAF is that it takes from 40ms to 140ms to process a request, depending on the type of request. This is an unacceptable time for WAF. WAF must be able to handle thousands of requests per second, otherwise, it will greatly affect the user experience. So, in the latency criteria, our WAF has failed to pass.

8

Conclusion

This chapter summarizes the results for the thesis. Finally, we want to present future improvements.

Table of Contents

8.1 Result

After completing the thesis, we have had time to research and learn more about NLP and other machine learning models, as well as security applications. We have benefited greatly from understanding engineering in machine learning and provided an unconventional method in integrating two deep learning models to identify attack requests during the execution of the topic *Building a web application firewall using machine learning techniques*. We have managed to construct a malicious request validator that can validate incoming requests. Our machine learning-based WAF succeeded in reducing the false positive rate by less than 1%. However, the response time of 40ms to 140ms per request is still not satisfactory for practical WAF. And based on the comparison of our machine learning-based WAF with others' works in the field, we can conclude that depending on the requirements, a machine learning-based WAF or machine learning assist WAF can be applied.

8.2 Limitations and future improvements

8.2.1 Limitations

Our thesis faces difficulties and has many drawbacks.

- The dataset destined for this thesis became inaccessible midway through. So we have to improvise and redesign the whole thesis. This led to the dataset currently used for the module being small and outdated, along with being impractical. This can be improved by further data cleaning and processing. Also, data labeling and preprocessing is a gruesome task. If we had had more reliable data, the machine learning model could have been trained more accurately.
- The malicious request validator is yet to be put to the test in a real-world production environment. But only by implementing the module, we could conclude that our proposed approach is not an effective one.

8.2.2 Further improvements

For each of the modules separately, there are rooms and potential for them to be further improved. About the logistic regression model, more data can be collected by constructing a proxy server. This proxy server will then take on the role of a forward proxy, transferring all HTTP request data from the user to it at this time and also collecting real request data for the model. We can extract more features, and the model could be built to be a phishing website URLs detector.

The structural language categorization module, however, has the potential to be applied in other domains. We can train data lacking categories (SQL script) and shell script, as well as crawl additional data. The CNN model would be able to categorize the languages independently rather than only categorizing them into the five suggested categories. It may then be implemented as a programming languages detection addon for VSCode in a real-world setting. To track the languages and create a continuous detector for programming

languages, we may attempt extracting the model and combining it with an RNN model (for example, LSTM).

Bibliography

[1] A. Shaheed and M.-B. Kurdy, "Web application firewall using machine learning and features engineering," *Security and Communication Networks*, vol. 2022, Jun. 2022. DOI: 10.1155/2022/5280158.

[2] I. Jemal *et al.*, "Swaf: A smart web application firewall based on convolutional neural network," Nov. 2022, pp. 01–06. DOI: 10.1109/SIN56466.2022.9970545.

[3] B. Dawadi, B. Adhikari, and D. Srivastava, "Deep learning technique-enabled web application firewall for the detection of web attacks," *Sensors*, vol. 23, Feb. 2023. DOI: 10.3390/s23042073.

[4] S. Toprak and A. Yavuz, "Web application firewall based on anomaly detection using deep learning," *Acta Infologica*, Oct. 2022. DOI: 10.26650/acin.1039042.

[5] B. Gogoi, T. Ahmed, and H. Saikia, "Detection of xss attacks in web applications: A machine learning approach," *International Journal of Innovative Research in Computer Science & Technology*, vol. 9, pp. 1–10, Jan. 2021. DOI: 10.21276/ijircst.2021.9.1.1.

[6] D. T. A. M. Devi and B. A. Kumar, "Machine learning with logistic regression for web application firewall," *INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) ICCIDT - 2022*, vol. 10, 04 Aug. 2022, ISSN: 2278-0181. DOI: 10.17577/IJERTCONV10IS04059.

[7] M. T. Nguyen, P. H. Truong, and T. N. Hoang, "A new approach to improving web application firewall performance based on support vector machine method with analysis of http request," *Journal of Science and Technology on Information security*, vol. 1, pp. 62–73, Jun. 2022. DOI: 10.54654/isj.v1i15.842.

[8] N. T. Tran *et al.*, "Improving modsecurity waf with machine learning methods," in *Future Data and Security Engineering. Big Data, Security and Privacy, Smart City and Industry 4.0 Applications - 7th International Conference, FDSE*, (Quy Nhon, Viet Nam), ser. Communications in Computer and Information Science, vol. 1306, Springer, Nov. 2020, pp. 93–107. DOI: 10.1007/978-981-33-4370-2_7.

[9] A. Alshammari and A. Aldribi, "Apply machine learning techniques to detect malicious network traffic in cloud computing," *Journal of Big Data*, vol. 8, Jun. 2021. DOI: 10.1186/s40537-021-00475-1.

[10] N. M. K. Le *et al.*, "Machine learning approaches to cyber threats detection," UNDERGRADUATE THESIS, Ho Chi Minh City University of Technology, 2021.

[11] M. Alsaedi *et al.*, "Cyber threat intelligence-based malicious url detection model using ensemble learning," *Sensors*, vol. 22, no. 9, 2022, ISSN: 1424-8220. DOI: 10.3390/s22093373.

[12] D. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural Computation*, vol. 8, Mar. 1996. DOI: 10.1162/neco.1996.8.7.1341.

List of keywords

The following list lists the keywords mentioned in this thesis in alphabetical order.