Detection of SQL injection using Reinforcement Learning

BACHELOR OF COMPUTER ENGINEERING

by

Tejas Sheth, 19102026

Nidhi Singh, 19102042

Janhavi Anap, 19102043

Het Patel, 19102005

Guide

Prof. Ramya R B

Department of Computer Engineering

A. P. SHAH INSTITUTE OF TECHNOLOGY, THANE

2022-2023

A. P. SHAH INSTITUTE OF TECHNOLOGY, THANE

CERTIFICATE

This is to certify that the project entitled “***Detection of SQL injection using Reinforcement Learning***” is a bonafide work of **“Tejas Sheth, (19102026), Nidhi Singh (19102042), Janhavi Anap (19102043), Het Patel(19102005)”** submitted to the University of Mumbai in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Computer Engineering**

Guide

Prof. Ramya R B

Project Coordinator

Prof. Rushikesh Nikam

Head of Department

Prof. Sachin Malave

Principal

Dr. Uttam Kolekar

Date:

A. P. SHAH INSTITUTE OF TECHNOLOGY, THANE

Project Report Approval for B.E.

This project report for Sem-VII entitled ***Detection of SQL injection using Reinforcement Learning*** by **“Tejas Sheth, (19102026), Nidhi Singh (19102042), Janhavi Anap (19102043), Het Patel(19102005)”** is approved for the degree of ***Bachelor of Engineering*** in ***Computer Engineering***, ***2022-23***.

Examiner Name Signature

1. ______________________________

2. ______________________________

Date:

Place:

Declaration

We declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principlesof academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and canalso evoke penal action from the sources which have thus not been properly citedor from whom proper permission has not been taken when needed.

(Tejas Sheth 19102026)

(Nidhi Singh 19102042)

(Janhavi Anap 19102043)

(Het Patel 19102005)

Date:

Abstract

SQL Injection continues to be one of the most noxious security exploits widely used by hackers all over the world, featured in the OWASP Top 10 report for 2022 [23]. The vulnerability can be unintentional by software developers during the development phase, an intentional ploy employed by a hacker to exploit or corrupt personal sensitive data or for reasons unknown. Amongst all the database types, relational databases are very popular, eg. MySQL, MS SQL, and Postgres. The flexibility of SQL makes it a very powerful language, allowing users to fetch any available information without having sufficient knowledge of databases. The vast databases subject the hacker's attention, potentially risking critical confidential information.

In this project, we plan to analyze the outcomes of various machine learning algorithms on a dataset consisting of potential SQL Injection queries. We propose a Machine Learning solution to minimize the threat posed by SQL Injection using various supervised, and unsupervised algorithms and plan to implement a Reinforcement Learning algorithm to detect a SQL attack beforehand and prevent any potential data loss or theft.

Keywords: ***SQL injection SQL detection SQL prevention SQLIA Information security Q-learning Reinforcement learning Vulnerability detection Machine learning Capture the flag.***

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Abbreviation

SQLi	Structured query language injection
CIA	Confidentiality, Integrity , Availability
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Network

CHAPTER 1

Introduction

SQL is a Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database. SQL is the standard language for Relational Database Systems. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language. SQL is widely popular because it offers the following advantages −

- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows embedding within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create views, stored procedures, and functions in a database.
- Allows users to set permissions on tables, procedures and views.

Due to its wide popularity, it makes SQL more susceptible to attacks. SQL is designed to allow people access to information and is therefore inherently vulnerable. SQL is agnostic, meaning it works across database platforms. The upside to this is that it allows code to be database-server agnostic. But it is also the source of the problem. To prevent most vulnerabilities, developers should use parameterized SQL or stored procedures specific to the database server.

One of the big reasons why SQL injection maintains traction is due to improper development planning and the use of insecure development architecture. Making use of unsupported or legacy software or features introduces security holes that may not be patched or caught as quickly as they would with modern software. Running patched and modern versions of software are critical to avoiding security exploits, including SQL injection.

SQL injection is a web security vulnerability that allows an attacker to interfere with queries that an application makes to its database. This attack occurs at the application layer. Through successful SQL injection an attacker can view, modify, delete the data and, also get access to sensitive data. This leads to breach of the three security principles, CIA, i.e. confidentiality, integrity and authenticity of data. The main consequences are:

Confidentiality: Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL Injection vulnerabilities.

Authentication: If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.

Authorization: If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of a SQL Injection vulnerability.

Integrity: Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL Injection attack.

This attack occurs when you ask for user input, like username, and the user incorrectly fills it with some SQL statement that gets unknowingly executed on one’s Database. SQL Injection is very common with PHP and ASP applications due to the prevalence of older functional interfaces. Because they are relatively easy to implement, and because the potential reward is great, SQL injection attacks are not uncommon.

Statistics vary, but it’s estimated that SQL injection attacks comprise the majority of attacks on software applications. According to the OWASP (Open Web Application Security Project**)**, injection attacks, which include SQL injections, were the third most serious web application security risk in 2021 [23].

1.1 SQL Injection

There has been an exponential rise in the number of dynamic websites. We have to handle large amounts of data stored for varied purposes, as well as their modification has to be fast to provide a rich user experience. For this, we have to rely on relational databases like MySQL, MSSQL, etc. which are based on standard query language SQL.

SQL communication between the website and the SQL server consists of SQL queries. The most common and frequently used operation is data retrieval using the SELECT command with the WHERE clause, an advancement can be the use of multiple SQL queries concatenated with a UNION statement returning just one table.

SELECT column1, column2 FROM table1 WHERE column1=column2;

SELECT column1 FROM table1 WHERE column1=10 UNION SELECT column2 FROM table2 WHERE column2=column1;

A hacker manages SQL Injection whenever the server-side scripting has an inappropriate input validation, leading to the attacker gaining partial or full access to the query and thus the database.

SELECT column1 FROM table WHERE column2=user-input;

SELECT column1 FROM table WHERE column1=1 OR 1 = 1;

This was the most simple SQL Injection, more refined queries are those when the attacker adds a UNION statement and combines multiple queries from the original query. There are many types of conventional and modern SQL Injection attacks. They are primarily divided into

A. Classical SQLi (Basic SQLi) attacks are the simplest and most frequently used form of SQLi. These may occur when users are permitted to submit a SQL statement to a SQL database through user input. There are 4 different types: Piggy-Backed, Stored Procedures, Union Query, and Alternative Encoding according to Halfond, W. G. et. al. [18] and Wei, K. et. al. [19].

1) Piggy-Backed Queries: Instead of modifying the original query, the attacker intends to develop new queries that piggyback on it. As a result, the DBMS gets inundated with SQL queries. The first query is a standard query that is run normally, while the succeeding ones are run to complete the attack.

   e.g., normal SQL statement + ";" + INSERT (or UPDATE, DELETE, DROP) <rest of injected query>

2) Stored Procedure: When a conventional SQL query (such as SELECT) is generated as a stored procedure, an attacker can inject another stored procedure as a substitute resulting in a denial of service (DOS), or execute remote instructions.

   e.g., normal SQL statement + "; SHUTDOWN; " <rest of injected query>

3) Union Query: An attacker injects a UNION SELECT query to mislead the programme into

providing data from a table other than the one intended as stated by Hwang, D. (2022) [20]. *e.g., normal SQL statement + "semicolon" + UNION SELECT <rest of injected query>*

4) Alternative Encoding: The attacker modifies the SQLi pattern such that standard detection and prevention systems miss it. In this method, the attacker employs hexadecimal, Unicode, octal, and ASCII code encoding in the SQL Statement

B. Advanced SQLi

1) Blind SQLi: Attackers devised strategies to circumvent the lack of error notifications while still knowing whether the input is being treated as a SQL statement. This technique is often used in two variations: content-based blind SQLi and time-based blind SQLi.

2) Fast Flux SQLi : Fast Flux is a DNS method to conceal phishing and malware distribution sites behind a constantly changing network of compromised servers. The Asprox botnet was used to launch the large SQLi assault employing rapid flux. In Fast Flux mode, the DNS (Domain Name Server) hosts many malware-infected IPs at the same time, and the IPs rapidly change.

3) Compounded SQLi: A compound SQLi attack is a pair of two or more attacks that target the webpage and have far-reaching implications than the previous SQLi mentioned. The fast development of detection and mitigation measures for multiple SQLis has resulted in the emergence of compound SQLi. SQLi combined with DDoS attacks, DNS hijacking, XSS, and insufficient authentication are just a few examples.

CHAPTER 2

Literature Survey

SQL Injection (SQLi) is a type of attack in which an attacker inserts a malicious SQL query into the web application by appending it to the input parameters. Sadeghian, A. et. al. illustrate the classification of injection attacks like tautologies, illegal / logically incorrect queries, union queries, piggy-backed queries, stored procedures, inference, and alternate encodings [15]. Security researchers have categorized the solutions for SQLi into three main groups: Best code practices, SQLi detection and SQLi runtime prevention. The optimum solution would be writing secure code and among best code practices- parameterized querying is the most secure and efficient technique.

Rai, A. et. al. illustrate the classification and prevention of different SQLi attacks. SQLi is generally classified as In-band SQLi, Inferential SQLi and Out of Bound SQLi [16]. In-Bound SQL injection is further classified as Error-based and Union-based SQLi. Inferential SQLi can be broken down into Boolean-based Blind SQL and Time-based SQL. Defensive techniques that could be used to prevent an SQLi attack include Whitelisting/Blacklisting, prepared statement/ parameterized query, stored procedure, defensive coding practice, taint-based approach, proxy filters, instruction set randomization, low privileges and output Escaping. Different countermeasures work for different SQL Attacks.

Medhane and M. H. A. S. based their approach on SQLi grammar to identify the SQLi vulnerabilities during software development and SQLi attack based on web-based applications [17].

The attacker's area unit used SQL queries for assaultive and hence these attacks reshare the SQL queries, thus neutering the behavior of the program.

John, A. proposed methods consisting of the best features of parse tree validation and code conversion techniques [4]. The algorithm parses the user input and checks whether it's vulnerable if any chance of vulnerability is found it applies code conversion over that input. Results show few drawbacks of code conversion as applying it to every user input is more time consuming and as well as the database also increases. The parse tree validation technique could raise a false alarm if a legitimate user is having blank space in his/her input. The proposed method proved to provide higher security levels than the individual techniques of code conversion and parse tree validation.

Hanmanthu, B. et. al. illustrates the use of the famous decision tree classification techniques to prevent SQLi attacks [5]. The proposed model works by sending different specially planned attack requests to the proposed SQLi decision tree model, and the final SQLi database is created for using classification data. It uses the satisfied analysis technique for finding the SQLi attack and uses the SQL decision tree. Software engineers usually rely on dynamic query building with string concatenation to construct SQL statements. The proposed method makes it possible to engineer different queries based on varying conditions set by users, without the need for manual interactions or error-prone code. The model showed consistency in attack detection and elimination at an average of 82% for all types of attacks. In order to perform a comparative evaluation of the proposed model, authors in [5] compared the proposed model to the other SQL scanning model which includes Acuneits, Netsparker, and Web cruiser and the results of the proposed model show good accuracy in comparison to other models.

Akinsola Jide et.al. gives us an idea about different types of SQLi attacks as already mentioned by Rai, A. et. al. [6][16]. The three main types are Classic In-band SQLI, Inferential Blind SQLI, and SQLi Based On Out-of-Band. They present the comparative analysis of different supervised ML algorithms to mitigate SQLi attacks. Besides precise accuracy and minimum errors, ML models also require putting several factors into consideration. The following metrics were taken into consideration to decide the effectiveness of the algorithm: Kappa Statistic, True Positive (TP) Rate, Accuracy, True Negative (TN) and (time to build the model (TTB)), for each of the machine learning algorithms.

Tang, P. et.al. only extracted and classified the URL features [7]. The factors like payload length, keywords and their weights are considered for feature extraction. The URL is classified as malicious or non-malicious using ANN (Artificial Neural Network) models. The method and algorithm used here are multi-layer perceptron (MLP) and LSTM, both of which were implemented using Pytorch. The trained model is deployed in the ISP system so that abnormal behaviours can be found in the network in real-time. One of the drawbacks of using such an approach is that using the LSTM, model recognition is poor with high processing time & has lower accuracy.

Four machine learning models were considered in Kamtuo, K., & Soomlek, C. and they were compared, Support Vector Machine (SVM), Boosted Decision Tree, Artificial Neural Network, and Decision Tree [8]. They have proposed a framework using a compiler platform and ML to detect SQLi in queries which are illegal and logically incorrect on server-side scripting. The dataset consists of 1100 samples of vulnerable SQL commands. After training the model with the dataset it was evaluated in terms of probability of detection, probability of false alarm, precision, accuracy, and processing time. Decision Jungle was the best in performance showing results as the best machine learning model which related to the processing time of 2.4725 seconds and accuracy of 0.9968.

Ross, K. collected traffic from two points: a web application host and a Datiphy appliance node [9]. It is demonstrated that the accuracy obtained with correlated datasets using algorithms such as rule-based and decision-tree are nearly the same as those with a neural network algorithm, albeit with significantly improved performance.

Reinforcement Learning (RL) is known for obtaining knowledge by trial and error and continuously interacting with a dynamic environment. It is characterized by self-improving and online learning, making it one of the intelligent agents (IA) core technologies. The reinforcement signal provided by the environment in RL is to make a kind of appraisal of the action quality of the IA, but not tell the IA how to generate the correct action. The basic model of RL as stated in Qiang, W., & Zhongli, Z. includes a state, action and reward system [10]. Where the IA perceives the environment and chooses an action to obtain the biggest reward value by continuously interacting with the environment. The ultimate goal of RL is to learn an action strategy. The basic theory of reinforcement learning technology is: If a certain system's action causes a positive reward for the environment, the system generating this action lately will strengthen the trend, this is a positive feedback process; otherwise, the system generating this action will diminish this trend. Typical RL method based on the Markov decision-making process (MDP) model includes two kinds: Model-based methods such as the SARSA algorithm and Model-irrelevant methods, such as the TD algorithm and the Q-learning algorithm.

Tian, W. et. al. illustrates methods to generate more effective penetration test case inputs to detect

SQLi vulnerability [11]. The model-based penetration test method is found to generate test cases covering more types and patterns of SQLi attack input to thoroughly test the 'blacklist filter mechanism' of web applications. Here, the authors proposed two-step penetration test case generation, building and instantiating, where step 1 reveals what test case should be used while step 2 expounds on how many test cases should be used. This study focuses on the adequacy of penetration test case inputs for the SQL injection vulnerability. It builds an experimental platform to verify the proposed test case generation methods.

Ghanem M. C., & Chen T. M. proposes and evaluates an AI-based pentesting system which makes use of RL to learn and reproduce average and complex pentesting activities [12]. The scope is limited to network infrastructures PT planning and not the entire practice. Moreover, the authors tackle the complex problem of expertise capturing by allowing the learning module to store and reuse PT policies in a more efficient way.

Niculae, S. et al. measured the performance of multiple fixed-strategy and learning-based agents [13]. They concluded that Q-learning, with some extra techniques applied and greedy agent initialisation, performed best, surpassing human performance in the given environment.

Hu, Z., Beuran, R., & Tan, Y. suggests an automated penetration testing framework, based on deep learning techniques, particularly deep Q-learning networks (DQN) [14]. The authors conducted an experiment in which a given network host was populated with real host and vulnerable data, to determine the optimal attack path, and to provide viable solutions.

Erdődi, L. et. al. simplified the dynamics of SQLi vulnerabilities by casting the problem as a security capture-the-flag and implementing it as an RL problem [1]. Assuming that the vulnerability has been identified, they rely on RL algorithms to automate the process of exploiting SQLi. They implemented the model using two simulations. The first simulation showed that a simple RL agent based on a Q-learning algorithm can successfully develop an effective strategy to solve the SQLi problem. A tabular Q-learning algorithm can discover a meaningful strategy by pure trial and error and can reach a performance close to the theoretical optimum. Using a table to store the Q-value function allowed them to carry out a close analysis of the learning dynamics of the agent, but this approach had poor scalability. Thus, in the second simulation, they sacrificed interpretability in order to work around the issue of scalability. They deployed a deep Q-learning agent to tackle the same problem as in the first simulation. The deep Q-learning agents were able to learn a good strategy for the SQLi problem as well as provide a solution to the space constraints imposed by the instantiation of an explicit Q-table.

Given the success of RL in tackling and solving games, Penetration Testing, when distilled as a capture-the-flag (CTF), can be expressed as a game. However, in the case of penetration testing, an artificial agent may learn only by trial and error while a human hacker may rely on alternative sources of knowledge, deductions, hypothesis testing, and social engineering. Although an RL agent may in principle learn the structure from scratch in a pure model-free way, this may turn out to be a computationally hard challenge. Thus according to Zennaro, F. M., & Erdodi, L.injecting some form of elementary a priori knowledge about the structure of the problem may simplify the learning problem [2]. Some basic forms of apriori knowledge which make the RL agent more efficient are lazy loading, state aggregation and imitation learning. The authors categorized CTFs in groups according to the type of vulnerability they instantiate and the type of exploitation that a player is expected to perform. The prototypical classes of CTF problems considered were port scanning and intrusion, server hacking and website hacking. All the simulations were implemented using the standard RL interface defined in the OpenAI gym library. Simulation 1 solved the Port Scanning CTF problem using the basic tabular Q-learning algorithm. Solving this challenge required learning the problem meaning that the RL agent has to rely strongly on exploration. Simulation 2 solved the Non-stationary Port Scanning CTF problem by extending the previous problem by considering a more challenging scenario in which the target system is not stationary, but it may randomly change in response to the actions of the agent. Introducing non-stationary dynamics made the problem more challenging by preventing the agent from learning the exact structure of the problem with certainty. Despite this, the Q-learning agent was still able to solve the CTF problem in a reasonable yet sub-optimal way. Simulation 3 solved the Server Hacking CTF problem with Lazy Loading which considers a more realistic scenario. The problem presented a serious challenge to the tabular Q-learning agent because of the size of its Q-table. Relying on a priori knowledge in the form of lazy loading controlled the dimensionality of the state and action state pruning the non-relevant states. This method allowed the agent to discriminate between relevant and non-relevant states based on its experience. Simulation 4 solved the Website Hacking CTF problem with State Aggregation preserving most of the complexity of Simulation 3. State aggregation allowed them to inject useful prior information about the structure of the problem, thus simplifying exploration and reducing the number of (state, action) pairs. Simulation 5 solved the Web Hacking CTF problem with Imitation Learning which emulates learning in a teacher-and-student setting, where expert paradigmatic behaviors are offered to a student to speed up its learning. Imitation learning proved to be an effective technique to enable faster learning for the RL agent. The improvement was due to the possibility of introducing the agent's knowledge of the structure of the problem. Instead of encoding knowledge of the structure of the problem in a formal mathematical way, they provided the RL agent with concrete observations about the structure of the problem. The agent could successfully exploit this information in order to learn an optimal policy.

Verme, M. D. et. al. considered the problem of exploiting SQLi vulnerabilities, representing it as a capture-the-flag scenario in which an attacker can submit strings to an input form with the aim of obtaining a flag token representing private information [3]. The attacker was modeled as an RL agent that interacts with the server to learn an optimal policy leading to an exploit. The authors did a comparison between two types of agents, one was a simple structured agent that relied on significant a priori knowledge and used high-level actions and the other was a structureless agent that had limited a priori knowledge and generated SQL statements. The comparison showcased the feasibility of developing agents that relied on less ad-hoc modeling.

Author	Title	Source	Findings
Erdődi, L., Sommervoll, Å. Å., & Zennaro, F. M. (2021)	Simulating SQL Injection Vulnerability Exploitation Using Q-Learning Reinforcement Learning Agents[1]	https://arxiv.org/pdf/2101.03118.pdf	Assumed that the injection was detected and automate the process of exploiting SQLi using tabular Q-learning and deep Q-learning of the Reinforcement Learning
Zennaro, F. M., & Erdodi, L. (2020)	Modelling Penetration Testing with Reinforcement Learning Using Capture-the-Flag Challenges[2]	https://arxiv.org/pdf/2005.12632.pdf	It uses similar method as the above paper but also provides some a priori knowledge to the model to improve results. It also explored situations like Port scanning, Server Hacking, and Web Hacking
Verme, M. D., Sommervoll, Å. Å., Erdődi, L., Totaro, S., & Zennaro, F. M. (2021, November)	SQL Injections and Reinforcement Learning: An Empirical Evaluation of the Role of Action Structure[3]	https://link.springer.com/chapter/10.1007/978-3-030-91625-1_6	This paper did comparison between the performance of simple structured agents and agents which were provided with a priori knowledge.

Table 2.1: Survey of Literature

CHAPTER 3

Limitation of Existing system

The SQLi detection is done using the various models available on the internet. The available datasets consist of older injection queries and as a result our project seems to be outdated to current injection queries

The project is able to detect majorly the tautology queries and some complex queries.

The injection on variety of databases cannot be predicted as for now due to limitation of the training dataset

CHAPTER 4

Problem Statement, Objectives and Scope

4.1 Problem Statement

SQL injection is the most common web application vulnerability. The vulnerability can be generated unintentionally by software developers during the development phase. So far existing methods have considered dealing with SQLi by identifying and classifying the vulnerabilities and relied on supervised machine learning algorithms or unsupervised machine learning algorithms. The problem of performing SQL injection can be considered a challenge that can be tackled with reinforcement learning methods. Thus , the reinforcement learning agents can help to identify the SQLi attack and prevent it from being executed.

4.2 Objectives

- Create a basic Web Application to perform SQLi attacks.
- Detection and Prevention of SQLi Attacks using Reinforcement Machine Learning Agents.

4.3 Scope

So far existing methods have considered dealing with SQLi by identifying and classifying the vulnerabilities and relied on supervised or unsupervised machine learning algorithms. The currently available datasets work effectively on tautology queries, but lack the means to detect the compounded queries. The problem of performing SQL injection can be considered a challenge that can be tackled with reinforcement learning methods. The reinforcement learning agents can help to identify the SQLi attack and prevent it from being executed, thus improving the security of the website by detecting the injection before executing the query in the backend.

CHAPTER 5

Proposed System

5.1 Proposed system overview

SQL Injection continues to be one of the most noxious security exploits widely used by hackers all over the world. The vulnerability can be unintentional by software developers during the development phase, an intentional ploy employed by a hacker to exploit or corrupt personal sensitive data or for unknown reasons. The exploitation leads to the breach of CIA principle i.e., Confidentiality, Integrity and Availability.

In this project, we aim to analyze the outcomes of various machine learning algorithms on a dataset consisting of potential SQL Injection queries. We propose a Machine Learning solution to minimize the threat posed by SQL Injection using various supervised, and unsupervised algorithms and plan to implement a Reinforcement Learning algorithm to detect a SQL attack beforehand and prevent any potential data loss or theft.

5.2 Design Details

Architecture Diagram

An architectural diagram is a visual representation that maps out the physical implementation for components of a software system. It shows the general structure of the software system and the associations, limitations, and boundaries between each element.

The architecture of the project includes two databases, a detection model and a user interface.

There are two databases, one for storing user details and one for storing the training SQL injection training data. The SQL injection detection model refers to the database having SQL injection to train itself and predict whether the input is an injection or not.

The User interface comprises the signup and login section. The signup section would register the user whereas the login section would be the interface containing the input field through which actual login and injections can be passed for SQL injection detection.

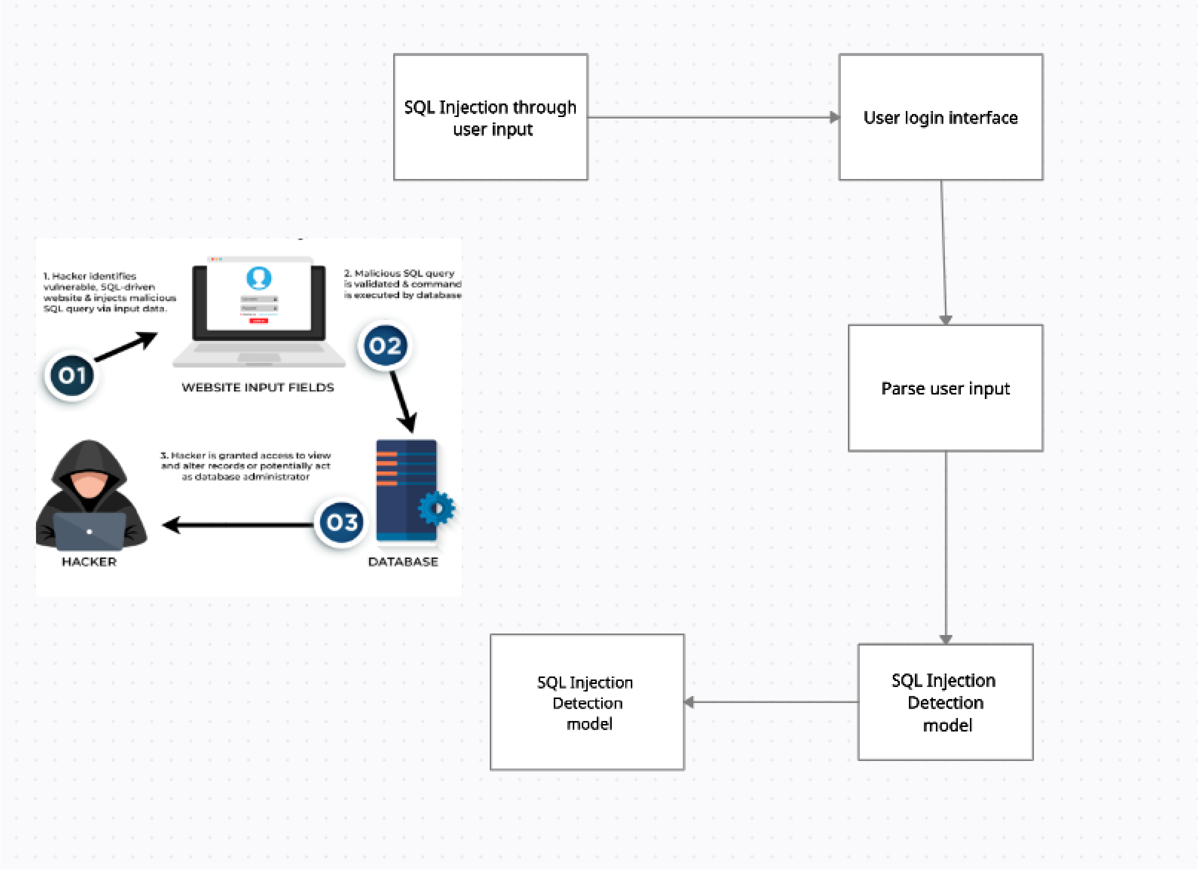

SQL Injection through user input
User login interface
Parse user input
SQL Injection Detection model
SQL Injection Detection model
1. Hacker identifies vulnerable, SQL-driven website & injects malicious SQL query via input data.
01
WEBSITE INPUT FIELDS
2. Malicious SQL query is validated & command is executed by database
02
3. Hacker is granted access to view and alter records or potentially act as database administrator
03
HACKER
DATABASE

Figure 5.1.1: Architecture Diagram

Data Flow Diagram

A Data Flow Diagram (DFD) is a graphical representation of the “flow” of data through an information system , modeling its process aspects. Often it is a preliminary step used to create an overview of the system that can later be elaborated.

Level-0 Data Flow Diagram

The general flow of the program is that the user interacts with our web page The user is asked to login if it is a valid user then they can enter their credentials and log in to the system. If the user is with malicious intent or attacker then they can perform SQL injection

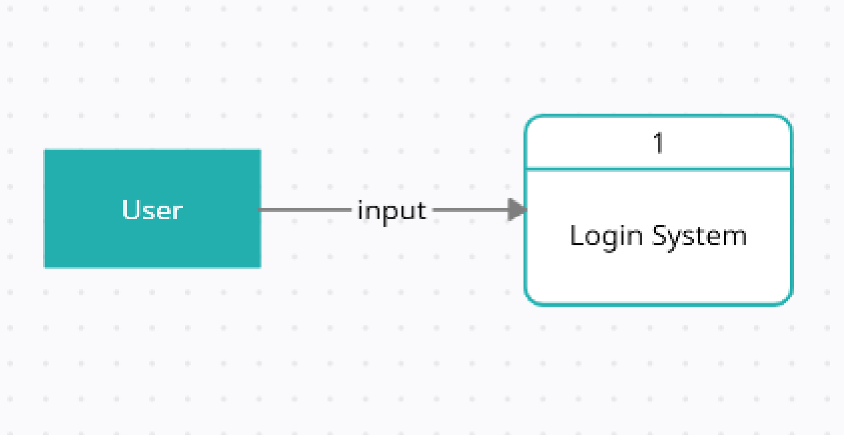


Figure 5.1.2: Data Flow Diagram (level 0)

Level-1 Data Flow Diagram

If a new user interacts with the Login System they can create a new account by providing details and signing up. By signing up, their details will be stored in the user database. If an existing user wants to log in then they can login by providing input through the login interface. The user has to provide username and password as input data which is checked for any SQL injection. If the password by the user matches that in the user database then it is a successful login else it is unsuccessful. If a user inputs an injection then it will be detected by the SQL injection model which is trained over dataset containing SQL injection and display an “attack detected” message

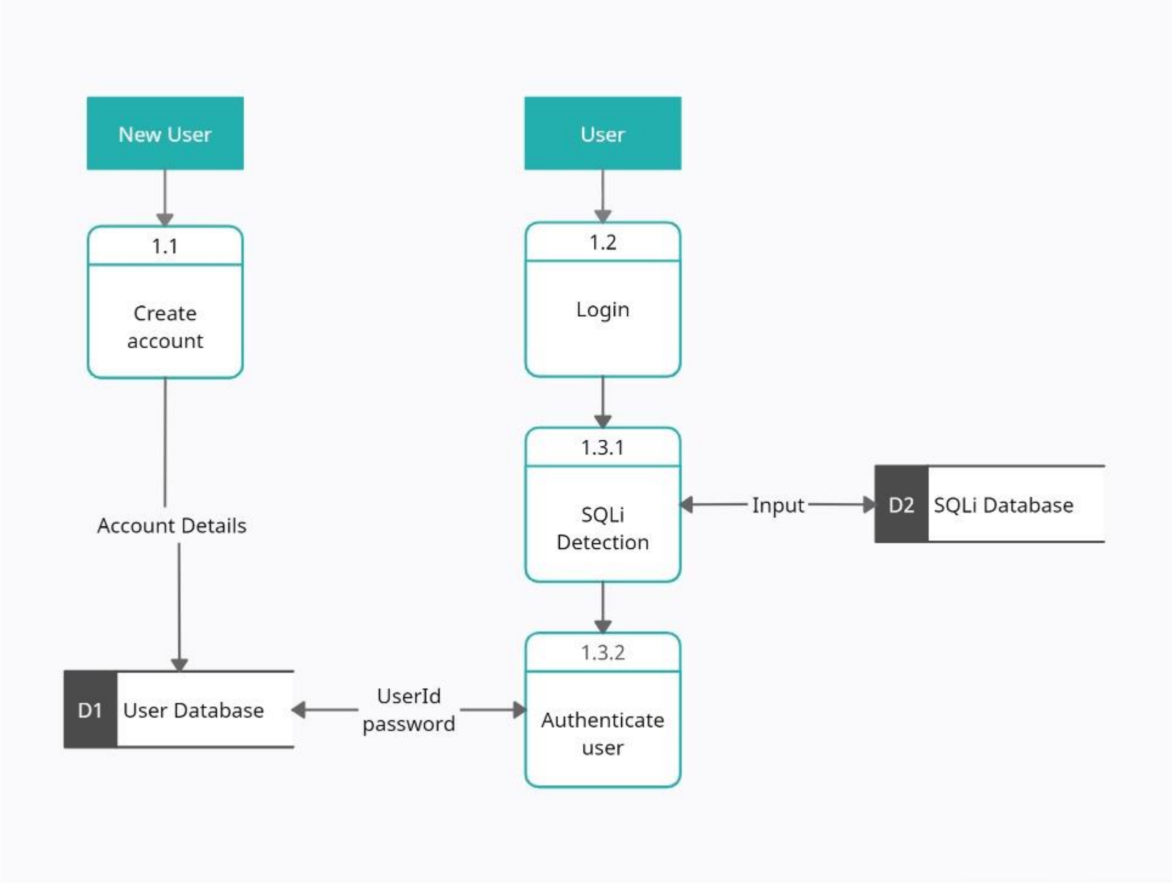


Figure 5.1.3: Data Flow Diagram (level 1)

Sequence Diagram

A sequence diagram is a Unified Modeling Language (UML) diagram that illustrates the sequence of messages between objects in an interaction. A sequence diagram consists of a group of objects that are represented by lifelines, and the messages that they exchange over time during the interaction.

The sequence diagram with respect to the user can be explained as follows. First, the user provides an input through the user interface implemented on the web browser. In the backend, input is sent to the SQL injection detection model. If the model classifies the input as injection then an SQL injection attack detected message will be shown to the user. If the injection is not classified as SQL injection then it consults the user database and matches the password for the input user. If the password is matched then successful login message is displayed else unsuccessful login message is displayed.

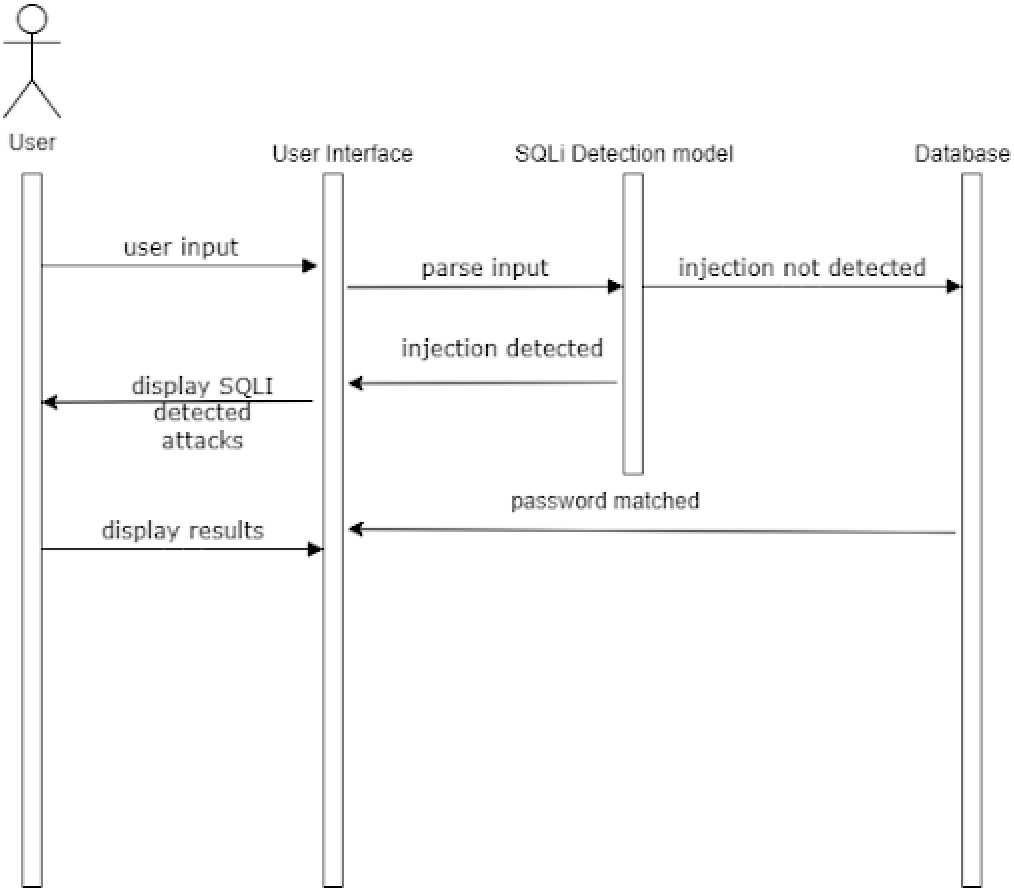


Figure 5.1.4: Sequence Diagram

Activity Diagram

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent.

The user will input the data through the login interface and the detection model will check if the input is an injection or not. If the input is classified as an injection then a message displaying that an injection was detected will be shown. If the input is not classified as an injection then the login credential will be checked from the user database and will show if the login was successful or not.

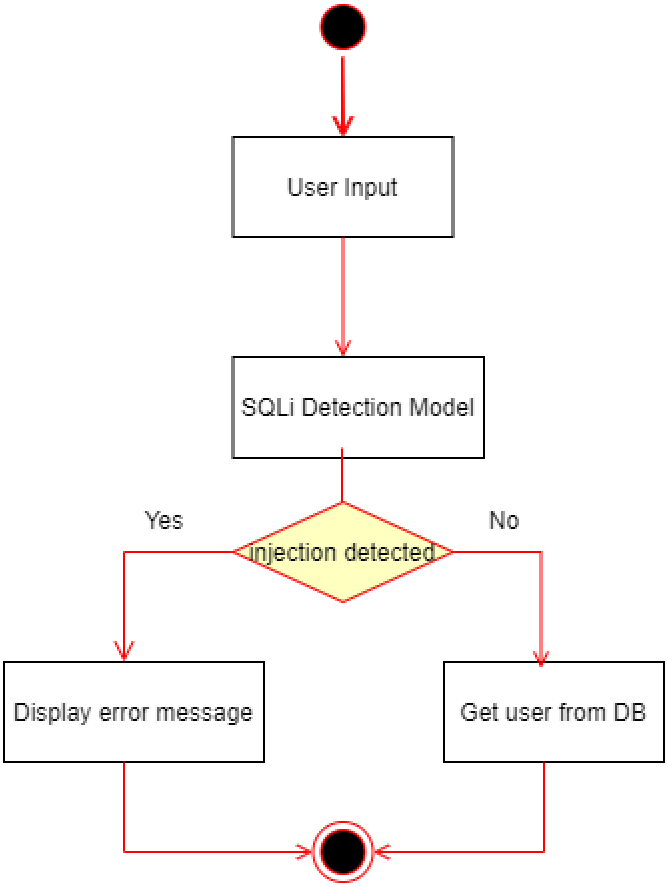


Figure 5.1.5: Activity Diagram

5.3 Methodology

The user provides an input through the user interface implemented on the web browser. In the backend, input is sent to the SQL injection detection model. If the model classifies the input as injection then an SQL injection attack detected message will be shown to the user. If the injection is not classified as SQL injection then it consults the user database and matches the password for the input user. If the password is matched then successful login message is displayed else unsuccessful login message is displayed.

For choosing the model:

We have studied the various supervised and unsupervised algorithms used for the detection and prevention of SQL injection attacks. A comprehensive dataset was created considering all the data types of SQLi attack queries. We have compared multiple models used for detection like Naive Bayes, Logistic Regression, Random Forest, SVM, Decision Tree, CNN and BERT. CNN and BERT models are found to show the most consistent accuracy and F1 scores. CNN model has a precision and F1 score of 97.26% and 94.85% respectively. The BERT model has a precision and F1 score of 99.80% and 99.73% respectively.

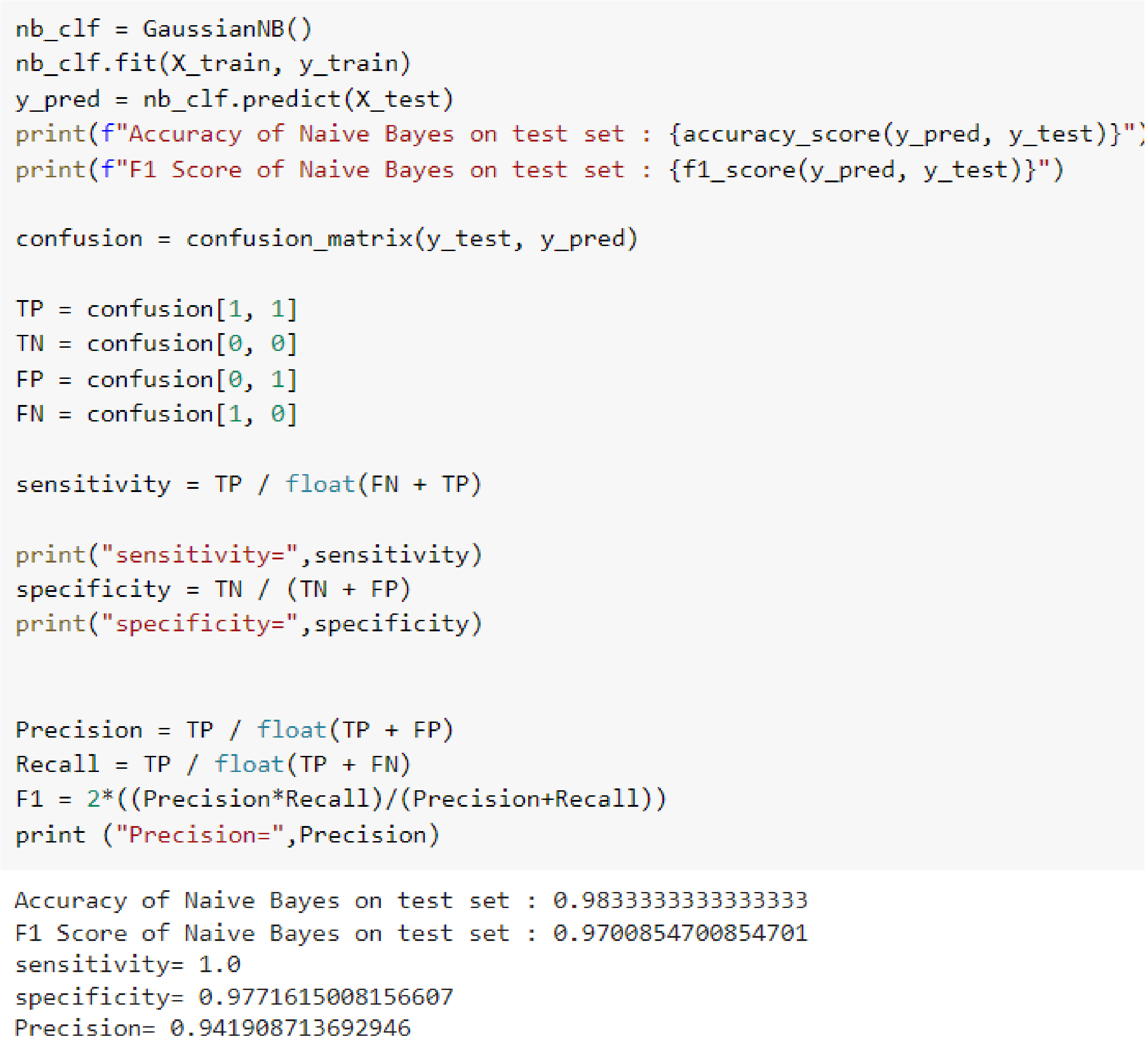
Since these results are obtained after training the model on an exhaustive dataset, we can conclude that CNN and BERT can prove to be a good a priori for the Reinforcement Learning model which could improve the performance measures.

5.3.1 Working of Models

1] Naive Bayes:

Naive Bayes is a technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a normal (or Gaussian) distribution. This means if predictors take continuous values instead of discrete ones, then the model assumes that these values are sampled from the Gaussian distribution.



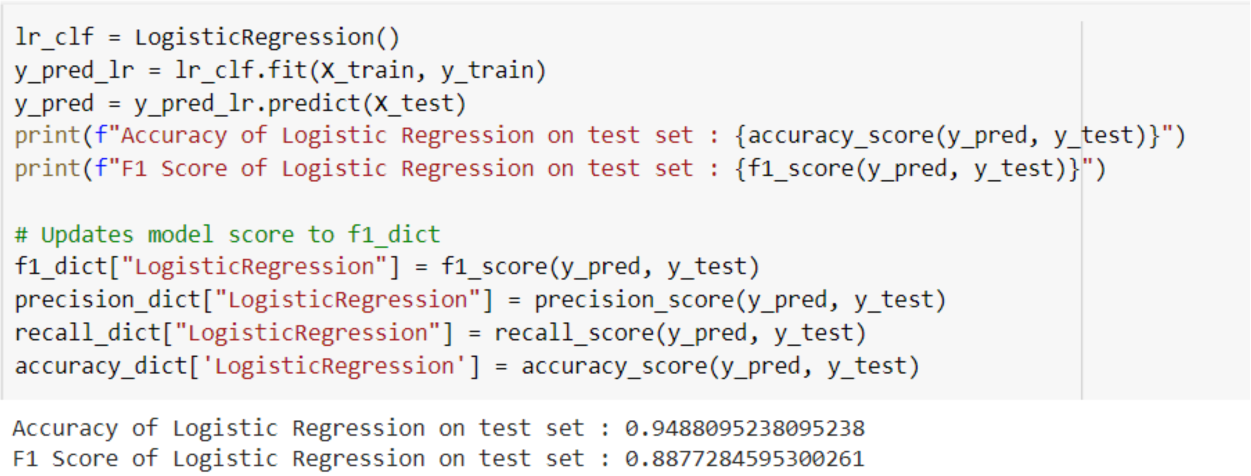
```
nb_clf = GaussianNB()
nb_clf.fit(X_train, y_train)
y_pred = nb_clf.predict(X_test)
print(f"Accuracy of Naive Bayes on test set : {accuracy_score(y_pred, y_test)}")
print(f"F1 Score of Naive Bayes on test set : {f1_score(y_pred, y_test)}")

confusion = confusion_matrix(y_test, y_pred)

TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]

sensitivity = TP / float(FN + TP)

print("sensitivity=",sensitivity)
specificity = TN / (TN + FP)
print("specificity=",specificity)



Precision = TP / float(TP + FP)
Recall = TP / float(TP + FN)
F1 = 2*((Precision*Recall)/(Precision+Recall))
print ("Precision=",Precision)
```

```
Accuracy of Naive Bayes on test set : 0.9833333333333333
F1 Score of Naive Bayes on test set : 0.9700854700854701
sensitivity= 1.0
specificity= 0.9771615008156607
Precision= 0.941908713692946
```

2)Logistic Regression:

Logistic regression is a supervised learning algorithm used to predict a dependent categorical target variable and is based on the concept of probability. Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, True or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.

Logistic regression models the data using the sigmoid function, which is also known as the logistic function. The logistic function is an S-shaped curve that stretches from zero to one, while never being exactly zero and never being exactly one, either.It forms a curve like the "S" form.

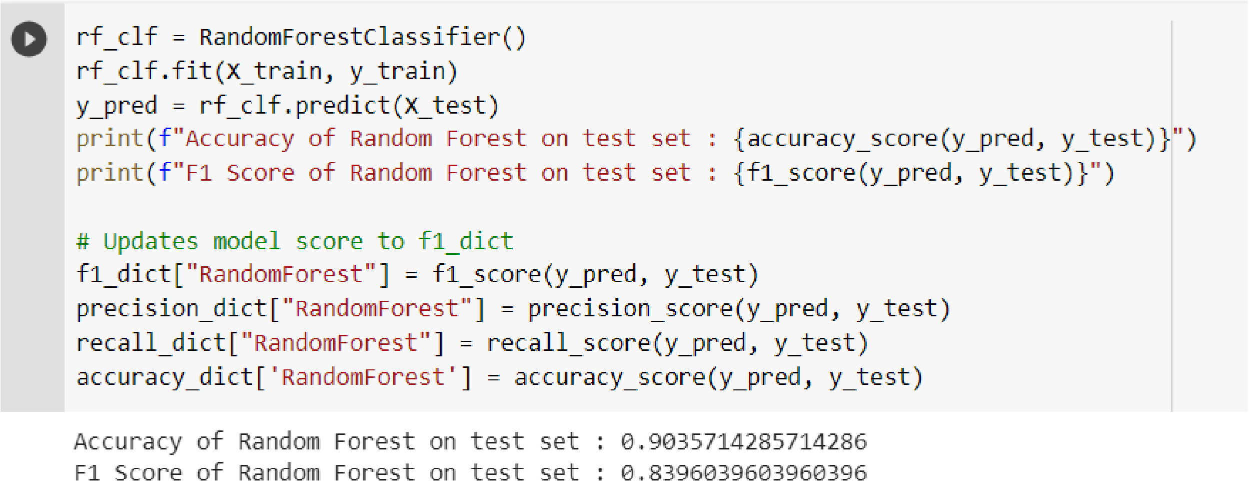


```
lr_clf = LogisticRegression()
y_pred_lr = lr_clf.fit(X_train, y_train)
y_pred = y_pred_lr.predict(X_test)
print(f"Accuracy of Logistic Regression on test set : {accuracy_score(y_pred, y_test)}")
print(f"F1 Score of Logistic Regression on test set : {f1_score(y_pred, y_test)}")

# Updates model score to f1_dict
f1_dict["LogisticRegression"] = f1_score(y_pred, y_test)
precision_dict["LogisticRegression"] = precision_score(y_pred, y_test)
recall_dict["LogisticRegression"] = recall_score(y_pred, y_test)
accuracy_dict['LogisticRegression'] = accuracy_score(y_pred, y_test)
```

```
Accuracy of Logistic Regression on test set : 0.9488095238095238
F1 Score of Logistic Regression on test set : 0.8877284595300261
```

3)Random Forest:

Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for decision trees' habit of overfitting to their training set. Random forests generally outperform decision trees, but their accuracy is lower than gradient-boosted trees. However, data characteristics can affect their performance.



```
rf_clf = RandomForestClassifier()
rf_clf.fit(X_train, y_train)
y_pred = rf_clf.predict(X_test)
print(f"Accuracy of Random Forest on test set : {accuracy_score(y_pred, y_test)}")
print(f"F1 Score of Random Forest on test set : {f1_score(y_pred, y_test)}")

# Updates model score to f1_dict
f1_dict["RandomForest"] = f1_score(y_pred, y_test)
precision_dict["RandomForest"] = precision_score(y_pred, y_test)
recall_dict["RandomForest"] = recall_score(y_pred, y_test)
accuracy_dict['RandomForest'] = accuracy_score(y_pred, y_test)
```

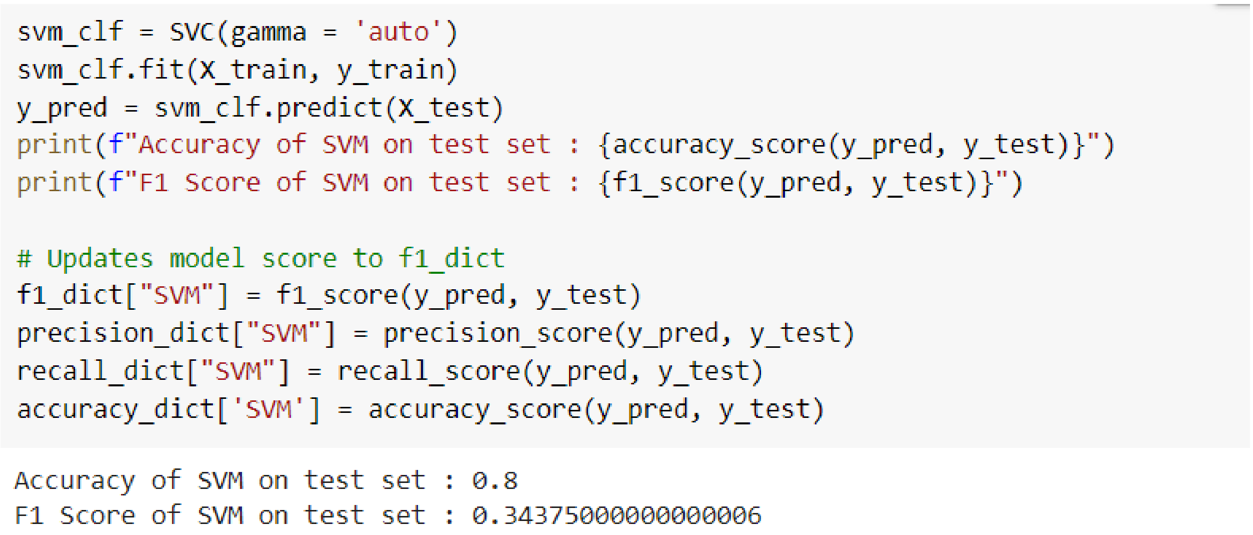
```
Accuracy of Random Forest on test set : 0.9035714285714286
F1 Score of Random Forest on test set : 0.8396039603960396
```

4)Support Vector Machine:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence algorithm is termed as a Support Vector Machine.



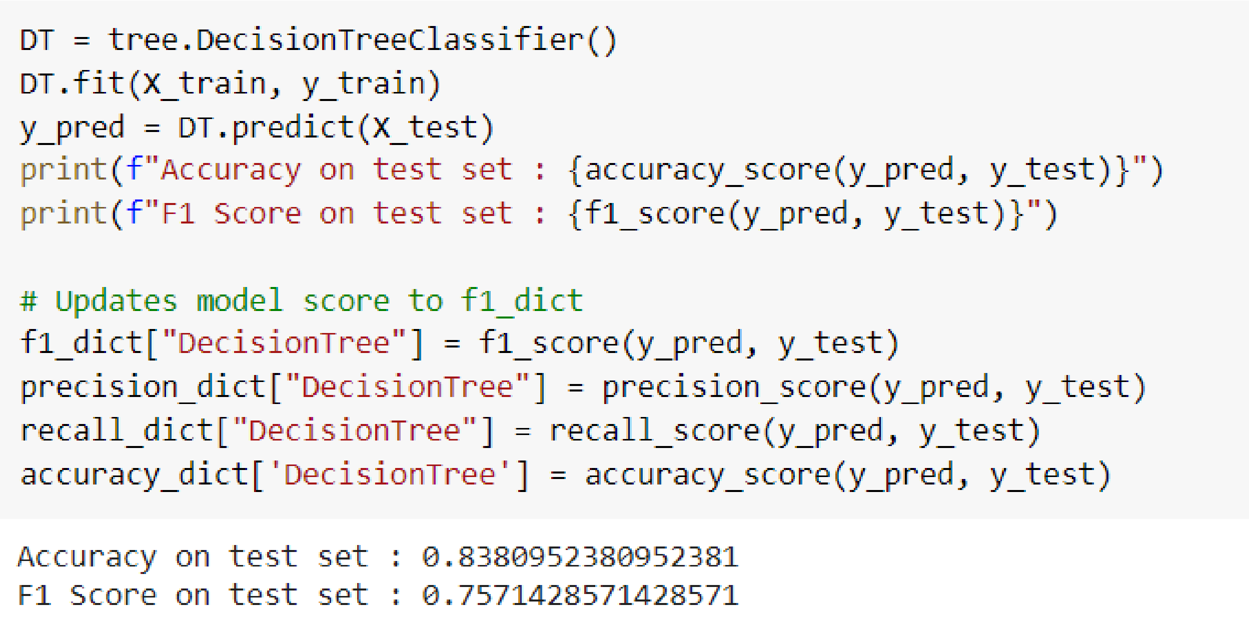
```
svm_clf = SVC(gamma = 'auto')
svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)
print(f"Accuracy of SVM on test set : {accuracy_score(y_pred, y_test)}")
print(f"F1 Score of SVM on test set : {f1_score(y_pred, y_test)}")

# Updates model score to f1_dict
f1_dict["SVM"] = f1_score(y_pred, y_test)
precision_dict["SVM"] = precision_score(y_pred, y_test)
recall_dict["SVM"] = recall_score(y_pred, y_test)
accuracy_dict['SVM'] = accuracy_score(y_pred, y_test)
```

```
Accuracy of SVM on test set : 0.8
F1 Score of SVM on test set : 0.34375000000000006
```

5)Decision Tree Classifier:

A decision tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of the root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node. For the next node, the algorithm again compares the attribute value with the other sub-nodes and moves further. It continues the process until it reaches the leaf node of the tree.



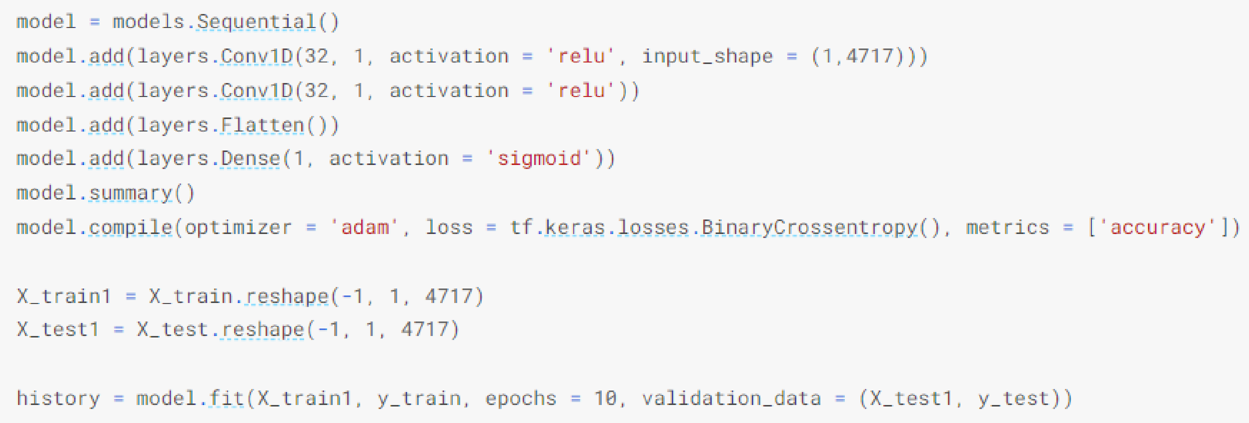
```
DT = tree.DecisionTreeClassifier()
DT.fit(X_train, y_train)
y_pred = DT.predict(X_test)
print(f"Accuracy on test set : {accuracy_score(y_pred, y_test)}")
print(f"F1 Score on test set : {f1_score(y_pred, y_test)}")

# Updates model score to f1_dict
f1_dict["DecisionTree"] = f1_score(y_pred, y_test)
precision_dict["DecisionTree"] = precision_score(y_pred, y_test)
recall_dict["DecisionTree"] = recall_score(y_pred, y_test)
accuracy_dict['DecisionTree'] = accuracy_score(y_pred, y_test)
```

```
Accuracy on test set : 0.8380952380952381
F1 Score on test set : 0.7571428571428571
```

6)Convolutional Neural Network:

Convolutional Neural Network is a deep learning technique that uses weights and biases to distinguish distinct aspects/objects from one another. A CNN requires the least amount of pre-processing when compared to the other models in this research. A CNN combined with an Intrusion Detection System (IDS) allows us to analyze traffic and make educated judgments. This enhances accuracy and outcomes, and the frequency of false warnings may be greatly decreased.

The accuracy of CNN on the test set was 0.9726 and the F1 Score of CNN on the test set was 0.9485.

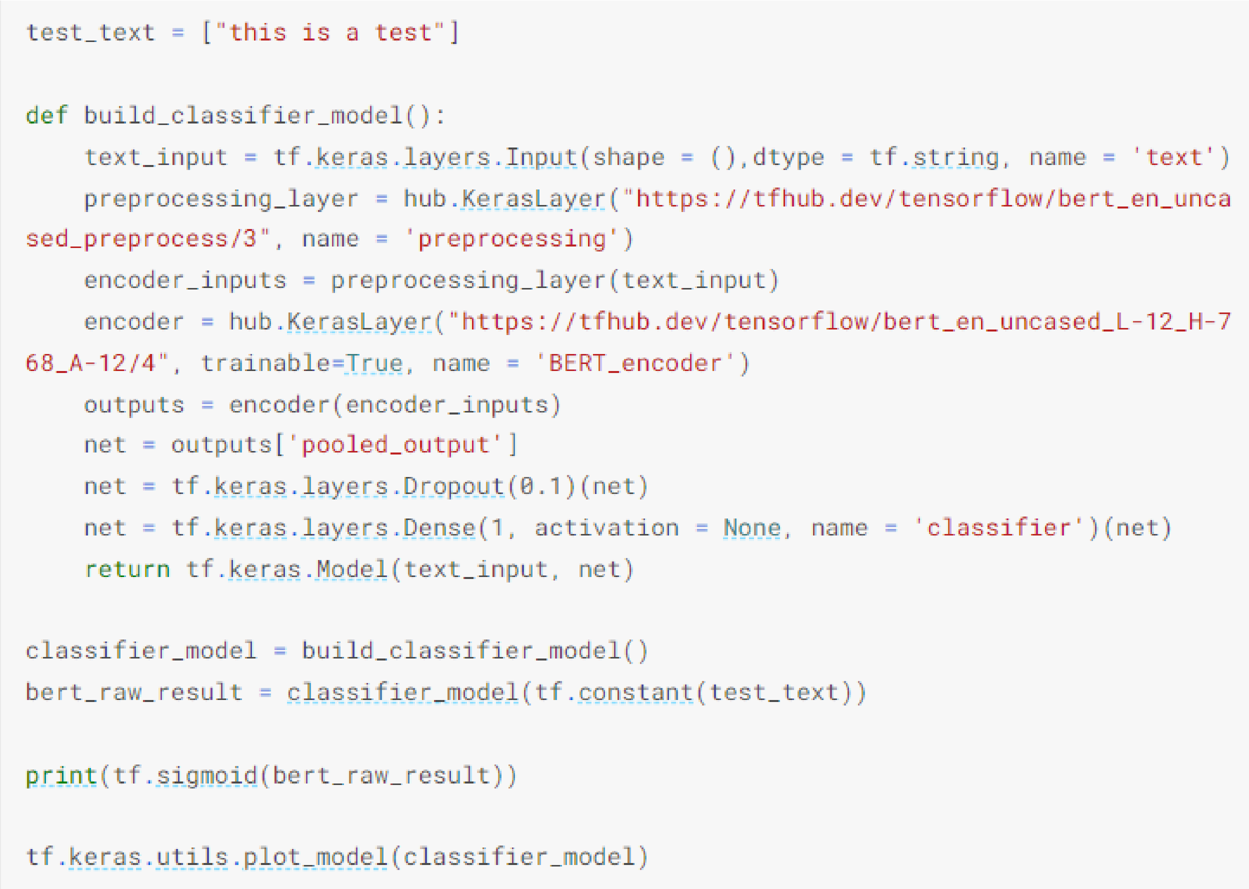


```
model = models.Sequential()
model.add(layers.Conv1D(32, 1, activation = 'relu', input_shape = (1,4717)))
model.add(layers.Conv1D(32, 1, activation = 'relu'))
model.add(layers.Flatten())
model.add(layers.Dense(1, activation = 'sigmoid'))
model.summary()
model.compile(optimizer = 'adam', loss = tf.keras.losses.BinaryCrossentropy(), metrics = ['accuracy'])

X_train1 = X_train.reshape(-1, 1, 4717)
X_test1 = X_test.reshape(-1, 1, 4717)

history = model.fit(X_train1, y_train, epochs = 10, validation_data = (X_test1, y_test))
```

7) BERT:

BERT (Bidirectional Encoder Representation from Transformers) is a language representation model designed by J. Devlin et. al.[21]to pre-train left and right-sided presentments by co-conditioning unlabeled data, unlike traditional language models. This model achieved an F1 score over 83%(the harmonic mean of precision and recall). BERT is an open-source pre-trained NLP model in which both autoencoders and word embeddings are utilized to efficiently represent data through projections onto appropriate vector space embedding for some data. By adding a suitable output layer to a pre-trained BERT model, significantly better results can be obtained in language processing tasks when compared to classical NLP methods. BERT uses two basic learning strategies to overcome contextual constraints and facilitates a bidirectional association. Masked Language Modeling (MLM) is performed before the word strings are transferred to the BERT model. 15% of the word strings are replaced with the [MASK] token. In this manner, MLM attempts to predict the original value of masked words based on the context formed by other unmasked words in the sequence. In Next Sentence Prediction (NSP) sentence pairs are taken as inputs to the model in the BERT training process. The objective is to train the model in such a way that the model can predict whether the second sentence in the pair is the next sentence in the document. In the training of the model, the second sentence in the original document is chosen for 50% of the inputs, and in the other 50%, the second sentence is randomly selected. In fact, a successfully trained model can determine that the sentence chosen randomly is not related to the first sentence. BERT can be considered as a stack of encoders and decoders. The main difference of BERT when compared to RNN, Attention, and Transformers is its double-sided examination of the text, which results in its improved handling of the relationship with the words on the right and left of the word under processing, and learning the content with MLM and NSP. The transformer network used by BERT consists of encoders and decoders that include self-attention mechanisms and feed-forward networks.

The accuracy of BERT is 0.998 and the F1 score of BERT was found to be 0.997 according to Shankar Das et. al.[22].



```
test_text = ["this is a test"]

def build_classifier_model():
    text_input = tf.keras.layers.Input(shape = (),dtype = tf.string, name = 'text')
    preprocessing_layer = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_unca
sed_preprocess/3", name = 'preprocessing')
    encoder_inputs = preprocessing_layer(text_input)
    encoder = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-7
68_A-12/4", trainable=True, name = 'BERT_encoder')
    outputs = encoder(encoder_inputs)
    net = outputs['pooled_output']
    net = tf.keras.layers.Dropout(0.1)(net)
    net = tf.keras.layers.Dense(1, activation = None, name = 'classifier')(net)
    return tf.keras.Model(text_input, net)

classifier_model = build_classifier_model()
bert_raw_result = classifier_model(tf.constant(test_text))

print(tf.sigmoid(bert_raw_result))

tf.keras.utils.plot_model(classifier_model)
```

8) Reinforcement Learning:

Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents should take an action in an environment to maximise the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. Reinforcement learning differs from supervised learning in not needing labelled input/output pairs to be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead, the focus is on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). Partially supervised RL algorithms can combine the advantages of supervised and RL algorithms. The environment is typically stated as a Markov decision process (MDP) because many reinforcement learning algorithms for this context use dynamic programming techniques. The main difference between the classical dynamic programming methods and reinforcement learning algorithms is that the latter does not assume knowledge of an exact mathematical model of the MDP and they target large MDPs where exact methods become infeasible.

5.3.2 Implementation

This is the user interface on visiting the website

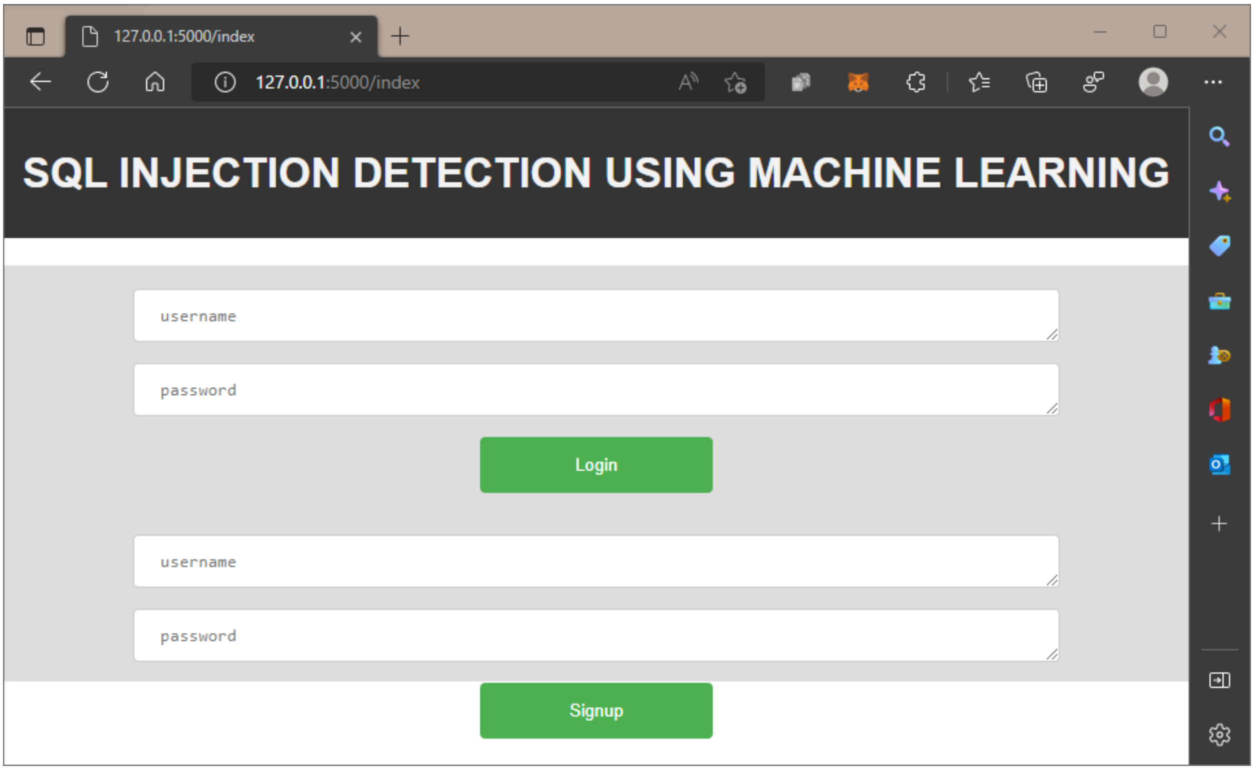
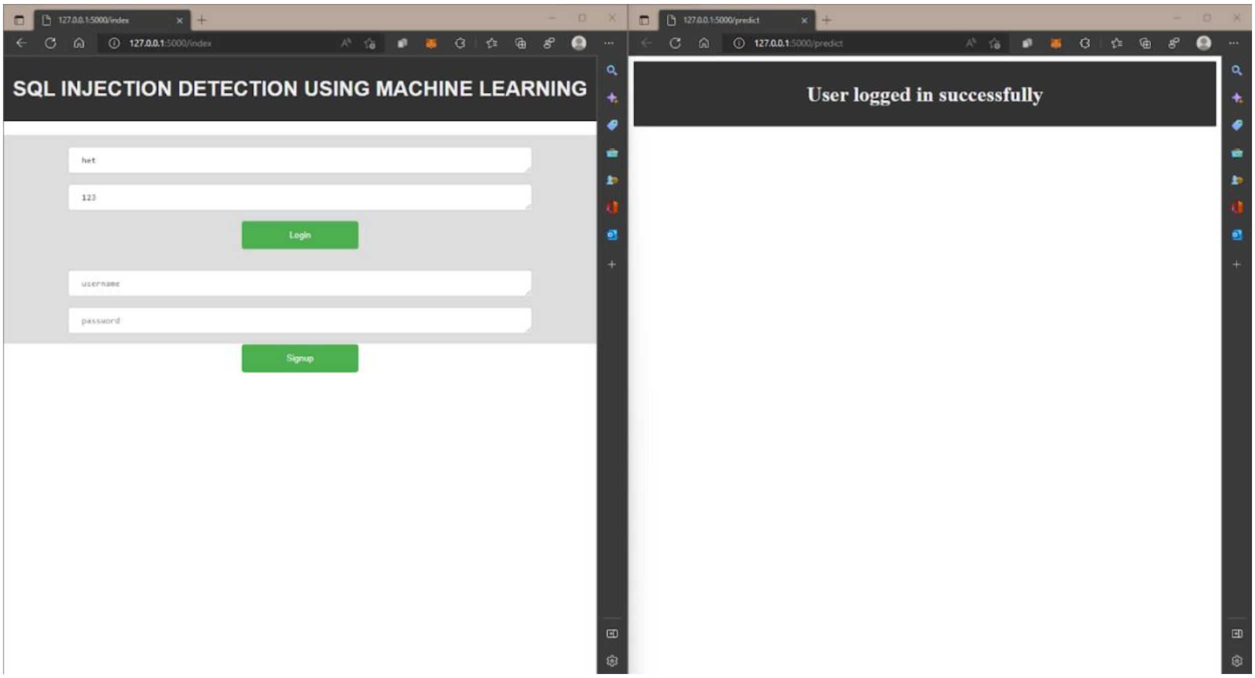


Fig 5.3.2.1: User interface

Once the user enters the credentials, the input is sent to the injection dataset to check if the entered string belongs to any class of injection. If it is not an injection, then the data is sent to our user database to check whether the user exists in our system.

Accordingly, the message is displayed.


127.0.0.1:5000/index

SQL INJECTION DETECTION USING MACHINE LEARNING

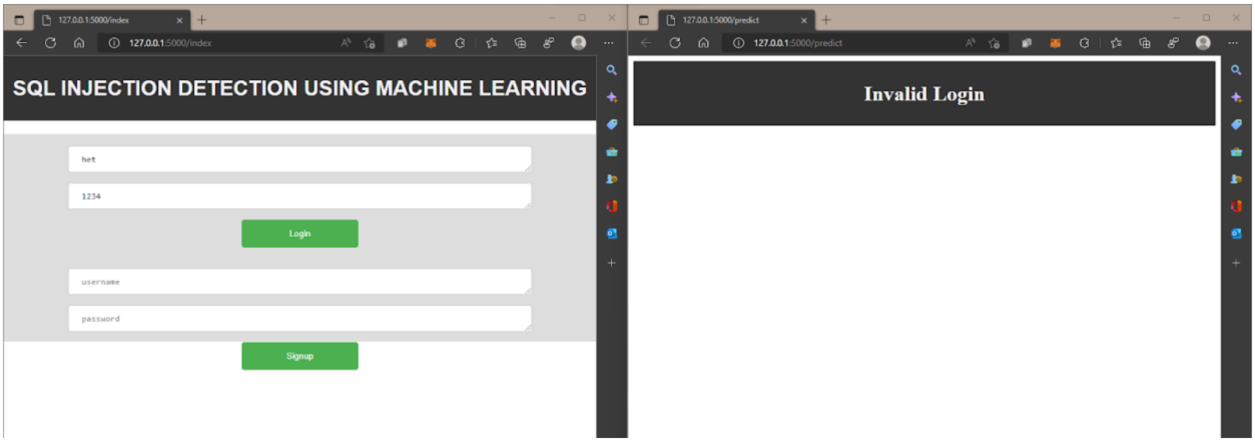
het

123

127.0.0.1:5000/predict

User logged in successfully

Fig 5.3.2.2: Successful login


127.0.0.1:5000/index

SQL INJECTION DETECTION USING MACHINE LEARNING

het

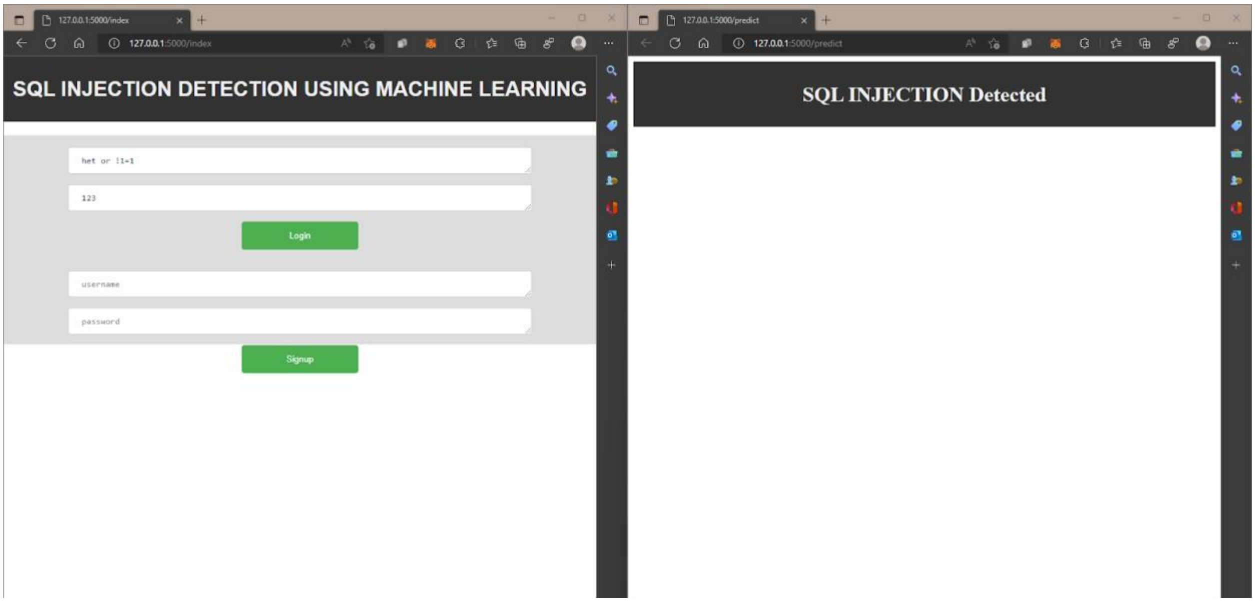
1234

127.0.0.1:5000/predict

Invalid Login

Fig 5.3.2.3: Invalid login

If the entered data is an injection , then following message is displayed


127.0.0.1:5000/index

SQL INJECTION DETECTION USING MACHINE LEARNING

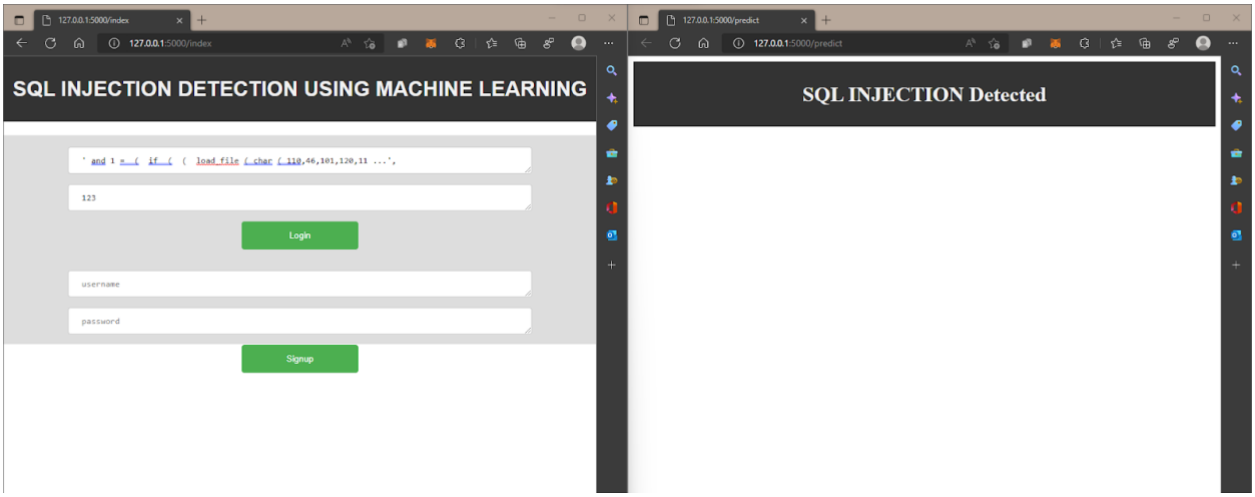
het or !1=1

123

127.0.0.1:5000/predict

SQL INJECTION Detected

Fig 5.3.2.4: Injection detected #1


127.0.0.1:5000/index

SQL INJECTION DETECTION USING MACHINE LEARNING

' and 1 = (if ((load_file (char (110,46,101,120,11 ...',

123

127.0.0.1:5000/predict

SQL INJECTION Detected

Fig 5.3.2.5: Injection detected #2

5.3.3 Results

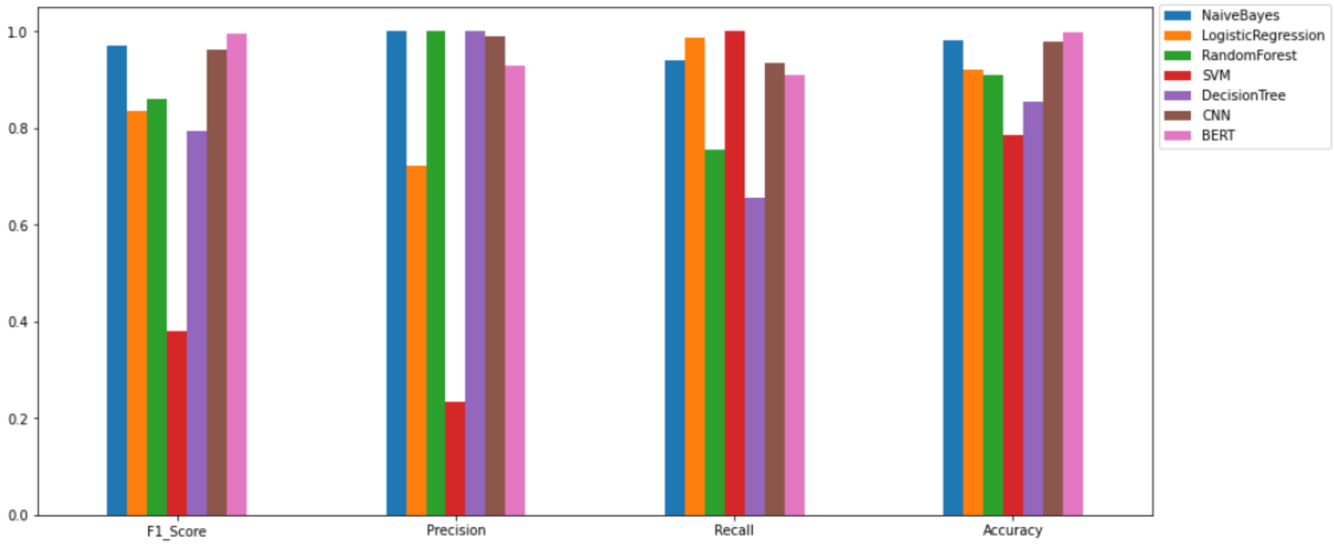


Fig 5.3.3.1: Comparison of the models

The performance measures of Naive Bayes, Logistic Regression, Random Forest, SVM, Decision Tree, CNN and BERT were compared. Where CNN model seems to have most stable performance measures. BERT model also provides better and stable performance measures as compared to the other models. Since these results are obtained after training the model on an exhaustive dataset, we can conclude that CNN and BERT can prove to be a good a priori for the Reinforcement Learning model which could improve the performance measures.

CHAPTER 6

Experimental Setup

6.1 Software Requirements:

1) **Python:** General Programming Language used to code the model, includes several libraries.
2) **Python libraries:** Numpy, Matplotlib, PIL (Python Imaging Library), pandas, Convolutional CNN.
3) **VS Code:** IDE to run, train and test the ML model. Plenty of extensions, open-source, cross-platform support,
4) **Git & GitHub:** Version Control System used for collaboration.
5) **Flask:** Flask is used for developing web applications using python, implemented on Werkzeug and Jinja2. The advantage of using the Flask framework is that here is a built-in development server and a fast debugger is provided.
6) **Google Workspace:** Project Management Tool to improve team coordination and file sharing.
7) **Teamgantt:** Project Management Tool to improve team coordination and file sharing using the Gantt chart.

6.2 Hardware Requirements:

1) **CPU:** 1.8 GHz or faster 64-bit processor; Quad-core or better recommended.
2) **Processor:** Intel i5 or greater.
3) **RAM:** Minimum of 4GB of ram.
4) **Storage:** 4GB of free hard disk space.

CHAPTER 7

Project Plan

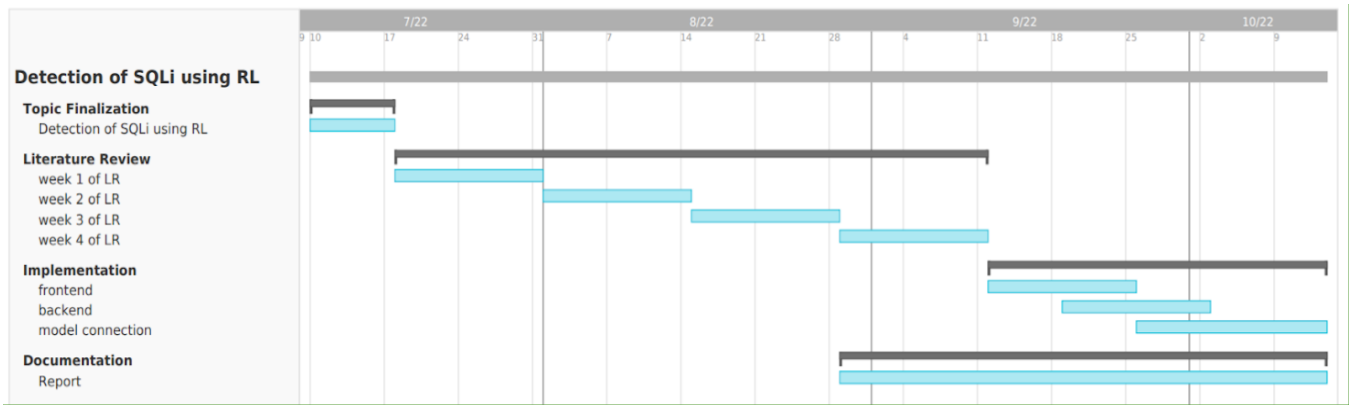


Fig 7: Gantt Chart

CHAPTER 8

Expected Outcome

We have studied the various supervised and unsupervised algorithms used for the detection and prevention of SQL injection attacks. A comprehensive dataset was created considering all the data types of SQLi attack queries. We have compared multiple models used for detection like Naive Bayes, Logistic Regression, Random Forest, SVM, Decision Tree, CNN and BERT. CNN and BERT models are found to show the most consistent accuracy and F1 scores. CNN model has a precision and F1 score of 97.26% and 94.85% respectively. The BERT model has a precision and F1 score of 99.80% and 99.73% respectively.

Since these results are obtained after training the model on an exhaustive dataset, we can conclude that CNN and BERT can prove to be a good a priori for the Reinforcement Learning model which could improve the performance measures.

References

[1] Erdődi, L., Sommervoll, Å. Å., & Zennaro, F. M. (2021). Simulating SQL injection vulnerability exploitation using Q-learning reinforcement learning agents. Journal of Information Security and Applications, 61, 102903.

[2] Zennaro, F. M., & Erdodi, L. (2020). Modeling penetration testing with reinforcement learning using capture-the-flag challenges: trade-offs between model-free learning and a priori knowledge. arXiv preprint arXiv:2005.12632.

[3] Verme, M. D., Sommervoll, Å. Å., Erdődi, L., Totaro, S., & Zennaro, F. M. (2021, November). SQL Injections and Reinforcement Learning: An Empirical Evaluation of the Role of Action Structure. In Nordic Conference on Secure IT Systems (pp. 95-113). Springer, Cham.

[4] John, A. (2015). SQL Injection Prevention by adaptive algorithm. IOSR journal of computer engineering, 17, 19-24.

[5] Hanmanthu, B., Ram, B. R., & Niranjan, P. (2015). SQL Injection Attack prevention based on decision tree classification. 2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO).

[6] Akinsola Jide, E. T., Oludele, A., & Idowu Sunday, A. (2020). SQL injection attacks predictive analytics using supervised machine learning techniques. International Journal of Computer Applications, (4), 139-149.

[7] Tang, P., Qiu, W., Huang, Z., Lian, H., & Liu, G. (2020). Detection of SQL injection based on artificial neural network. Knowledge-Based Systems, 105528. doi:10.1016/j.knosys.2020.105528

[8] Kamtuo, K., & Soomlek, C. (2016). Machine Learning for SQL injection prevention on server-side scripting. 2016 International Computer Science and Engineering Conference (ICSEC).

[9] Ross, K. (2018). SQL injection detection using machine learning techniques and multiple data sources.

[10] Qiang, W., & Zhongli, Z. (2011). Reinforcement learning model, algorithms and its application. 2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC).

[11] Tian, W., Yang, J.-F., Xu, J., & Si, G.-N. (2012). Attack Model Based Penetration Test for SQL Injection Vulnerability. 2012 IEEE 36th Annual Computer Software and Applications Conference Workshops. doi:10.1109/compsacw.2012.108

[12] Ghanem, M. C., & Chen, T. M. (2019). Reinforcement learning for efficient network

penetration testing. Information, 11(1), 6.

[13] Niculae, S., Dichiu, D., Yang, K., & Bäck, T. (2020). Automating penetration testing using reinforcement learning.

[14] Hu, Z., Beuran, R., & Tan, Y. (2020). Automated Penetration Testing Using Deep Reinforcement Learning. 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW).

[15] Sadeghian, A., Zamani, M., & Abdullah, S. M. (2013). A Taxonomy of SQL Injection Attacks. 2013 International Conference on Informatics and Creative Multimedia.

[16] Rai, A., Miraz, M. M. I., Das, D., Kaur, H., & Swati. (2021). SQL Injection: Classification and Prevention. 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM).

[17] Medhane, M. H. A. S. (2013). Efficient solution for SQL injection attack detection and prevention. International Journal of Soft Computing and Engineering (IJSCE), 3, 396-398.

[18] Halfond, W. G., Viegas, J., and Orso, A. A Classification of SQL-Injection Attacks and Countermeasures. In SSSE (2006).

[19] Wei, K., Muthuprasanna, M., & Suraj Kothari. (2006). Preventing SQL injection attacks in stored procedures. Australian Software Engineering Conference (ASWEC’06).

[20] Dr. Drew Hwang, “SQL Injection”, 2022, https://hwang.cisdept.cpp.edu/swanew/text/SQL-Injection.htm

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, ‘‘BERT: Pre-training of deep bidirectional transformers for language understanding,’’ 2018, arXiv:1810.04805.

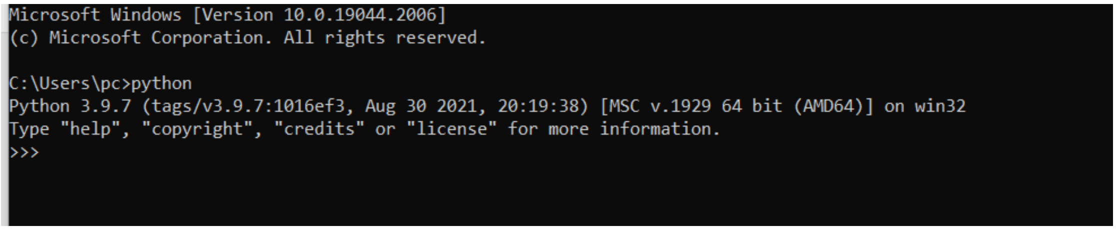
[22] Shankar Das, S., Serra, E., Halappanavar, M., Pothen, A., & Al-Shaer, E. (2021). V2W-BERT: A Framework for Effective Hierarchical Multiclass Classification of Software Vulnerabilities. arXiv e-prints, arXiv-2102.

[23] “OWASP Top Ten”, https://owasp.org/www-project-top-ten

Appendix

Appendix A
Python Download and Installation

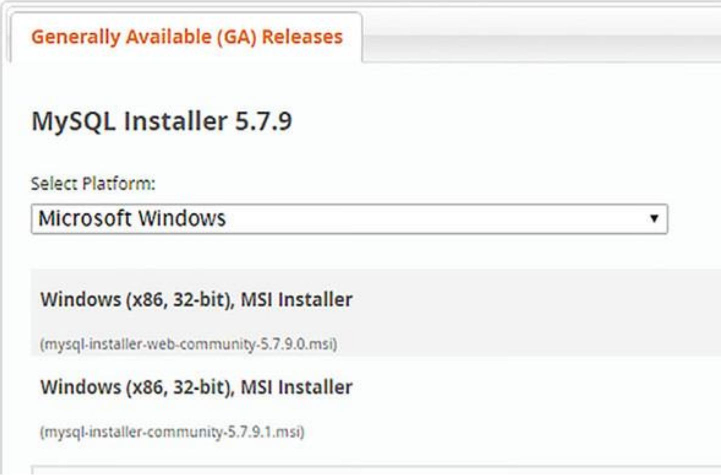
1. Visit the official website and go to https://www.python.org/downloads/. Click the Download button.
2. Once we click the download button, it might ask for a location to save the file. Select an appropriate location and then proceed towards the installation.
3. Double Click the downloaded .exe file and select the Add Python to PATH checkbox below to ensure it is automatically added to the Windows Environment variable. Else we have to do it later on manually. Once the box is checked, click on Install Now.
4. At the time of installation of python, the pop-up will show that the installation is in progress here.
5. Once the setup is complete, we will get a message like this. Click on the Close button to finish the installation of python.
6. Once Python is installed, go to the command prompt and type python , output like this should appear on successful installation.



```
Microsoft Windows [Version 10.0.19044.2006]
(c) Microsoft Corporation. All rights reserved.

C:\Users\pc>python
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Appendix B
MySQL Download and Installation

1. Download MySQL Installer from http://dev.mysql.com/downloads/installer/ and open it. Then select the suitable Setup Type you prefer.



Generally Available (GA) Releases

MySQL Installer 5.7.9

Select Platform:

Microsoft Windows ▾

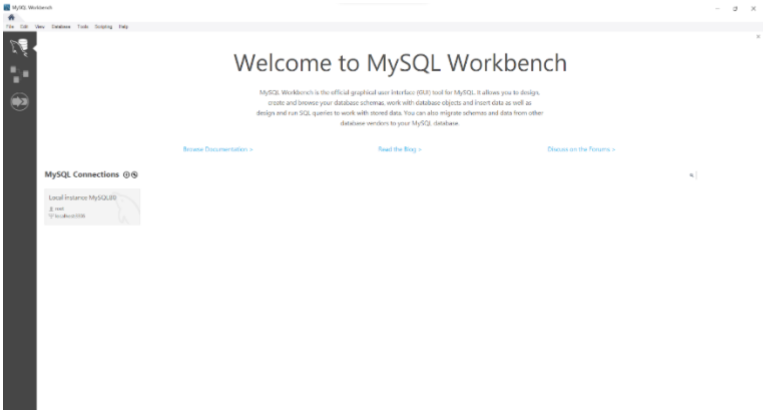
Windows (x86, 32-bit), MSI Installer

(mysql-installer-web-community-5.7.9.0.msi)

Windows (x86, 32-bit), MSI Installer

(mysql-installer-community-5.7.9.1.msi)

2. Then the MySQL installation wizard's instructions will guide you through the setup process. The installation completes. If you opted for a full install, by default MySQL server, MySQL workbench, and MySQL notifier will start automatically at computer startup.
3. The instance should be up and running, and you can connect to it using the MySQL workbench.



Welcome to MySQL Workbench

MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.

Browse Documentation >

Read the Blog >

Discuss on the Forums >

MySQL Connections ⊕ⓢ

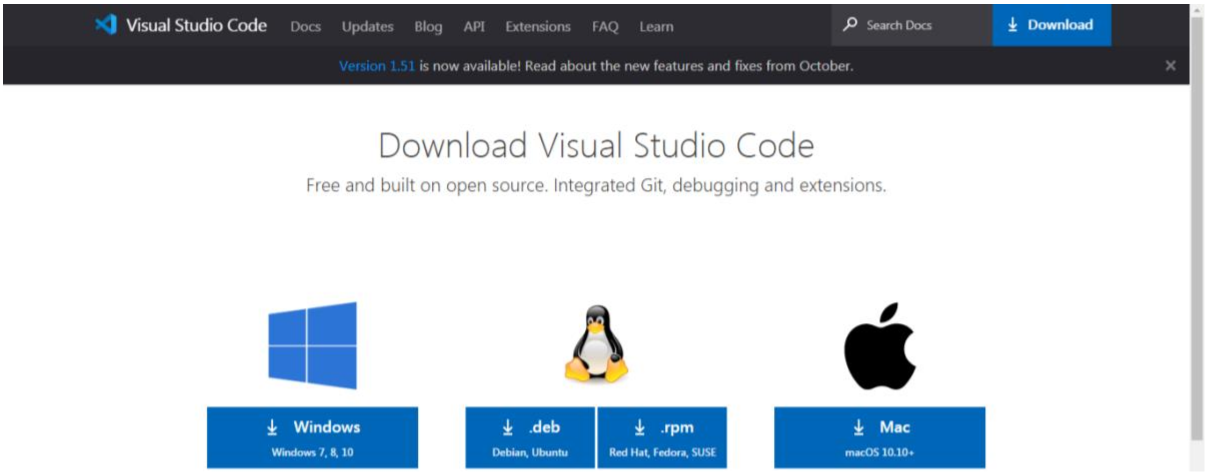
Local instance MySQL80

root

localhost:3306

Appendix C
VS Code Download and Installation

1. Download VS code from https://code.visualstudio.com/download
2. Download the Visual Studio Code installer for Windows. Once it is downloaded, run the installer (VSCodeUserSetup-{version}.exe). Then, run the file

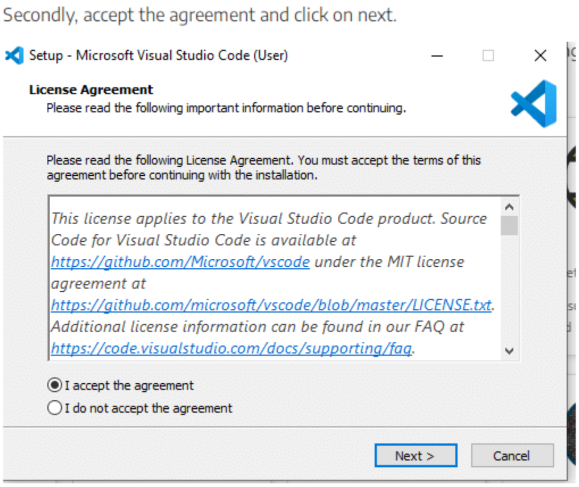


Version 1.51 is now available! Read about the new features and fixes from October.

Download Visual Studio Code

Free and built on open source. Integrated Git, debugging and extensions.

3. Accept the agreement and click “next.”



Secondly, accept the agreement and click on next.

License Agreement

Please read the following important information before continuing.

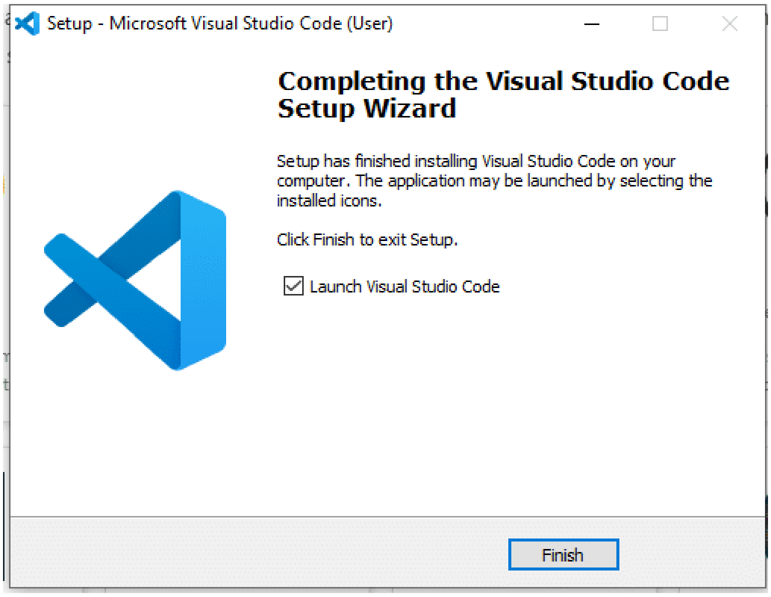
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.

This license applies to the Visual Studio Code product. Source Code for Visual Studio Code is available at https://github.com/Microsoft/vscode under the MIT license agreement at https://github.com/microsoft/vscode/blob/master/LICENSE.txt. Additional license information can be found in our FAQ at https://code.visualstudio.com/docs/supporting/faq.

● I accept the agreement

○ I do not accept the agreement

4. After accepting all the requests press the finish button. By default, VS Code installs under: “C:\users{username}\AppData\Local\Programs\Microsoft VS Code.”



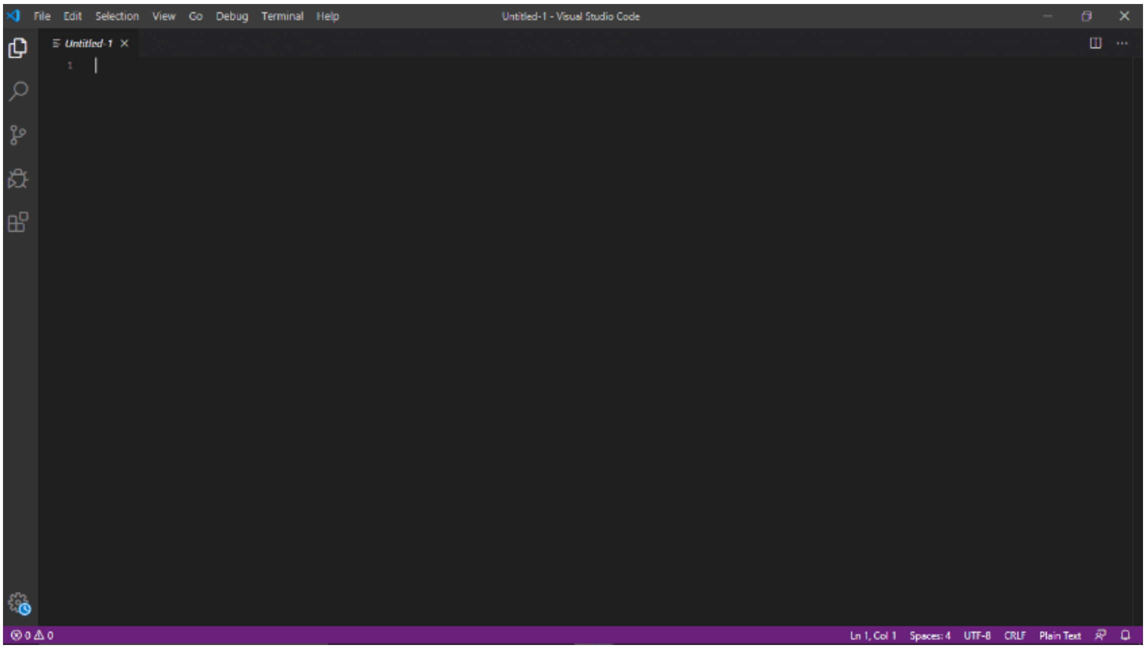
Completing the Visual Studio Code Setup Wizard

Setup has finished installing Visual Studio Code on your computer. The application may be launched by selecting the installed icons.

Click Finish to exit Setup.

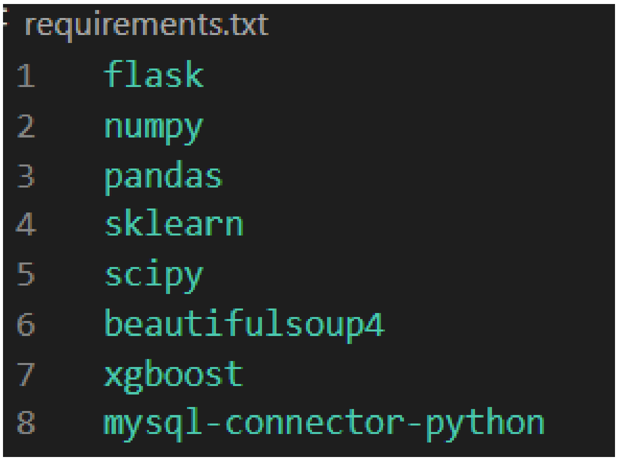
☑ Launch Visual Studio Code

If the installation is successful, you will see the following



Appendix D
Packages and libraries installation

pip install requiremnts.txt



requirements.txt

```
flask
numpy
pandas
sklearn
scipy
beautifulsoup4
xgboost
mysql-connector-python
```