Indonesian Journal of Electrical Engineering and Informatics (IJEEI)
Vol. 9, No. 3, September 2021, pp. 702~718
ISSN: 2089-3272, DOI: 10.52549/ijeei.v9i3.3131

SQL Injection Vulnerability Detection Using Deep Learning: A Feature-based Approach

Md. Maruf Hassan[1], R. Badlishah Ahmad[2], Tonmoy Ghosh[3]
[1,2]Universiti Malaysia Perlis, Perlis, Malaysia
[1,3]Daffodil International University, Dhaka, Bangladesh

Article Info

Article history:

Received Apr 28, 2021
Revised Jul 29, 2021
Accepted Jul 28, 2021

Keyword:

SQL injection
injection vulnerability detection
machine learning
neural network

ABSTRACT

SQL injection (SQLi), a well-known exploitation technique, is a serious risk factor for database-driven web applications that are used to manage the core business functions of organizations. SQLi enables an unauthorized user to get access to sensitive information of the database, and subsequently, to the application's administrative privileges. Therefore, the detection of SQLi is crucial for businesses to prevent financial losses. There are different rules and learning-based solutions to help with detection, and pattern recognition through support vector machines (SVMs) and random forest (RF) have recently become popular in detecting SQLi. However, these classifiers ensure 97.33% accuracy with our dataset. In this paper, we propose a deep learning-based solution for detecting SQLi in web applications. The solution employs both correlation and chi-squared methods to rank the features from the dataset. Feed-forward network approach has been applied not only in feature selection but also in the detection process. Our solution provides 98.04% accuracy over 1,850 recorded datasets, where it proves its superior efficiency among other existing machine learning solutions.

Corresponding Author:

Md. Maruf Hassan,
Universiti Malaysia Perlis,
Perlis, Malaysia
Email: ancssf@gmail.com

1. INTRODUCTION

The versatility of the internet raises the expectations of its user by offering limitless information and connectivity. Most businesses, therefore, reshape their existing organizational and commercial processes into internet-based solutions, i.e. web applications, to reach its targeted audience. Web applications provide different modes of user engagement in the system so that businesses can observe stakeholders' need for the product(s) and can offer them customized deals. All critical business information is stored in the application's database whereby owners or the system can make the appropriate decision for action. Therefore, ensuring the security of the stored data is a vital responsibility of the system's designer/developer.

However, a large number of web applications are developed without following the secured convention of software development [1]. As a result, web systems are vulnerable to different kinds of cyber-attacks [2]. According to the IBM X-Force Threat Intelligence Index, 79% of malicious incidents result from injection attacks. The number of injection attacks has increased 37% in 2017 compared to 2016 [3]. Also, since 2010 injection attacks have been at the top of the list of web application attacks, according to the Open Web Application Security Project (OWASP) [4,5]. These attacks can cause huge financial and reputational losses to business organizations [6–13].

An SQLi attack occurs due to SQLi vulnerabilities of web applications, which lead the intruder to perform an attack [14,15]. Such attacks mostly occur due to a lack of proper validation of inputs of web applications [16–23]. The OWASP's records from 2010 to 2017 reveal the carelessness of web developers about good practices for developing web applications.

Fig. 1 shows an approach to checking SQLi vulnerability. When there is an escape string, i.e., single quotation, backslash, etc. at end of the URL, it needs also to be at the end of the SQL query generated for the URL, which makes the SQL query a form of payload.

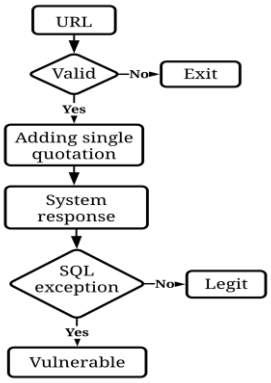

Figure 1. SQLi vulnerability check of a web application.

Records in the database correspond to the SQL query generated against the web application's URL. A query with a single quote at the end is unknown to the database for any records it contains. Therefore, this forces the database to throw exceptions from which the SQLi vulnerability of a web application can be known. After analyzing the exceptions, attackers try to inject several malicious SQL queries to get access to secured confidential information.

The following example explains the processes showed in fig. 1 that how SQLi vulnerability can be disclosed by converting a simple SQL query which is produced from a URL to a payload.

http://www.example.com/product.php?id=10

Let the above URL a targeted web application to be checked SQLi vulnerability.
After adding a single quotation at the end of URL it looks like,
http://www.example.com/product.php?id=10'

This generates a http request to the web application's server and the predefined SQL query for this URL tries get executed with the single quotation

```
SELECT * FROM products WHERE id_product=$id_product
//This query is to be executed
```

For the single quotation with the SQL query the database throws an error/exception as there is a syntactic mistake in the SQL query.

Warning: mysqli_fetch_array() expects parameter 1 to be mysqli_result, bool given in
/nfs/c05/h02/mnt/83231/domains/example.com/html/product.php on line 67

From this error/exception an attacker gets fingerprinting information and a determination that this web application builder might not follow all the coding conventions.

There are numerous SQLi detection approaches to diminish such attacks through SQLi vulnerability exploitation. These detection approaches can be of various types: detection using pattern matching [24-32], learning-based detection [33-43], and other approaches [44-49].

1.1. Pattern Matching Approaches

Pattern matching approaches are mostly focused on comparing a predefined benign SQL query with the SQL query generated at the runtime. Anything extra coming with the SQL query at the runtime is considered as an injection attempt.

The papers [24-32] use pattern matching approaches to detect and prevent SQLi attacks. These operations are performed at the runtime, which makes these practices time-consuming and complex. Wahid Rajeh and Alshreef Abed have proposed a three-tier method [24]. In their architecture, they use several types of operations to detect and prevent SQLi attacks. But their method seems costly and time-consuming. Mao Chenyu and Guo Fan [25] have proposed an intention-oriented approach to detect and prevent SQLi attacks. At the runtime they check every SQL query corresponding to their SQL intention description language, which is then converted into deterministic finite automata. Thereafter, it is checked if the request is for an SQLi attack or not. C. Shi et al. [26] proposed a self-learning SQLi detection approach with a pattern

matching and feature-filtering method. They generated a tree of selected SQL syntaxes to make a pattern and use that pattern to identify SQLi attacks. D. Kar et al. [27] developed a tool called SQLiDDS, which they used to transform particular portions of the SQL query into plain text and compared it with the runtime SQL query. Inyong Lee et al. [28] proposed a method for SQLi detection where the method removes the value of an SQL query attribute of web pages when a user submits parameters; it then compares the query with the predetermined query. A. Ghafarian [29] proposed a hybrid model where they monitored the execution of all incoming SQL queries dynamically and performed string matching between the received query and their expected query. Then they compared the result of the string matching process with the valid SQL query to identify SQLi attacks. In their proposed model, R.P. Karuparthi and B. R.P. Karuparthi and B. Zhou [30] proposed a dynamic query matching technique where they compared the executed runtime SQL query with a sanitizer, SQL master file then with predefined threshold value respectively to detect SQLi attack. The result is then sent to the approximate matching algorithm for analysis. A. Kumar and S. Binu [31] proposed a method where they made a set of tokens with the SQL query and matched them with user input at the runtime to detect SQLi attacks. N. Lambert and K. S. Lin [32] broke down various portions of the legal SQL query and used it in comparison with the user-defined SQL query to identify SQLi attacks.

1.2. Learning-based approaches

Learning-based approaches focus on studies to detect SQLi wherein solutions based on machine learning are used to solve the problems. Nowadays, learning-based approaches perform better than traditional rule-based approaches. Rawat and Shrivastav [33] proposed a detection approach based on SQL injection performable query tokenization [32] using a support vector machine (SVM). Kamtuo and Soomlek [34] inspected queries that could be used to perform SQLi, and they defined some specific conditions to produce a dataset in order to detect SQLi vulnerability in a web application. They compared various machine learning algorithms and received the best performance with the decision jungle algorithm. Zhuang Chen et al. [35] constructed a dictionary using the selected keywords from SQL injectable queries and HTTP requests, where the Word2vector algorithm was used to extract dataset from the dictionary. They also selected SVM as a learning algorithm to predict SQLi vulnerability. Shar and Tan [17] achieved above 85% prediction accuracy on SQLi and cross site scripting (XSS) vulnerabilities in different web applications using WEKA [36]. They implemented a tool named "PhpMinerI" to extract data by static analysis. Hua et al. [37] used statistical features and existing security knowledge to extract the features of SQLi attack requests. They proposed a web attack detection technology using SVM. Joshi and Geetha [18] used blank separation and query tokenization to prepare a dataset of SQLi detection features. They used a Naive Bayes algorithm to perform the detection operation. Moises et al. [38] analyzed the criteria of SQLMap to detect the frequencies of keywords and non-alphabetic characters used in SQLi. They used Naive Bayes and decision tree classifiers to classify SQLi attacks. D. Das et al. [39] indexed the strings of dynamic SQL queries and employed SVM to classify a runtime SQL query as normal or an attack attempt. Y. Wang and Z. Li [40] generated a parse tree of SQL queries, analyzed HTTP request parameters, and used them to compare with another parse tree. They used an SVM classifier to detect SQLi attempts. Other learning-based studies also found for phishing detection of web applications using features extraction methods.Sahingoz et al. [110] proposed a language-independent real-time anti-phishing system containing seven different algorithms for classification and NLP-based features to detect phishing websites. Kasim [111] introduced an approach that evaluated a phishing event using the classification of deep-hybrid features accompanied by the Light Gradient Boosted Machine model as soon as the web address entered the address bar. Yang et al. [112] proposed a deep learning-based fast phishing detection approach focusing on the multidimensional features. Lakshmi et al. [113] suggested a unique approach to discovering phishing websites where the hyperlinks exist in the HTML page's source code in the corresponding website. Basitet al. [114] reviewed various phishing attack detection techniques based on Artificial Intelligence (AI) to evaluate the qualities and shortcomings of the given methodologies.

1.3. Other Approaches

C. Ping [41] used the second-order SQLi to bypass the protection provided to web applications. They proposed adding random numbers to the selected keywords of executed queries to detect and prevent SQLi attacks. B.D. Priyaa and M. I. Devi [42] collected a query tree from database logs and extracted SQLi vulnerability features. They used an efficient data-adapted decision tree (EDADT) and binary SVM to effectively distinguish between SQLi attacks and normal SQL requests. P. Li et al. [43] proposed an SQLi attack detection technique that analyzes the user's interaction with the web application. They analyzed the user's log of web applications to find out the criteria of SQLi attacks and applied them to their model. A. Ciampa et al. [16] developed a tool to inspect SQLi vulnerability of web applications. K. Kemalis and T. Tzouramanis [44] proposed a technique where they extracted some syntactical structures of SQL queries of a web application and filtered runtime SQL queries based on those structures to detect SQLi attacks. V.

Shanmughaneethi and S. Swamynathan [45] proposed syntactic verification with XML and error message customization for SQLi attack detection.

Thus, researchers sought to detect and prevent SQLi attacks with either traditional rule-based approaches or machine learning-based approaches, while some worked with other approaches. However, no studies have so far used deep learning-based approaches in an SQLi attack scenario. Moreover, all the researchers focused on the SQL query, but no one considered the features of a web application to deal with SQLi attacks.

SQLi refers to the result of formidable threat to security and privacy concern for both client and web application. These threats become possible for not abiding the proper development conventions [1,30] by the developers which imposes the state of weakness of a web application. Lack of those conventions produce errors that provide information [29,30] about the technologies of that web application. Operators in SQL queries by which attackers can operate an attacking expedition to a web application are used to fuel learning based SQLi detection [34,35,38]. Payloads through user inputs [31] and clicking behavior [43] on different parts of a web application can also lead to SQLi attack.

In view of the above, it can be perceived that SQLi attack can be performed through different types of strategies and various parts of a web application. Those strategies and different parts together can be called web application's features by which a proper implication of the SQLi attacks can be delineated.

Among the types of SQLi detection solutions discussed earlier learning based approaches have become very popular due to its precision in detection problems. By this time shallow architectures such as Support Vector Machine (SVM), Naive Bayes are being used widely in various detection and classification problems. However, these architectures have their limitations which can produce problems in their respective ways. Some relatable drawbacks of SVM are, it is time and memory consuming [46-52], complexity in choosing right kernel function [51-54], high algorithmic complexity [50-52] and inability to work with massive data [55]. Independence assumption [47,51,56,57], sometimes providing bad results with large dataset [47,57] and sometimes providing good results with large dataset [48,50,58], these are relatable drawbacks of Naive Bayes. And for that, works done by other researchers on this issue might not provide very accurate results. Besides, deep learning is another learning based solution which has a very good reputation over other learning based solutions [59,60]. Moreover, deep learning based solutions have been shown providing better results than other learning based approaches [59-65] including SVM and Naive Bayes.

Aiming the above problem, this paper proposed a technique to detect web application's SQLi vulnerability based on various web features using deep learning. This technique focuses on web features that could be involved in sensitive data disclosing or can lead to unauthorized access to the database of a web application. Those web features have been elected after analysis of the responses resulted from various actions of SQLi on web applications. Moreover, appearances of the name of these web features in scholarly articles on SQLi increases the assessment of those features.

The contributions of the present study are as follows:

- A dataset with 19 SQLi vulnerability features has been produced using manual penetration, testing the following double-blinded testing strategy.
- A deep learning-based approach has been used to detect SQLi vulnerabilities.

The rest of the document is organized as follows:

Section II describes the overall architecture of our SQLi vulnerability detection framework and also tells how the framework works in detail. Section III shows the experimental results and presents the ultimate feature set used in our framework. Section IV provides a summary of this research and describes the future outlook.

2. PROPOSED MODEL

Fig. 2 shows the proposed model of SQLi vulnerability detection using a Feedforward Neural Network. The model is designed to find SQLi vulnerability focusing on predefined rules and deciding whether a web application is vulnerable or not. It describes the extraction, processing, and optimization of datasets. It uses feature selection methods along with the feed-forward neural network to bring the dataset into the best result-producing state. Finally, the neural network model is utilized to predict the SQLi vulnerability of the target web application with the dataset.

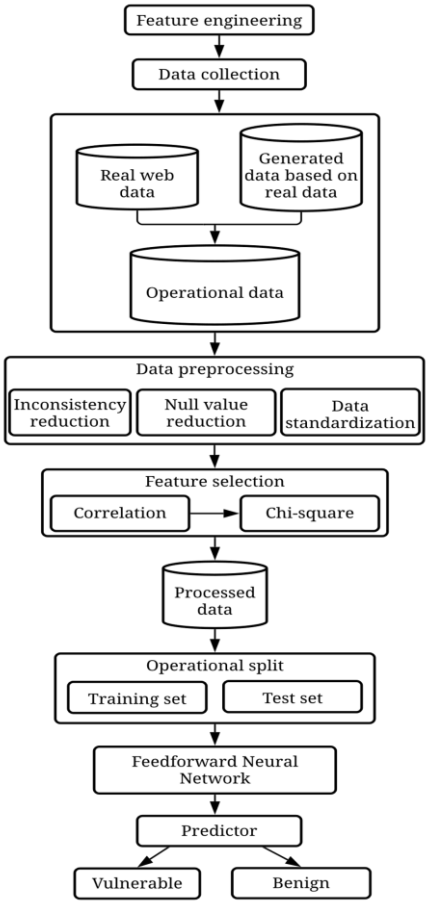

Figure 2. Proposed model of SQLi vulnerability detection.

Features are engineered on the basis of the logical conditions of SQL procedures and web applications usability facilities. Those features are used to perform data collection by checking every aspect of all the particular web applications. Hence, real web data related to SQLi have been acquired and with them some data which are generated based on real data have been combined to get an operational dataset. Those features then preprocessed by some predefined processing techniques and then feature selection methods have been applied to get the final dataset. After that, that dataset has been sent to perform the classification operation from which a decision can be taken about a particular web application.

From a user's perspective, the SQLi vulnerability detection processes using our model have the following steps:

(1) The user will take the desired web address of a web application and put it to a script written in Python.
(2) The user will be directed to the web application that could be vulnerable to SQLi. Basically, this web application is test data.
(3) The script will start processing to extract the features of the test data (current web application) and save them in a data structure.
(4) At this stage, to guess the type of the web application, the implemented model will be activated based on predefined rules learned from previous data collected from other web applications. To predict the types of the test data, the rules of the classifier are utilized.
(5) After the prediction operation, a warning will signal if the web application is vulnerable to SQLi; otherwise, that web application will be marked as benign.

The following sections explain the methodologies used for building the proposed framework. The framework supports the prediction model in such a way that it can predict the web application's SQL injection vulnerability. The technique used for SQL injection vulnerability detection and prediction, and the neural network model, research methods, and plan are described.

2.1. Formal definitions of dataset

As it has been said earlier, this paper focusing on web application's various features that can put a web application at risk and no state-of-art has been given an eye on that issue, implies a new dataset construction using the web features. That dataset consists of data found with practical testing on various web applications to maintain efficiency and precision. Based on those real life data some dummy data was created to prevent data inadequacy for the operation of deep learning models. Those data have been combined and went through some processing techniques to increase the integrity of the dataset.

Definition 1: D denotes the dataset, while C_j denotes a class in the dataset.

Definition 2: FeatureValue refers to the feature name and feature value. Feature name and value are denoted as (F_i, f_i).

Definition 3: The n^{th} row in the dataset D refers to a set of lists of feature values, which can be denoted as $(F_{n1},f_{n1}),\ldots,(F_{nk},f_{nk})$.

Definition 4: FeatureValueSet refers to a set of disjoint attribute values contained in a row as a training case, denoted as $\{(F_{i1},f_{i1}),\ldots(F_{ik},f_{ik})\}$.

Definition 5: {antecedent, C} denotes the form of a rule item r in the D, where the antecedent refers to FeatureValueSet and C refers to the class.

Definition 6: A single rule belonging to a class is represented as antecedent → C, where the antecedent refers to FeatureValueSet and C refers to the class.

Table 1. Dataset Sample

Instance Number	$Attr_1$	$Attr_2$	$Attr_n$	Class
1	F_1	F_2	F_n	C_0
2	F_1	F_4	F_n	C_0
3	F_3	F_7	F_n	C_1
4	F_5	F_6	F_n	C_1
5	F_2	F_5	F_n	C_0

2.1.1. Feature Extraction for SQLi Vulnerability

The extraction process has been executed in this study to find out the essential features in order to detect SQL vulnerability in a web application based on the previous literature. These features help the proposed model to classify the given weakness of any web application effectively and efficiently. In this study, 19 different features have been used to determine the class of vulnerability. These features are discussed in the following sections.

2.1.2. Selected features

The following subsection presents the features used in our detection operation, as well as their corresponding rules.

(1) Input Validation: Unvalidated input points of a web application offer an easy way to inject malicious SQL code in order to get access to confidential information [16-23].

```
Rule: If inputs are validated → Legit
            else → Suspicious
```

(2) Parameter Tampering: Parameters are used to identify records in the database. Tempering parameters provides exceptions when no records in the database can be identified [16,18,20,21,51].

```
Rule: If parameter temperable → Suspicious
            else → Legit
```

(3) Exception Handling: Exceptions are reactions of the databases when faulty SQL requests are sent to it. The database can disclose its SQLi vulnerability through exceptions [66].

```
Rule: If exceptions handled → Legit
            else → Vulnerable
```

(4) Use of parameterized query: The existence of parameters in query refers to a particular record in the database. Tempering with parameters raises exceptions [21,22,67,71].

```
Rule: Existence of parameter → Suspicious
            else → Legit
```

(5) Visibility of Page Extension: The visibility of page extension denotes that the possible web technologies have been used in the web application [68].

```
Rule: Existence of page extension → Suspicious
        else → Legit
```

(6) Illegal Input Acceptance: The acceptance of illegal inputs allows injecting any type of malicious codes that bypass the security logics of the database [21,69].

```
Rule: Acceptance of illegal inputs → Vulnerable
        else → Legit
```

(7) Error Controlling: Not controlling fatal errors in the web application and the database can increase chances of attack to the web application [16,21,23,72].

```
Rule: Controlling error → Legit
        else → Suspicious
```

(8) DB Info Disclosing: Database info helps attackers to find the drawbacks of a particular database technology [70-74].

```
Rule: Disclosing DB info → Suspicious
        else → Legit
```

(9) SQL Version Disclosing: The limitations of a specific SQL version can disclose exploitable vulnerabilities [23,71,75].

```
Rule: Disclosing SQL version → Suspicious
        else → Legit
```

(10) Guessable Table/Column Name: Gaining access to private information would become easier if an attacker could guess the table/column name of the database [76,77].

```
Rule: If easy to guess table column name → Suspicious
        else → Legit
```

(11) Directory Readability: An attacker can collect internal information if the directory can be read to perform a SQLi attack [73].

```
Rule: If directories readable → Suspicious
        else → Legit
```

(12) Allowing Null Byte: Force the application to minimize exceptions to continue the execution of malicious query [78].

```
Rule: Allows null byte → Suspicious
        Else → Legit
```

(13) Proper Implementation of Firewall Rules: Find out the level of secured firewall being used to protect the database as well as the web application [43].

```
Rule: Rule: If firewall implemented → Legit
        else → Suspicious
```

(14) Trust Intruders: Make decisions about whether the system allows suspicious activities or not [21].

```
Rule: Trust intruders → Suspicious
        else → Legit
```

(15) Use of Real Escape String: Deny the execution of SQL queries containing escape strings [47,79].

```
Rule: Escape string uses → Legit
        else → Vulnerable
```

(16) Parameter Encoding: Sometimes developers try to block SQL query execution by encoding the parameter [21,72].

```
Rule: Encoded Parameter → Legit/Suspicious
        else → Legit/Suspicious
```

(17) Encoding Data while Taking Input: Encoding user inputs can be used to defend against SQLi attacks [21,72].

```
Rule: If inputs are encoded → Legit/Suspicious
                else → Legit/Suspicious
```

(18) Execution of Malicious Code: If any kind of malicious code is executable in a web application, then it must be in serious danger [16,21,23,71].

```
Rule: If can execute malicious code → Vulnerable
                        else → Legit
```

(19) SQL Server Info: An attacker can exploit the system's database based on SQL server information [71,73].

```
Rule: Gaining SQL server info → Suspicious
                        else → Legit
```

2.2. Preprocessing

We have used inconsistency reduction, handling null value, and data standardization to preprocess the data. These techniques are commonly used to ensure the integrity of data in a dataset.

2.2.1. Inconsistency Reduction

Several types of inconsistencies have been discussed in the previous literature [80-82]. They cause serious problems in the dataset and come in the way of right decision-making, thereby decreasing usability [83]. Bonding between sets of data has a weakening effect and results in an undesired state of the dataset, which defines the question of integrity. To counter this problem, other researchers [82, 84-86] have used various techniques and mechanisms. Inconsistent data may lead to an inaccurate training accuracy in the model.

2.2.2. Missing Value Reduction

In data preprocessing, missing values remain a concern to the researchers because it spoils the efficiency of data in the dataset and forces the classification algorithms to produce unreliable and insubstantial results [87-89]. It could make features biased as well as hamper the desired output from the dataset [90]. Steps to handle this problem have been discussed on [89-92].

2.2.3. Data Standardization

The data processing technique converts the structure of an unbalanced dataset to a common data format, known as data standardization [93, 94]. It improves the quality of attributes in the dataset and prevents an attribute from dominating [96]. Data standardization transforms data from the dataset after the data is pulled from the source and before it is loaded into the model for training. Sometimes some features in the dataset influence the training process and make accuracy biased [93-95].

2.3. Feature Selection

Correlation coefficient and chi-square feature selection methods have been used to have a better understanding of the features. These feature selection methods were chosen because of their model independency. Correlation helps to find out the relationship between features, because highly correlated features can influence the training of the model and lead to biased prediction [97-99]. The chi-square method helps to understand the significance of the features by ranking them [100-103]. The later sections discuss these techniques.

2.4. Deep Learning for SQLi Vulnerability Classification

The deep learning model has been chosen for the prediction operation because it gives best-in-class performance in solving classification problems. In recent years researchers have had a good success rate in

solving detection problems using deep learning [59-65]. The deep learning model used in this paper contains 19 input neurons with one hidden layer. The activation functions used in the model are a rectified linear unit (ReLU) in the first layer and a sigmoid in the second layer. A formal description of the used neural network is provided below.

Let n be the neural network, $l^n = \{1,2,3\}$ a layer in the n neural network, x^n the input to the corresponding n neural network, $i^{(ln)}$ the incoming inputs to the layer l^n, r^n the output of the layer l^n, $w^{(ln)}$ the weights of the layer l^n, $b^{(ln)}$ the biases of the layer l^n, f the activation function (ReLU) in the layer l^n, and o the function in the output layer (Sigmoid). The equation of the feed-forward operation of the neural network is as follows:

$$r^{(0n)} = x \quad (1)$$

$$i^{(ln+1)} = w^{(ln+1)} + b^{(ln+1)} \quad (2)$$

$$r^{(ln+1)} = f(r^{(ln+1)}) = \max(0, r^{(ln+1)}) \quad (3)$$

$$r^{(3)} = o(r^{(3)}) = 1/(1+e^{-(r(3))}) \quad (4)$$

The predefined labels for the inputs are determined as shown in Equations (5) and (6). L defines the predefined labels of inputs, and L_0 defines the predicted label result.

$$\{r^{(3)} \geq 0.5, L = 1 \quad r^{(3)} < 0.5, L_0 = 0 \quad (5)$$

During the training process, the used neural network model is tuned to minimize the value of the loss function, i.e. cross-entropy function. The loss function is described as follows:

$$J(L, L_0) = -\sum_{i=1}^{2} L^{(i)} log L_0^{(i)} + \left(1 - L^{(i)}\right) log\ log\ \left(1 - L_0^{(i)}\right) \quad (6)$$

2.5. Regularization

In the case of neural network model training, overfitting is one of the major problems. An overfitted neural network with a particular train set cannot generally be used in classification. In the neural model of this research, dropout regularization [104] is used to avoid the overfitting problem. Dropout is a technique to skip some units of the neural network during the training operation. The incoming and outgoing connections of some neurons are removed, and during this the probability remains fixed. It prevents the neural network model from becoming too dependent on a specific set of units and their associated weights and biases. In this research the neural network is modeled with the dropout rate of 0.2. This rate is generally used in typical neural network-based models.

2.6. Validation

The overfitting problem discussed in the section on regularization can also be solved by cross validation. Here, K-fold cross validation [105] has been used to validate the efficiency of data in the dataset, as well as to make the training process effective. K-fold cross validation is a technique to use the dataset as a validation set by splitting the whole dataset into equal folds with the random selection of samples. During the validation process, one fold is randomly selected for test operation, while the rest is taken for training operation. This process is repeated until all the folds are used as test and training sets. This ensures traversing through all the data in a dataset and helps to have a good understanding of the dataset used in the model. The dataset in this research was split into several folds with 30 samples in each fold.

The novelty of the work includes creating a new dataset by extracting 19 SQLi features after observing the state-of-the-art literature and conducted black-box manual penetration testing to verify the legitimacy of the records. Feedforward Neural Network classification algorithm has been nominated to detect SQLi injection vulnerability where it uses both correlation coefficient and chi-square method for selecting features to increase its accuracy.

3. EVALUATION

3.1. Experimental Environment

We used a Windows 10 computer with Intel Core i3-5005U CPU, integrated GPU, and 4GB RAM to deploy our proposed model. We implemented the module of the model in Python. Most of the modules were executed on the CPython Interpreter [106]. The feed-forward neural network modeling tool was implemented using the Keras library [107]. Scikit-learn [108] and Tensorflow [109] were also utilized to implement the clustering and other machine learning algorithms.

3.2. Correlation of input attributes

Linear correlation is used to identify correlations between 19 input attributes of 1,850 SQL injection vulnerabilities and to find samples containing a dataset in the neural network. After finding the correlation among 19 features, iteration has been performed based on the correlation range of -1 to 1, using the score obtained from the correlation operation. In each phase of the iteration, the neural network has been trained using the reduced features based on the correlation range value to see which features are effective to detect the vulnerability. The iteration continues until the best accuracy is acquired. Fig. 3 shows the reduced features and the correlation heatmap between them.

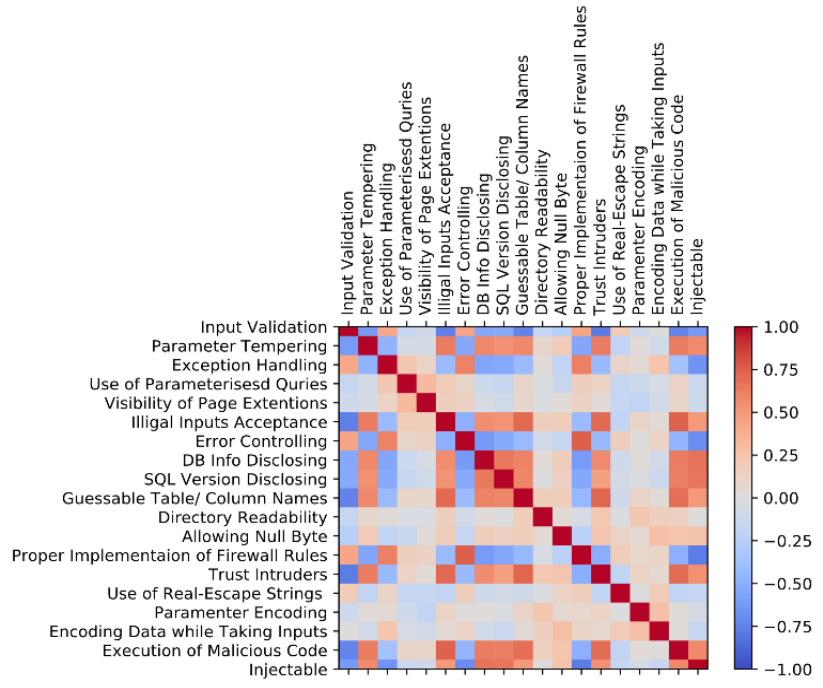

Figure 3. Visualization of correlated features.

3.3. Chi-square of correlated input attributes

The chi-square feature selection method is applied to the features reduced by correlation to find out the significance level of the features shown in Fig. 4.

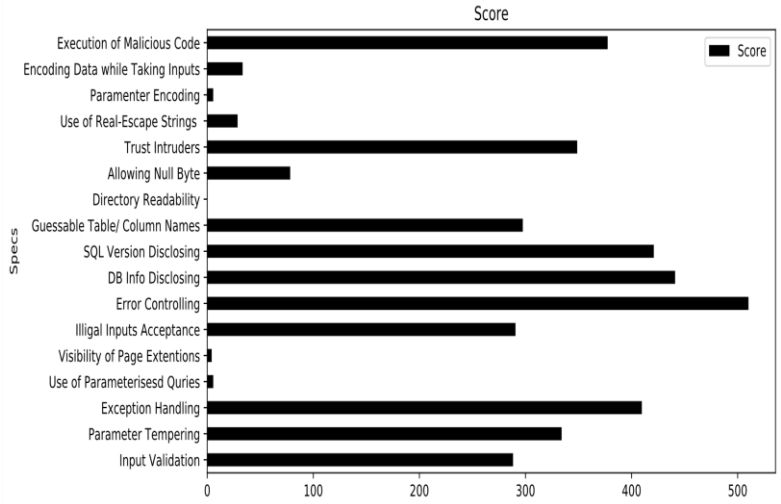

Figure 4. Plot describing chi-square level of significance. Each plot refers to a particular attribute and shows how important they are to the classification operation.

No samples were reduced during this process. After applying chi-square, the same type of iteration has been done as in the correlation operation to identify important features. This time the iteration is performed based on the chi-square value of significance. Table 1 summarizes the reduced features and their significant values.

Table 2. Extracted features and the value of importance from Fig. 5

Attributes	Chi-square scores	Significance
Input validation	288.256	Significant
Parameter tampering	334.047	Significant
Exception handling	409.634	Significant
Use of parameterized queries	5.68944	Not significant
Visibility of page extension	4.19805	Not significant
Illegal input acceptances	290.574	Significant
Error controlling	510.157	Significant
DB info disclosing	440.99	Significant
SQL version disclosing	420.94	Significant
Guessable table/column name	297.471	Significant
Directory readability	0.000650201	Not significant
Allowing null byte	78.2139	Significant
Trust intruders	348.714	Significant
Use of real escape string	28.6231	Not significant
Parameter encoding	5.63055	Not significant
Encoding data while taking input	33.3797	Significant
Execution of malicious code	377.44	Significant

3.4. Model validation

Model validation refers to the integrity and reliability of the neural network model. A validated model ensures a better acceptance of classification results. The used model was validated based on a comparison of the results of regularization and validation. Both suggest that the model is reliable to use in this scenario.

3.5. Deep learning model accuracy

The supervised dataset was split into 90% for testing and 10% for testing. The neural network used 500 epochs with a batch size of 30. The network was randomly initialized. With 17 input attributes from correlation and chi-square with more than 1,500 samples, the neural network model obtained 98.22% accuracy (Fig. 4). The loss of model (Fig. 5) denotes how bad the model is. The model sustained very little loss. Hence, it can be said that the model is close to accuracy.

Table 3. Comparison with other classifiers

Learning Process	Accuracy
SVM	94.66%
Random forest	97.33%
Naive Bayes	84.49%
Proposed Method (NN)	**98.04%**

Table 3 compares the different classifiers used in this research to check if any other machine learning algorithm can better result from the neural network. Here, an SVM, random forest, and Naive Bayes with an accuracy of 94.66%, 97.33%, and 84.49%, respectively, have been used because these were used by other researchers discussed in the learning-based approach in the section on related work. However, in this research, the neural network has provided the best performance with an accuracy of 98.04%.

Table 4. SQLi Vulnerability Detection Accuracy Comparison

Literatures	Accuracy
Shar and Tan [17]	85.00%
Joshi and Geetha [18]	93.30%
Rawat and Shrivastav [33]	96.47%
Lei and Shen [37]	93.30%
Lodeiro-Santiago et al. [38]	97-98%
Proposed Method (NN)	98.04%

Table 4 shows the SQLi vulnerability accuracy compared with the other five models. We found that 85.00%, 93.30%, 96.47%, 93.30%, and 97-98% accuracy obtained by the models of Shar and Tan [17], Joshi and Geetha [18], Rawat and Shrivastav [33], Lei and Shen [37], and Lodeiro-Santiago et al. [38], respectively. However, our proposed model ensured 98.04% accuracy.

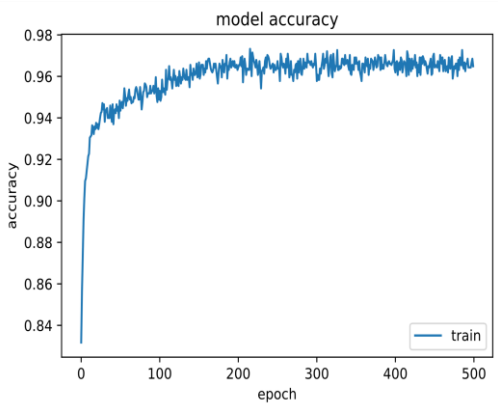

Figure 5. Adeptness of model at gaining accuracy.

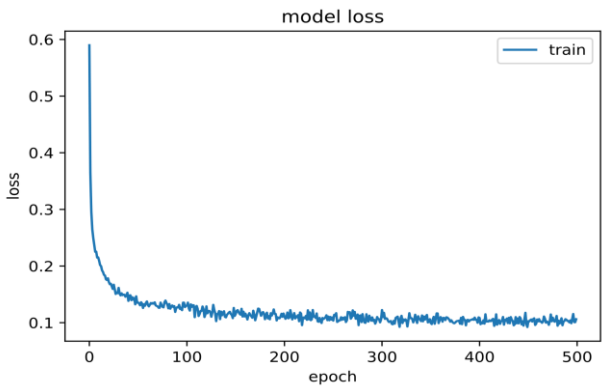

Figure 6. Adeptness of model at sustaining less loss.

Figs. 5 and 6 depict model training accuracy and loss, respectively. The determination of the training process can be visualized from these plots. The good accuracy shown in Fig. 4 has been due to the iteration performed. The model training loss shown in Fig. 5 is very low as we know less loss leads to good accuracy.

4. CONCLUSION AND FUTURE WORK

The objective of this paper has been to propose the framework of SQL injection vulnerability detection of a web application using deep learning by extracting various vulnerability finding points of the web application. In this paper, primarily the deep learning part is described since it is the core of the section on SQL injection vulnerability prediction. More than 1,850 samples of injection vulnerability with selected features are sent through a neural network model for prediction. The system delivers a performance accuracy of 98.04%, which is better than that provided by existing systems. Future experiments should focus on building an automated tool and an effective way to prevent SQL injection.

APPENDIX A

Data Collection Algorithm

An algorithm has taken place to perform the data collection operation against every feature. A python based technology has been used to build a tool which takes a particular web application and checks whether the predefined conditions of the features set meet or not. If the checking works according to the conditions then the data collection operation takes place. An overview of the algorithm has been given below for better understanding,

Input Feature Existance Based Data Collection

Input: **Predefined feature set:*F_set, URL, url_feature***
Output: **Input feature existence based data**

```
Input_feature ← |0 | 0 | … | 0|
index ← 0
if valid ← URL then
for ∀f1 ∈ F_set do // for all features in feature set
if f1 ∈ url_feature then
Input_feature[index] ← 1
return input_feature
```

APPENDIX B

Vulnerable Feature Dataset

Feature	Datatype	Category	Impact
Input Validation	Boolean	User input	Severe
Parameter Tampering	Boolean	User Input	Severe
Exception Handling	Boolean	Error/Exception	Severe
Use of parameterized query	Boolean	User input	Low
Visibility of Page Extension	Boolean	URL lookup	Low
Illegal Input Acceptance	Boolean	User input	Severe
Error Controlling	Boolean	Error/Exception	Severe
DB Info Disclosing	Boolean	Footprinting/Fingerprinting	Severe
SQL Version Disclosing	Boolean	Footprinting/Fingerprinting	Severe
Guessable Table/Column Name	Boolean	Unsecure coding convention	Severe
Directory Readability	Boolean	Unsecure coding convention	Low
Allowing Null Byte	Boolean	Unsecure coding convention	Moderate

Proper Implementation of Firewall Rules	Boolean	Unsecure coding convention	Moderate
Trust Intruders	Boolean	Unsecure coding convention	Severe
Use of Real Escape String	Boolean	Unsecure coding convention	Moderate
Parameter Encoding	Boolean	Encryption	Low
Encoding Data while Taking Input	Boolean	Encryption	Moderate
Execution of Malicious Code	Boolean	User input	Severe
SQL Server Info	Boolean	Footprinting/Fingerprinting	Moderate

REFERENCES

[1] A. Sadeghian, M. Zamani, and A. A. Manaf, "A Taxonomy of SQL Injection Detection and Prevention Techniques," 2013 International Conference on Informatics and Creative Multimedia, 2013.

[2] K. Gupta, "A Survey On Web Application Attack Detection Methods," International Journal of Advanced Research in Computer Science, vol. 8, no. 7, pp. 565–569, 2017.

[3] "IBM X-Force Threat Intelligence Index | IBM." [Online]. Available: https://www.ibm.com/security/data-breach/threat-intelligence. [Accessed: 27-Apr-2021].

[4] "Open Web Application Security Project | OWASP." report 2017. Available: https://www.owasp.org/index.php/Top_10_2017-Main. [Accessed: 27-Apr-2021]

[5] "Open Web Application Security Project | OWASP." report 2010. Available: https://www.owasp.org/index.php/Top_10_2010-Main. [Accessed: 27-Apr-2021]

[6] "Questions for TalkTalk | BBC News." October 26, 2015. Available: https://www.bbc.com/news/technology-34636308. [Accessed: 27-Apr-2021]

[7] "US man stole 130m card numbers | BBC News." August 18, 2009. Available: http://news.bbc.co.uk/2/hi/americas/8206305.stm. [Accessed: 27-Apr-2021]

[8] "RockYou Hacker: 30% of Sites Store Plain Text Passwords | New York Times." December 16, 2009. Available: https://archive.nytimes.com/www.nytimes.com/external/readwriteweb/2009/12/16/16readwriteweb-rockyou-hacker-30-of-sites-store-plain-text-13200.html. [Accessed: 27-Apr-2021]

[9] "Yahoo reportedly hacked: Is your account safe? | CBS News." July 12, 2012. Avaiable: https://www.cbsnews.com/news/yahoo-reportedly-hacked-is-your-account-safe/. [Accessed: 27-Apr-2021]

[10] "Hackers Breach 53 Universities and Dump Thousands of Personal Records Online | New York Times." October 3, 2012. Available: https://bits.blogs.nytimes.com/2012/10/03/hackers-breach-53-universities-dump thousands-of-personalrecordsonline/?mtrref=en.wikipedia.org&gwh=220D0829194985857B41F9E177E05436&gwt=pay&assetType=REGIWALL. [Accessed: 27-Apr-2021]

[11] "Anti-U.S. Hackers Infiltrate Army Servers | Information week." May 28, 2009. Available: https://www.informationweek.com/architecture/anti-us-hackers-infiltrate-army-servers/d/d-id/1079964. [Accessed: 27-Apr-2021]

[12] "Russian Hackers Amass Over a Billion Internet Passwords | New York Times." Aug. 5, 2014. Available: https://www.nytimes.com/2014/08/06/technology/russian-gang-said-to-amass-more-than-a-billion-stolen-internet-credentials.html?_r=0. [Accessed: 27-Apr-2021]

[13] "SQL Injections Continue to Embarrass Big Names | EconoTimes." May 9, 2017. Available: https://www.econotimes.com/SQL-Injections-Continue-to-Embarrass-Big-Names-690349. [Accessed: 27-Apr-2021]

[14] D. Scott and R. Sharp, "Abstracting application-level web security," Proceedings of the eleventh international conference on World Wide Web - WWW 02, 2002.

[15] J. Abirami, R. Devakunchari, and C. Valliyammai, "A top web security vulnerability SQL injection attack — Survey," 2015 Seventh International Conference on Advanced Computing (ICoAC), 2015.

[16] A. Ciampa, C. A. Visaggio, and M. D. Penta, "A heuristic-based approach for detecting SQL-injection vulnerabilities in web applications," Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems - SESS 10, 2010.

[17] L. K. Shar and H. B. K. Tan, "Mining input sanitization patterns for predicting SQL injection and cross site scripting vulnerabilities," 2012 34th International Conference on Software Engineering (ICSE), 2012.

[18] A. Joshi and V. Geetha, "SQL Injection detection using machine learning," 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014.

[19] K. Wei, M. Muthuprasanna, and S. Kothari, "Preventing SQL injection attacks in stored procedures," Australian Software Engineering Conference (ASWEC06), 2006.

[20] Y. J. Tian, Z. M. Zhao, and H. C. Zhang, "Second-order SQL Injection Attack Defense Model," Netinfo Security, 2014.

[21] A. Tajpour, M. Massrum, and M. Z. Heydari, "Comparison of SQL injection detection and prevention techniques," 2010 2nd International Conference on Education Technology and Computer, 2010.
[22] L. Qian, Z. Zhu, J. Hu, and S. Liu, "Research of SQL injection attack and prevention technology," 2015 International Conference on Estimation, Detection and Information Fusion (ICEDIF), 2015.
[23] Priyanka, and V. K. Bohat, "Detection of SQL Injection Attack and Various Prevention strategies," International Journal of Engineering and Advanced Technology (IJEAT) Vol. 2, Issue 4, April 2013.
[24] W. Rajeh and A. Abed, "A novel three-tier SQLi detection and mitigation scheme for cloud environments," 2017 International Conference on Electrical Engineering and Computer Science (ICECOS), 2017.
[25] M. Chenyu and G. Fan, "Defending SQL injection attacks based-on intention-oriented detection," 2016 11th International Conference on Computer Science & Education (ICCSE), 2016.
[26] Z. Z. Zhang, Q. Y. Wen, and Z. Zhang, "An Improved Approach for SQL Injection Vulnerabilities Detection," Applied Mechanics and Materials, vol. 263-266, pp. 3017–3020, 2012.
[27] D. Kar, S. Panigrahi, and S. Sundararajan, "SQLiDDS: SQL Injection Detection Using Query Transformation and Document Similarity," Distributed Computing and Internet Technology Lecture Notes in Computer Science, pp. 377–390, 2015.
[28] I. Lee, S. Jeong, S. Yeo, and J. Moon, "A novel method for SQL injection attack detection based on removing SQL query attribute values," Mathematical and Computer Modelling, vol. 55, no. 1-2, pp. 58–68, 2012.
[29] A. Ghafarian, "A hybrid method for detection and prevention of SQL injection attacks," 2017 Computing Conference, 2017.
[30] R. P. Karuparthi and B. Zhou, "Enhanced Approach to Detection of SQL Injection Attack," 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA), 2016.
[31] A. Kumar and S. Binu, "Proposed Method for SQL Injection Detection and its Prevention," International Journal of Engineering & Technology, vol. 7, no. 2.6, p. 213, 2018.
[32] L. Ntagwabira and S. L. Kang, "Use of Query tokenization to detect and prevent SQL injection attacks," 2010 3rd International Conference on Computer Science and Information Technology, 2010.
[33] R. Rawat and S. K. Shrivastav, "SQL injection attack Detection using SVM," International Journal of Computer Applications, vol. 42, no. 13, pp. 1–4, 2012.
[34] K. Kamtuo and C. Soomlek, "Machine Learning for SQL injection prevention on server-side scripting," 2016 International Computer Science and Engineering Conference (ICSEC), 2016.
[35] Z. Chen, M. Guo, and L. Zhou, "Research on SQL injection detection technology based on SVM," MATEC Web of Conferences, vol. 173, p. 01004, 2018.
[36] I. H. Witten and E. Frank, Data Mining. Burlington: Elsevier, 2005.
[37] J. Lei and X.-Q. Shen, "A new method for edge detection based on support vector classification," Proceedings. International Conference on Machine Learning and Cybernetics.
[38] M. Lodeiro-Santiago, C. Caballero-Gil, and P. Caballero-Gil, "Collaborative SQL-injections detection system with machine learning," Proceedings of the 1st International Conference on Internet of Things and Machine Learning, 2017.
[39] D. Das, U. Sharma, and D. K. Bhattacharyya, "Rule based Detection of SQL Injection Attack," International Journal of Computer Applications, vol. 43, no. 19, pp. 15–24, 2012.
[40] Y. Wang, and Z. Li, "SQL Injection Detection with Composite Kernel in Support Vector Machine," International Journal of Security and its Applications 6(2), pp. 191–196, April 2012.
[41] C. Ping, "A second-order SQL injection detection method," 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2017.
[42] B. D. Priyaa and M. I. Devi, "Hybrid SQL injection detection system," 2016 3rd International Conference on Advanced Computing and Communication Systems (ICACCS), 2016.
[43] P. Li, L. Liu, J. Xu, H. Yang, L. Yuan, C. Guo, and X. Ji, "Application of Hidden Markov Model in SQL Injection Detection," 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), 2017.
[44] K. Kemalis and T. Tzouramanis, "Sql-Ids," Proceedings of the 2008 ACM symposium on Applied computing - SAC 08, 2008.
[45] V. Shanmughaneethi, "Detection of SQL Injection Attack in Web Applications using Web Services," IOSR Journal of Computer Engineering, vol. 1, no. 5, pp. 13–20, 2012.
[46] P. Chhabra, R. Wadhvani, and S. Shukla, "Spam Filtering using Support Vector Machine," International Journal of Computer and Communication Technology, pp. 256–261, 2010.
[47] S. R. Pakize, and A. Gandomi, "Comparative study of classification algorithms based on MapReduce model," International Journal of Innovative Research in Advanced Engineering (IJIRAE), 1(7), pp. 251-254, 2014.
[48] S. Archana, and K. Elangovan, "Survey of classification techniques in data mining," International Journal of Computer Science and Mobile Applications, 2(2), pp. 65-71, 2014.

[49] H. Maldonado, L. Leija, and A. Vera, "Selecting a computational classifier to develop a clinical decision support system (CDSS)," 2015 12th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE), 2015.

[50] S. S. Nikam, "A comparative study of classification techniques in data mining algorithms," Oriental journal of computer science & technology, 8(1), pp. 13-19, 2015.

[51] M. Y. Asr, M. M. Ettefagh, R. Hassannejad, and S. N. Razavi, "Diagnosis of combined faults in Rotary Machinery by Non-Naive Bayesian approach," Mechanical Systems and Signal Processing, vol. 85, pp. 56–70, 2017.

[52] A. S. Subaira and P. Anitha, "Efficient classification mechanism for network intrusion detection system based on data mining techniques: A survey," 2014 IEEE 8th International Conference on Intelligent Systems and Control (ISCO), 2014.

[53] R. Senthilkumar and R. K. Gnanamurthy, "Performance improvement in classification rate of appearance based statistical face recognition methods using SVM classifier," 2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS), 2017.

[54] T.-T. Dai and Y.-S. Dong, "Introduction of SVM Related Theory and Its Application Research," 2020 3rd International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE), 2020.

[55] S. Chen, J.-Q. Wang and H.-Y. Zhang, "A hybrid PSO-SVM model based on clustering algorithm for short-term atmospheric pollutant concentration forecasting," Technological Forecasting and Social Change, vol. 146, pp. 41–54, 2019.

[56] O. A. ., "Comparative Study Of Classification Algorithm For Text Based Categorization," International Journal of Research in Engineering and Technology, vol. 05, no. 02, pp. 217–220, 2016.

[57] B. K. Bhavitha, A. P. Rodrigues, and N. N. Chiplunkar, "Comparative study of machine learning techniques in sentimental analysis," 2017 International Conference on Inventive Communication and Computational Technologies (ICICCT), 2017.

[58] I. A. A. Amra and A. Y. A. Maghari, "Students performance prediction using KNN and Naïve Bayesian," 2017 8th International Conference on Information Technology (ICIT), 2017.

[59] Y. Shi, Y. Sagduyu, and A. Grushin, "How to steal a machine learning classifier with deep learning," 2017 IEEE International Symposium on Technologies for Homeland Security (HST), 2017.

[60] A. Korotcov, V. Tkachenko, D. P. Russo, and S. Ekins, "Comparison of Deep Learning With Multiple Machine Learning Methods and Metrics Using Diverse Drug Discovery Data Sets," Molecular Pharmaceutics, vol. 14, no. 12, pp. 4462–4475, 2017.

[61] W. C. F. Mariel, S. Mariyah, and S. Pramana, "Sentiment analysis: a comparison of deep learning neural network algorithm with SVM and naïve Bayes for Indonesian text," Journal of Physics: Conference Series, vol. 971, p. 012049, 2018.

[62] R. Moraes, J. F. Valiati, and W. P. G. Neto, "Document-level sentiment classification: An empirical comparison between SVM and ANN," Expert Systems with Applications, vol. 40, no. 2, pp. 621–633, 2013.

[63] M.-Y. Day and Y.-D. Lin, "Deep Learning for Sentiment Analysis on Google Play Consumer Review," 2017 IEEE International Conference on Information Reuse and Integration (IRI), 2017.

[64] A. Koutsoukas, K. J. Monaghan, X. Li, and J. Huan, "Deep-learning: investigating deep neural networks hyper-parameters and comparison of performance to shallow methods for modeling bioactivity data," Journal of Cheminformatics, vol. 9, no. 1, 2017.

[65] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "DeepTox: Toxicity prediction using deep learning," Toxicology Letters, vol. 280, 2017.

[66] W. G. J. Halfond and A. Orso, "Preventing SQL injection attacks using AMNESIA," Proceeding of the 28th international conference on Software engineering - ICSE 06, 2006.

[67] A. Sadeghian, M. Zamani, and A. A. Manaf, "A Taxonomy of SQL Injection Detection and Prevention Techniques," 2013 International Conference on Informatics and Creative Multimedia, 2013.

[68] N. Salih and A. Samad, "Protection Web Applications using Real-Time Technique to Detect Structured Query Language Injection Attacks," International Journal of Computer Applications, vol. 149, no. 6, pp. 26–32, 2016.

[69] D. A. Kindy, and K. P. Al-Sakib, "A Survey on SQL Injection Vulnerabilities, Attacks, and Prevention Techniques," IEEE 15th International Symposium on Consumer Electronics 2011.

[70] M. Howard and D. L. Blane. "Writing Secure Code," Vol. II. Microsoft Press, Redmond, Washington, 2003.

[71] S. San-Tsai, W. H. Ting, L. Stephen , and L. Sheung, "Classification of SQL Injection Attacks," University of British Columbia, Term Project, 2007.

[72] J. P. Singh, "Analysis of SQL Injection Detection Techniques," Theoretical and Applied Informatics, Vol. 28, No. 1–2, 2017.

[73] C. Anley, "Advanced SQL Injection in SQL Server Applications, white paper Next Generation Security Software," 2002.
[74] D. Litchfield, "Director of Security Architecture. Web application dissembly with odbc error message," a report.
[75] P. Kaur, and K. P. Kour, "SQL injection: Study and augmentation," 2015 International Conference on Signal Processing, Computing and Control (ISPCC), 2015.
[76] K. Kemalis and T. Tzouramanis, "Sql-Ids," Proceedings of the 2008 ACM symposium on Applied computing - SAC 08, 2008.
[77] X. Ping-Chen, "SQL injection attack and guard technical research," Procedia Engineering, vol. 15, pp. 4131–4135, 2011.
[78] C. Victor, "Advanced SQL injection," OWASP Foundation, April 2005.
[79] L. K. Shar, and H. B. K. Tan, "Defeating SQL Injection," Computer, 46(3), pp. 69–77, March 2013.
[80] D. Zhang, "Inconsistencies in big data," 2013 IEEE 12th International Conference on Cognitive Informatics and Cognitive Computing, 2013.
[81] E. Cabrio, J. Cojan, and F. Gandon, "Mind the Cultural Gap: Bridging Language-Specific DBpedia Chapters for Question Answering," Towards the Multilingual Semantic Web, pp. 137–154, 2014.
[82] G. Bingham, B. Yip, M. Ferguson, and C. Nansalo, "MORPH-II: Inconsistencies and Cleaning," University of North Carolina Wilmington NSF REU, 2017.
[83] M. C. Rendleman, J. M. Buatti, T. A. Braun, B. J. Smith, C. Nwakama, R. R. Beichel, B. Brown, and T. L. Casavant, "Machine learning with the TCGA-HNSC dataset: improving usability by addressing inconsistency, sparsity, and high-dimensionality," BMC Bioinformatics, vol. 20, no. 1, 2019.
[84] T. Martins, and J. C. dos Reis, "Mechanism for inconsistency correction in the DBPedia Live," Technical report, Universidade Estadual de Campinas-UNICAMP, 2019.
[85] P. Pernot, and F. Cailliez, "A critical review of statistical calibration/prediction models handling data inconsistency and model inadequacy," AIChE Journal, 63(10), pp. 4642–4665, 2017.
[86] S. Sathya, and A. Rajesh, "Enhanced Hybrid Data Preprocessing Technique for Eliminating Inconsistencies in the Diabetic Dataset to Improve Mining Results," Journal of Computational and Theoretical Nanoscience, Vol. 15, No. 6–7, pp. 1999–2002(4), June 2018.
[87] R. Sridevi, and Dr.S. Priyaa, "An Ensemble approach on Missing Value Handling in Hepatitis Disease Dataset," International Journal of Computer Applications, Vol. 130, No.17, November 2015.
[88] M. Juhola, and J. Laurikkala, "Missing values: how many can they be to preserve classification reliability," Artificial Intelligence Review, 40(3), pp. 231–245, 2011.
[89] J. Watada, C. Shi, Y. Yabuuchi, R. Yusof, and Z. Sahri, "A Rough Set Approach to Data Imputation and Its Application to a Dissolved Gas Analysis Dataset," Third International Conference on Computing Measurement Control and Sensor Network (CMCSN), 2016.
[90] J. Kaiser, "Dealing with missing values in data," Journal of systems integration 5(1), pp. 42–51, 2014.
[91] M. Priyakshi, G. A. Choudhury, and T. Dey, "An Effective Method to Estimate Missing Value for Heterogonous Dataset." International Journal of Knowledge Based Computer Systems, 6(2), pp. 8–22, December 2018.
[92] Z. Maciej, "Service-Oriented Medical System for SupportingDecisions with Missing and Imbalanced Data," IEEE Journal of Biomedical and Health Informatics, Vol. 18, No. 5, SEPTEMBER 2014.
[93] I. B. Mohamad and D. Usman, "Standardization and Its Effects on K-Means Clustering Algorithm," Research Journal of Applied Sciences, Engineering and Technology, vol. 6, no. 17, pp. 3299–3303, 2013.
[94] N. Bigdely-Shamlo, T. Mullen, C. Kothe, K.-M. Su, and K. A. Robbins, "The PREP pipeline: standardized preprocessing for large-scale EEG analysis," Frontiers in Neuroinformatics, 9, p. 16, 2015.
[95] C. Cabanes, A. Grouazel, K. V. Schuckmann, M. Hamon, V. Turpin, C. Coatanoan, F. Paris, S. Guinehut, C. Boone, N. Ferry, C. de Boyer Mont´egut, T. Carval, G. Reverdin, S. Pouliquen, and P. Y. Le Traon, "The CORA dataset: validation and diagnostics of in situ ocean temperature and salinity measurements," Ocean Science, 9(1), pp.1–18, 2013.
[96] J. F. Girres, and G. Touya, "Quality Assessment of the French OpenStreetMap Dataset," Transactions in GIS, 14(4), pp. 435–459, 2010.
[97] A. Wosiak, and D. Zakrzewska, "Integrating Correlation-Based Feature Selection and Clustering for Improved Cardiovascular Disease Diagnosis," Complexity, pp. 1–11, 2018.
[98] M. Doshi and S. K. Chaturvedi, "Correlation Based Feature Selection (CFS) Technique to Predict Student Perfromance," International journal of Computer Networks & Communications, vol. 6, no. 3, pp. 197–206, 2014.
[99] A. H. Mark, "Correlation-based Feature Selection for Machine Learning," Department of Computer Science, The University of Waikato.
[100] C. Jin, T. Ma, R. Hou, M. Tang, Y. Tian, A. Al-Dhelaan, and M. Al-Rodhaan, "Chi-square Statistics Feature Selection Based on Term Frequency and Distribution for Text Categorization," IETE Journal of Research, vol. 61, no. 4, pp. 351–362, 2015.

[101] B. S. Otong, H. M. Kusnadi, and N. Oky Dwi, "Feature selection based on chi square in artificial neural network to predict the accuracy of student study period," International Journal of Civil Engineering and Technology (IJCIET), Vol. 8, Issue 8, pp. 731–739, August 2017.

[102] C. Selvaraj, N. Bhalaji, and K. S. Kumar, "Empirical study of feature selection methods over classification algorithms," International Journal of Intelligent Systems Technologies and Applications, vol. 17, no. 1/2, p. 98, 2018.

[103] S. Deysarakar and S. Goswami, "Empirical Study on Filter based Feature Selection Methods for Text Classification," International Journal of Computer Applications, vol. 81, no. 6, pp. 38–43, 2013.

[104] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," Journal of machine learning research, Vol. 15, No. 1, pp. 1929–1958.

[105] R. C. Sharma, K. Hara, and H. Hirayama, "A Machine Learning and Cross-Validation Approach for the Discrimination of Vegetation Physiognomic Types Using Satellite Based Multispectral and Multitemporal Data," Scientifica, vol. 2017, pp. 1–8, 2017.

[106] R. V. Guido and B. D. Jelke, "Interactively Testing Remote Servers Using the Python Programming Language", CWI Quarterly, Volume 4, Amsterdam, pp. 283–303, Issue 4 (December 1991).

[107] François C. and others, Keras, 2015. Available: https://github.com/keras-team/keras

[108] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and others, "Scikit-learn: Machine learning in Python." Journal of machine learning research, 12(Oct), pp. 2825–2830, 2011.

[109] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, and others, "Tensorflow: A system for large-scale machine learning," 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 265–283, 2016.

[110] O. K. Sahingoz, E. Buber, O. Demir, and B. Diri, "Machine learning based phishing detection from URLs," Expert Syst. Appl., vol. 117, no. January 2019, pp. 345–357, 2019, doi: 10.1016/j.eswa.2018.09.029.

[111] O. Kasim, "Automatic detection of phishing pages with event-based request processing, deep-hybrid feature extraction and light gradient boosted machine model," Telecommunication Systems, 2021, pp. 1-13.

[112] P. Yang, G. Zhao, and P. Zeng, "Phishing website detection based on multidimensional features driven by deep learning," IEEE Access, vol. 7, no. c, pp. 15196–15209, 2019, doi: 10.1109/ACCESS.2019.2892066.

[113] L. Lakshmi, M. P. Reddy, C. Santhaiah, and U. J. Reddy, "Smart Phishing Detection in Web Pages using Supervised Deep Learning Classification and Optimization Technique ADAM," Wireless Personal Communications, 2021, pp. 1-16.

[114] A. Basit, M. Zafar, X. Liu, A. R. Javed, Z. Jalil, and K. Kifayat, "A comprehensive survey of AI-enabled phishing attacks detection techniques," Telecommun. Syst., vol. 76, no. 1, pp. 139–154, 2021, doi: 10.1007/s11235-020-00733-2.