Evaluation of SQL injection (SQLi) attack detection strategies in web applications using machine learning

Santiago Taborda Echeverri

Undergraduate project for the degree of Telecommunications Engineer

Advisor
Magister Jaime Alberto Vergara Tejada

Universidad de Antioquia
Engineering Faculty
Telecommunications Engineering
Medellín
2024

Citation	Taborda Echeverri [1]
Reference IEEE Style (2020)	[1] S. Taborda Echeverri, “Evaluation of SQL injection (SQLi) attack detection strategies in web applications using machine learning”, Industry Semester, Telecommunications Engineering, Universidad de Antioquia, Medellín, 2024.

SOC Manager AizoOn Technology Consulting: Jhonny Alexander Triana Maldonado
Cyber Senior Data Scientist AizoOn Technology Consulting: Daniele Ucci
Cyber Security Specialist AizoOn Technology Consulting: Ivan Russo

UNIVERSIDAD DE ANTIOQUIA
Vicerrectoría de Docencia

Sistema de Bibliotecas

Centro de Documentación de Ingeniería (CENDOI)

Institutional Repository: http://bibliotecadigital.udea.edu.co

Universidad de Antioquia - www.udea.edu.co

Rector: John Jairo Arboleda Céspedes.

Dean/Director: Julio César Saldarriaga Molina.

Department Director: Eduard Emiro Rodríguez Ramírez.

Acknowledgments

To my mom, my hero...

Contents

List of Figures

List of Tables

1 Abstract

This work evaluates strategies for detecting SQL injection attacks based on artificial intelligence to generate a recommendation that allows the improvement of the web application firewall of AizoOn Technology Consulting (Mithril). To achieve this, detection techniques known as Naïve Bayes, logistic regression, random forests, and one-class support vector machines were selected based on their relevance and effectiveness demonstrated in the scientific literature and the company's expressed interests. These techniques were implemented by structuring a hybrid database integrating public data from the "SQL Injection Dataset" available on Kaggle with data processed by Mithril. This process involved data analysis, preprocessing, and conditioning. Data integration proved useful for implementing the machine learning models. Subsequently, hyperparameter tuning was performed to improve the models' performance, identifying the best configurations for each of them, thus increasing detection capabilities and minimizing false positives. The evaluation and benchmarking of the models were conducted using performance metrics such as accuracy, precision, recall, and F1-Score. Finally, the results led to the recommendation of implementing the logistic regression model in Mithril, as it achieved the best performance with accuracy and F1-Score of 99.45%

2 Introduction

The constant use of the internet and computer services has led them to play an increasingly important role in our daily lives. According to the Our World in Data report [1], in the last three months, 4.7 billion people worldwide used the internet. These services offer various advantages and streamline tasks for users in different sectors. Gallup (an American analytics and advisory company) reported in November 2023 that 79% of the U.S. workforce worked remotely or with a hybrid model [2], directly implying the use of information technologies.

This growth brings with it some vulnerabilities to cyberattacks. Specifically, organizations face cyberattacks that constantly threaten their systems, including SQL injection (SQLi). SQLi allows attackers to insert malicious SQL code into input input fields, which can compromise databases and extract sensitive information.

SQLi attack ranks third among the top 10 security risks in web applications, according to the 2021 report by the Open Web Application Security Project (OWASP) [3]. This report reveals that 94% of evaluated web applications had vulnerabilities to SQLi attacks [4]. Similarly, the Common Weakness Enumeration (CWE [5]) places SQLi at the third position in the list of the 25 most dangerous software vulnerabilities of 2023 [6].

Machine learning has demonstrated effective performance in detecting cyber attacks, and as a result, the interest within the scientific and industrial communities for its application. This is evident in the study conducted by Hanif et al. (2021) [7], where 74.4% of the 90 reviewed published articles make use of computational intelligence techniques for cyber vulnerability detection.

Given the relevance of both the attack and the use of machine learning to detect the attack nowadays, it is expected that companies involved in the information technology (IT)

field will focus on implementations aimed at mitigating SQLi attacks using computational intelligence techniques.

AizoOn Technology Consulting is a technology consulting company based in Turin, Italy [8]. The organization's mission is to support its clients throughout the digital era, offering technological expertise and cutting-edge innovation [9]. The company has two technology divisions covering the spectrum of data acquisition and transformation, and security and strategic and/or operational risk management. These divisions correspond to the cybersecurity division and the innovation and digital engineering division. The former focuses on increasing organizations' resilience against cyber threats originating from the digital ecosystem, while the second focuses on creating value by transforming data into knowledge for decision-making.

Within the cybersecurity division, the Security Operations Center (SOC) acts as a Managed Security Services Provider (MSSP). The SOC implements customized cybersecurity solutions to fit each client's specific needs, combining incident management and response, threat hunting and intelligence, and digital forensic analysis [10].

This work focuses on the evaluation of SQLi attack detection strategies, developed within the AizoOn SOC. It starts with a literature review to identify SQLi attack detection techniques against web applications, focusing on those that make use of machine learning. Next, a database is structured by integrating data processed by AizoOn Technology Consulting's web application firewall with the publicly accessible "SQL Injection Dataset" available on the Kaggle platform [11], subjecting the resulting data to cleaning and conditioning. Subsequently, machine learning techniques are applied making use of the processed data to detect SQLi attacks. The performance of the implemented techniques is evaluated using established performance metrics such as accuracy, precision, recall, and F1-Score. Finally, the results are analyzed to compare the different machine learning techniques, identifying the one with the best performance to generate a recommendation for its future implementation.

3 Objectives

3.1 General objective

Evaluate strategies for detecting SQL injection (SQLi) attacks based on machine learning and provide a recommendation to enhance the security of the web application firewall of the company AizoOn Technology Consulting.

3.2 Specific objectives

- Select techniques for detecting SQL injection attacks against web applications, particularly the object of this project, those that make use of machine learning techniques.
- Structure a database containing both public datasets and data generated from the web application firewall of the company AizoOn Technology Consulting.
- Implement and train each of the previously chosen SQL injection attack detection techniques that use machine learning.
- Evaluate and compare the implemented techniques based on defined performance metrics and generate a recommendation according to the results obtained for its implementation in the web application firewall of the company AizoOn Technology Consulting.

4 Theoretical background and related work

Structured Query Language (SQL) is one of the most common standard languages for creating, interacting, and maintaining relational databases and comparable systems. User interaction (insert, delete, and update) can be performed by querying the data using the data manipulation component of the SQL language. In contrast, the administrator can perform maintenance and access control using the data control component of the SQL language [12].

Most web applications use databases managed by database management systems (DBMSs) that employ or are capable of understanding SQL queries, such as MySQL, PostgreSQL, or Oracle [13]. However, the existence of web applications that make use of non-relational databases called NoSQL (Not Only SQL) [14] should be recognized. NoSQL databases are suitable for scenarios where it is essential to handle large amounts of data like the Internet, multimedia, and social media applications, these databases do not employ the principles of relational database management systems (RDBMSs) [15].

SQL injection (SQLi) is an attack that consists of the insertion (injection) of malicious code into SQL queries from the client application inputs. According to the Open Web Application Security Project (OWASP [3]), the SQL commands are injected into data-plane input to affect the execution of predefined SQL commands, impacting confidentiality, authentication, authorization, and data integrity [16]. A successful SQL injection attack can have serious consequences, such as unauthorized reading and exposure of sensitive data, which in web applications can be user names, passwords, and even personal information [17], modification of database data (insert, delete, or update), execution of actions as administrator and in some cases issue commands to the operating system [18].

Incorrect validation, in other words, the lack of removal or quoting of SQL syntax in controlled user input, allows a SQL query to be interpreted as SQL code [19], which contributes to incorporating SQL injection attack vulnerabilities and leading to the success of the attack. This vulnerability is most likely to be found in PHP (Hypertext Preprocessor) and ASP (Active Server Pages) applications [17]; however, any site or product package with a minimal user database is susceptible to attack attempt of this type [20].

The Figure 4.1 shows the general flow of an SQL injection attack. When the user input does not have the appropriate validation, as mentioned above, attackers can inject packets containing SQL commands to be interpreted in the database as SQL code through POST and GET requests [21], reaching the target of the attack.

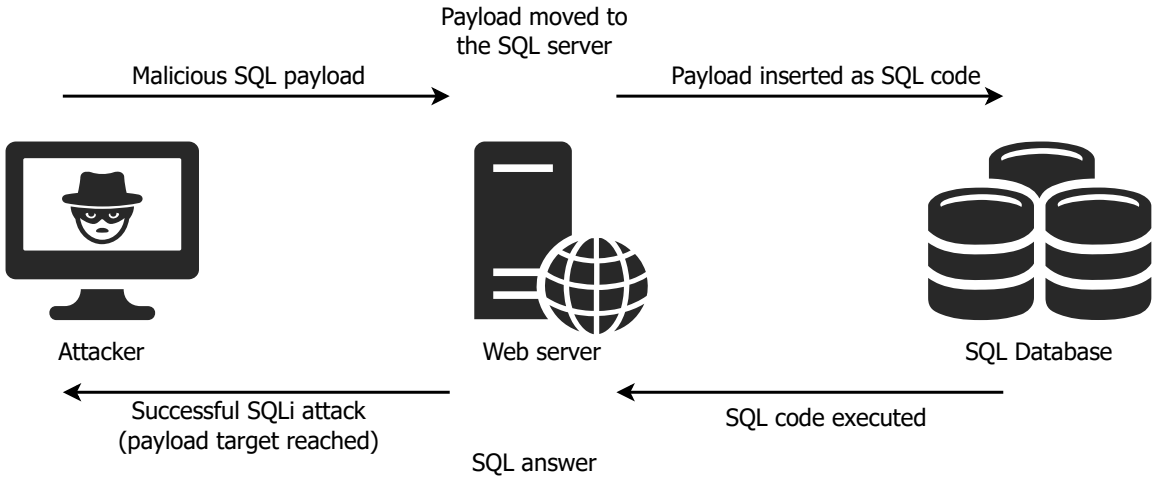


Figure 4.1: General mechanism of a SQL injection (SQLi) attack.

Proper knowledge of the general behavior of a SQL injection attack is essential for preventing the attack. It can be classified based on different factors from the literature, and generally speaking, an SQLi attack is mainly classified into the categories shown in the Figure 4.2 [22].

In-band SQLi: This SQL injection attack is the most common and easiest to carry out. The attacker uses the same communication channel to send the attack and receive the answer.

- **Error-based SQLi:** Based on error messages issued by the target database server, attackers use invalid inputs in queries through the user interface to intentionally

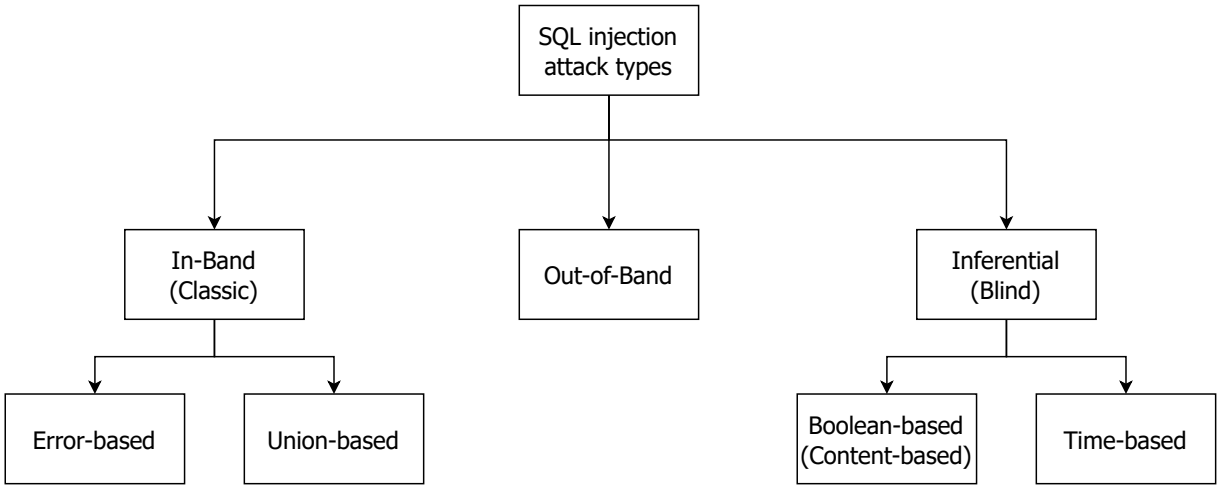


Figure 4.2: Types of SQL injection (SQLi) attacks.

cause the database to generate errors. The error messages can reveal information about the target system, including details such as its structure, version, and operating system, or even return complete query results [23], sometimes attackers can manipulate an entire database by applying this type of attack [24].

- **Union-based SQLi:** The UNION operator of the SQL language allows combining the result of two or more queries into a single result set, eliminating duplicates. Attackers exploit this operator to combine a malicious query with an original query, usually SELECT statements returned as part of the HTTP response. Allowing unauthorized access to column values from other tables that should not be accessed [19].

Out-of-Band SQLi: In this case, the database cannot respond directly to the web server when an attacker is trying to retrieve information sending queries. However, it is possible to take advantage of features enabled on the target site, such as HTTP, DNS, or FTP (commonly supported by popular SQL servers) [25], forcing the database to transmit the responses to a remote endpoint controlled by the attacker.

Inferential SQLi: It is characterized by no visible response on the web page to queries (also known as Blind SQLi) [17], even when no SQL query results are returned, an attacker can probe the server and observe its behavior to obtain information [25]. It requires the attacker to generate queries to the database and interpret the responses obtained, usually taking advantage of generic errors such as "Syntax Error" [24]. This type of SQL injection

attack is slower to exploit due to the lack of direct information about the web page.

- **Boolean-based SQLi:** Attackers manipulate SQL queries to force the web application to return a TRUE or FALSE (response from the web application, not from the database) [26]. The methodology consists of injecting a fake payload and examining the response, then performing an analysis with a true payload [17]. The vulnerability will depend on divergent behavior in the responses to different payloads; attackers could exploit this variation to deduce sensitive data.

- **Time-based SQLi:** In this strategy, time delays are conveniently incorporated into the injected SQL code in order to measure and control the response times of the application to indirectly deduce information about the database or extract sensitive data [26]. The attackers exploit the SLEEP query, which allows them to "pause" the database for a defined amount of time. The response time will allow the attacker to determine whether the query result is TRUE or FALSE.

SQLi attack prevention minimizes the attack surface by avoiding possible dynamic SQL statements and generates errors by default from the database [25]. This is possible by considering appropriate security measures from the development stage, such as exhaustive code cleaning, appropriate validation of user input, and restriction of automatic database responses to errors.

The ability to apply prevention activities may be limited in practical situations, making it necessary to detect SQLi attacks by implementing additional actions during system execution to protect the system's operational continuity and security. Traditional methodologies are based on predefined rules, such as filters and regular expressions, including black-box and white-box testing, pattern matching, sequence comparison, and other measures.

Traditional methodologies are widely used; however, they face scalability issues and may not effectively detect certain complex SQLi attack patterns [25]. Machine learning techniques, on the other hand, are emerging as practical tools for SQL injection detection in several types of applications such as enterprise systems, cloud-based systems, and websites [26], becoming a promising alternative due to their ability to detect behaviors that may bypass traditional detection methods.

In the Figure 4.3 it can be evidenced that in general terms, the use of computational intelligence techniques has significantly surpassed, until 2020 according to the study conducted by Hanif et al, the utilization of traditional approaches in the detection of vulnerabilities [7].

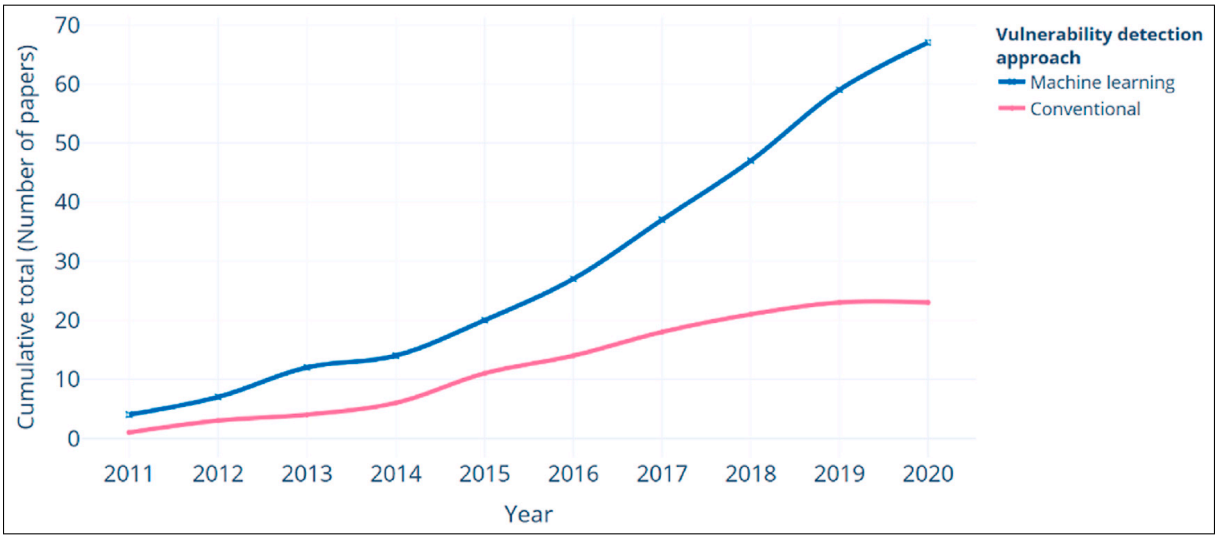


Figure 4.3: Cumulative total of different categories of vulnerability detection approaches from the year 2011 until 2020 [7].

The effectiveness of the use of machine learning techniques (algorithms) depends on several factors, such as the problem to be solved, the data pre-processing for training, and the origin of the data itself. Table 4.1 presents various studies related to the research topic addressed in this work.

Year	Title	Detection method	Dataset
2023	SQL injection attack detection in network flow data [27]	Logistic Regression (LR), LSVC, Perceptron+SGD, Random forest (RF), K-nearest neighbors (KNN)	Two Netflow V5 datasets public and available, collected using DOROTHEA tool

Continued on the next page

2022	SQL Injection Attack Detection by Machine Learning Classifier [28]	Logistic Regression (LR), AdaBoost, Naïve Bayes (NB), XGBoost, Random Forest	Kaggle SQL injection data set
2022	Deep Neural Network-Based SQL Injection Detection Method [29]	Deep Neural Network	Kaggle SQL Injection Dataset
2022	Multi-Phase Algorithmic Framework to Prevent SQL Injection Attacks using Improved Machine learning and Deep learning to Enhance Database security in Real-time [30]	Convolutional Neural Network (CNN) and Naïve Bayes (NB)	Private dataset created using different databases such as MySQL, Oracle, MS.SQL and Postgres
2021	SQL Injection Detection Using Machine Learning Techniques [31]	Logistic Regression (LR), Gradient boosting (GB), Decision tree (DT), Random forest (RF), Support vector machines (SVM), Neural networks (NN)	Kaggle SQL injection data set and All-Attacks dataset that use libingection
2021	SQL Injection Attack Detection and Prevention Techniques Using Deep Learning [32]	Convolutional Neural Network (CNN) and a Multilayer Perceptron (MLP)	Private dataset with records from internet request

Continued on the next page

2020	Machine Learning based Intrusion Detection System for Web-Based Attacks [33]	J48 (decision tree), One rule (OneR), Naïve Bayes (NB)	Cleaned and labeled CSIC HTTP 2010 dataset
2019	LSTM-Based SQL Injection Detection Method for Intelligent Transportation System [34]	Long short-term memory based (LSTM-based) and comparing with Support vector machine (SVM), K-nearest neighbors (KNN), Decision Tree, Naïve Bayes (NB), Random forest (RF), Convolutional neural network (CNN), Recurrent neural network (RNN) and Multilayer perceptron (MLP)	Private dataset, made for the proposed work and collected from different sources such as Github, exploit-db, among others
2019	Predicting Web Vulnerabilities in Web Applications Based on Machine Learning [35]	Random forest, J48 (decision tree), Naïve Bayes (NB)	PHP Security vulnerability dataset
2018	Web Application Attacks Detection Using Machine Learning Techniques [36]	Support vector machine (SVM), K-nearest neighbors (KNN), Random forest	PKDD 2007 challenge dataset, CSIC2010 dataset, and private dataset called DRUPAL based on the public website of the Universidad Católica de Uruguay.

Table 4.1: SQL injection detection strategies using machine learning

Machine learning algorithms can be categorized into supervised and unsupervised learning, as shown in the Figure 4.4.

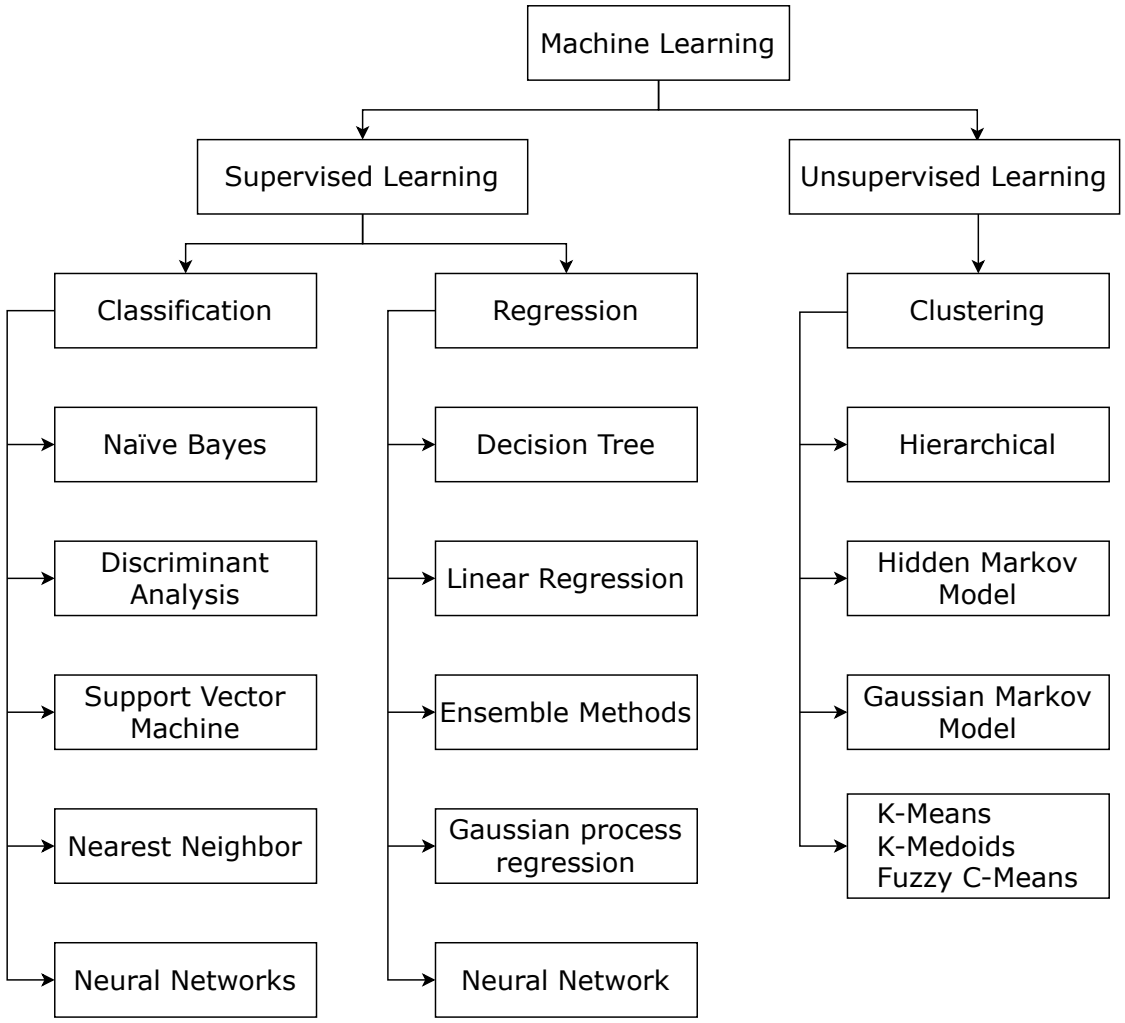


Figure 4.4: Machine learning categories and relevant algorithms [24] [37].

Supervised learning is a machine learning technique that creates a function from labeled training data capable of assigning to a specific input an output obtained from the learning process [38] [39]. Supervised learning can be oriented to regression, which usually consists of fitting the data with a numerical value (usually decimal type) as the function's output. On the other hand, classification consists of separating the data with a class label as the function's output. Although the class label is numerical in most cases, it represents a distinction [40].

Unlike supervised learning, unsupervised learning is a machine learning technique that analyzes unlabeled data sets. In the learning process, there are no input examples with known labels [38]. A classic example is clustering, a technique used to identify groups within certain data [39]. This type of learning is widely used to extract features, identify significant trends and structures, identify groupings, and facilitate exploratory data analysis [40].

From the literature review presented in Table 4.1, it is clear that there is interest in using supervised algorithms for detecting SQL injection attacks. However, it should be noted that some studies implement solutions combining both supervised and unsupervised learning [41].

Naïve Bayes: is a computational intelligence technique based on probability. It is derived from Bayes' theorem and is distinguished by assuming that all features of a data set are independent of each other [42]. Although real-world data features are often related, this assumption allows for reduced computational complexity [43], enabling the model to efficiently calculate the probability that a given data point belongs to a specific class based on the observed features, and then select the class with the highest probability [44] [45]. There are three main variants of the Naïve Bayes model:

- **Gaussian Naïve Bayes:** has demonstrated good performance with data that follows a Gaussian distribution, specifically continuous numerical data [46].
- **Multinomial Naïve Bayes:** is often used in problems involving discrete data and is commonly implemented with text data for natural language processing tasks [46].
- **Bernoulli Naïve Bayes:** designed for data that follows a Bernoulli distribution, where each variable has binary data [47].

Logistic Regression: is a technique used to solve binary classification problems by predicting the probability that a given data point belongs to one of two categories [48]. It differs from linear regression in that it applies a logistic function to constrain the output to a value between 0 and 1 [49]. The main advantage of this technique lies in its ability to be used for both classification and probability estimation. Although it is usually

employed for binary classification, it can also be extended to multi-class classification through techniques such as multinomial logistic regression [50].

Random Forest Classifier: consists of a combination of tree-type classifiers, where each classifier is generated using a unique random vector independently sampled from the input data. When classifying a new input, each tree casts a unit vote for a class, and the most popular class among all the trees determines the final classification [51].

One-Class Support Vector Machines (OCSVM): initially proposed by Müller and colleagues [52], its main objective is to construct a hyperplane that contains the data from the majority class (referred to as target data) and excludes data from the remaining class (considered as outliers) [53]. In other words, it aims to identify instances that significantly deviate from the norm.

5 Methodology

5.1 Selection of SQLi attack detection techniques

The literature review focused on SQLi attack detection techniques using machine learning techniques was developed in chapter 4, and the state of the art shown in Table 4.1 was obtained. From this state-of-the-art, the following computational intelligence techniques were selected:

- Naïve Bayes
  - Gaussian Naïve Bayes (GaussianNB)
  - Multinomial Naïve Bayes (MultinomialNB)
- Logistic Regression

In the work conducted by Roy, P. et al [28], five SQLi attack detection models were implemented and evaluated, with Naïve Bayes emerging as the most successful, achieving an accuracy of 98.33%. Meanwhile, Ashlam, A. et al [30] presented a multi-phase algorithmic framework designed to detect and prevent SQL injections (SQLi) in real-time, including controlled laboratory experiments and real-world deployments in a university computer network and a commercial bank, where Naïve Bayes achieved an accuracy of 95%.

The logistic regression model for classification has also been implemented in various works present in the state of the art. Specifically, the work conducted by Crespo-Martínez, I. S. et al [27] shows the training and testing of five different models based on supervised learning, achieving a detection rate higher than 97% with a false alarm rate lower than

0.07% for the logistic regression-based model. Similarly, Hosam, E. et al [31] evaluated six different machine learning models and concluded that the logistic regression is the most effective for generalizing unseen data during the model training process, with an accuracy of 87.3%.

In addition to the previously mentioned techniques, the company requested the inclusion of the following within the selected group:

- Random Forest
- One-Class Support Vector Machines (OCSVM)

5.2 Database structuring

The objective was to use a public database to structure a database including certain data from the web application firewall of AizoOn Technology Consulting (commercially known as Mithril).

5.2.1 Data Sources

Mithril Data: The data provided by the company from Mithril consists exclusively of SQLi attack payloads without network traffic information. AizoOn has various clients for whom it provides Mithril management and monitoring services as a product. The data was extracted from an implementation for a client that conducts periodic specific SQLi tests by its offensive security team, corresponding to January 2024.

Public Database: Considering the nature of Mithril's data and aiming to integrate it into a public database, the "SQL Injection Dataset" publicly available on the Kaggle platform [11] was selected. This dataset includes payloads considered acceptable in normal user behavior as well as SQLi payloads and has been used in various works present in the state of the art (Table 4.1).

The Figure 5.1 shows the number of data points classified by type, for both Mithril and Kaggle data.

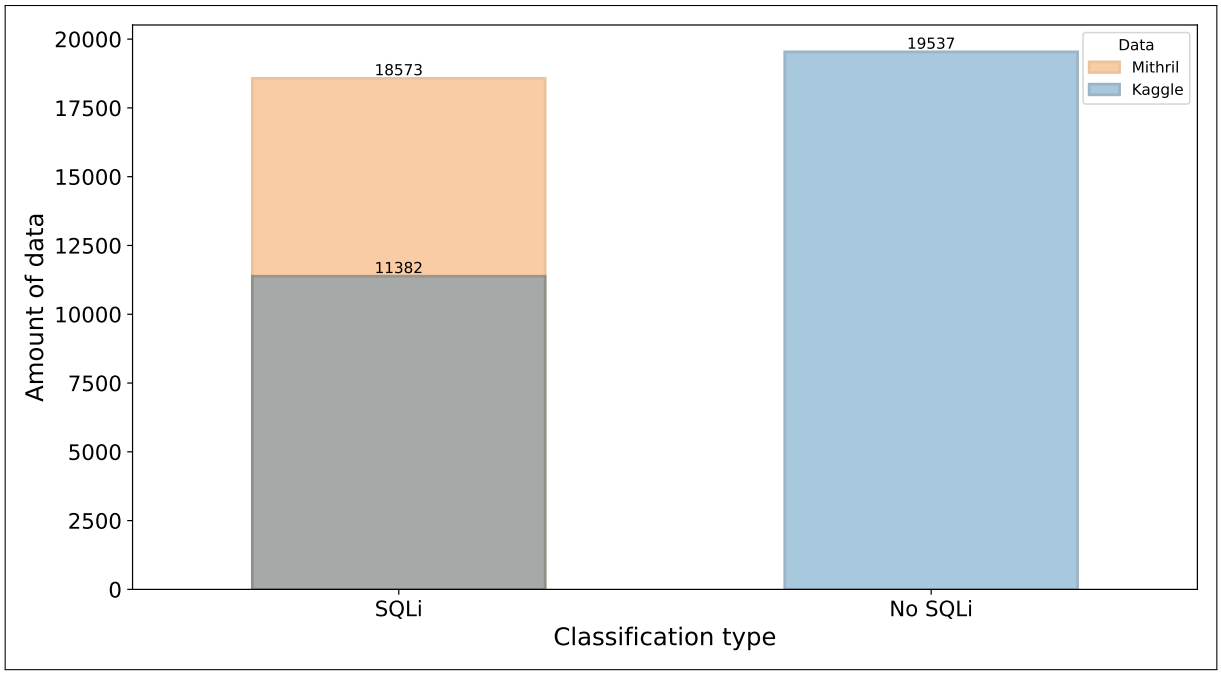


Figure 5.1: Number of data points by classification type according to each data source.

5.2.2 Data Preprocessing

From the literature ([29], [32], [34]), the following procedures have been shown to be relevant for obtaining good results when using data containing SQLi payloads:

- Replace all URLs with the text 'http://u'.
- Replace email addresses with the generic text 'user@email'.
- Decode content as necessary.
- Remove characters that are not in the ASCII set or are not printable.
- Replace all numbers with the digit "0".
- Convert all characters to lowercase.
- Add spaces before and after special characters.

These procedures were implemented using Python. SQLMap is a free and open-source tool developed under a GNU license by Bernardo Damele Assumpcao Guimaraes and Miroslav Stampar [54]. This tool has been recommended by OWASP for performing

SQL injection tests on websites [55] [56]. SQLMap has a series of encoding types and in the Table 5.1 it is possible to evidence the types of encoding against which the Python implementation can decode.

Tamper Scripts	Original String	Encoded String
apostrophemask	1 AND ‘1’=’1	1 AND %EF%BC%871%EF%BC%87 = %EF%BC%871
apostrophenullencode	1 AND ‘1’=’1	1 AND %00 %271 %00%27 = %00%271
appendnullbyte	1 AND 1=1	1 AND 1=1%00
base64encode	1’ AND SLEEP(5)#	MScgQU5EIFNMRUVQKDUpIw==
bluecoat	SELECT id FROM users WHERE id = 1	SELECT %09id FROM %09users WHERE %09 id LIKE 1
chardoubleencode	SELECT FIELD FROM TABLE	%2553%2545%254C%2545%2543 %2554%2520%2546%2549%2545 %254C%2544%2520%2546%2552 %254F%254D%2520%2554%2541 %2542%254C%2545
charencode	SELECT FIELD FROM TABLE	%53%45%4C%45%43%54 %46%49%45%4C%44 %46%52%4F%4D %54%41%42%4C%45
charunicodeencode	SELECT FIELD FROM TABLE	%u0053%u0045%u004C%u0045 %u0043%u0054 %u0020%u0046%u0049%u0045 %u004C%u0044 %u0046%u0052%u004F%u004D %u0054%u0041%u0042%u004C %u0045

Continued on the next page

overlongutf8	SELECT FIELD FROM TABLE WHERE 2>1	SELECT%C0%A0FIELD%C0%A0 FROM%C0%A0TABLE%C0%A0 WHERE%C0%A02%C0%BE1
percentage	SELECT FIELD FROM TABLE	%S%E%L%E%C%T %F%I%E%L%D %F%R%O%M %T%A%B%L%E
symboliclogical	1 AND '1'='1	1 %26%26 '1'='1
unmagicquotes	1' AND 1=1	1%bf%27AND 1=1

Table 5.1: Types of SQLMap encodings available for decoding using the custom Python implementation.

5.2.3 Data Analysis

As mentioned, the data comes from two different sources. To integrate this data, an analysis was performed by taking the data considered as SQLi payloads from the Kaggle database and comparing them in terms of the number of different words per sample with the data from Mithril. In Figure 5.2, the distribution in terms of different words is represented by a frequency histogram. When comparing the histogram for both data sources, it is noted that, although the tendency is not the same, it maintains some similarity.

A hypothesis test was conducted to determine the feasibility of data integration and verify the similarity evidenced by the frequency histogram in Figure 5.2, considering the count of different words per sample.

Statistical hypothesis tests evaluate whether a data sample is typical or atypical compared to a certain data set based on a formulated hypothesis (null hypothesis) [57]. This involves calculating a test statistic for the data sample and comparing it to the formulated hypothesis, with the resulting value known as the p-value. A low p-value (usually less than 5%) suggests that the sample is atypical, leading to the rejection of the null hypothesis, while a high p-value leads to acceptance [57].

Given that the amount of data in both sets is greater than 30, a hypothesis test known as the Z-Test for two means was implemented using the Python "statsmodels" module [58]. The following hypotheses were proposed:

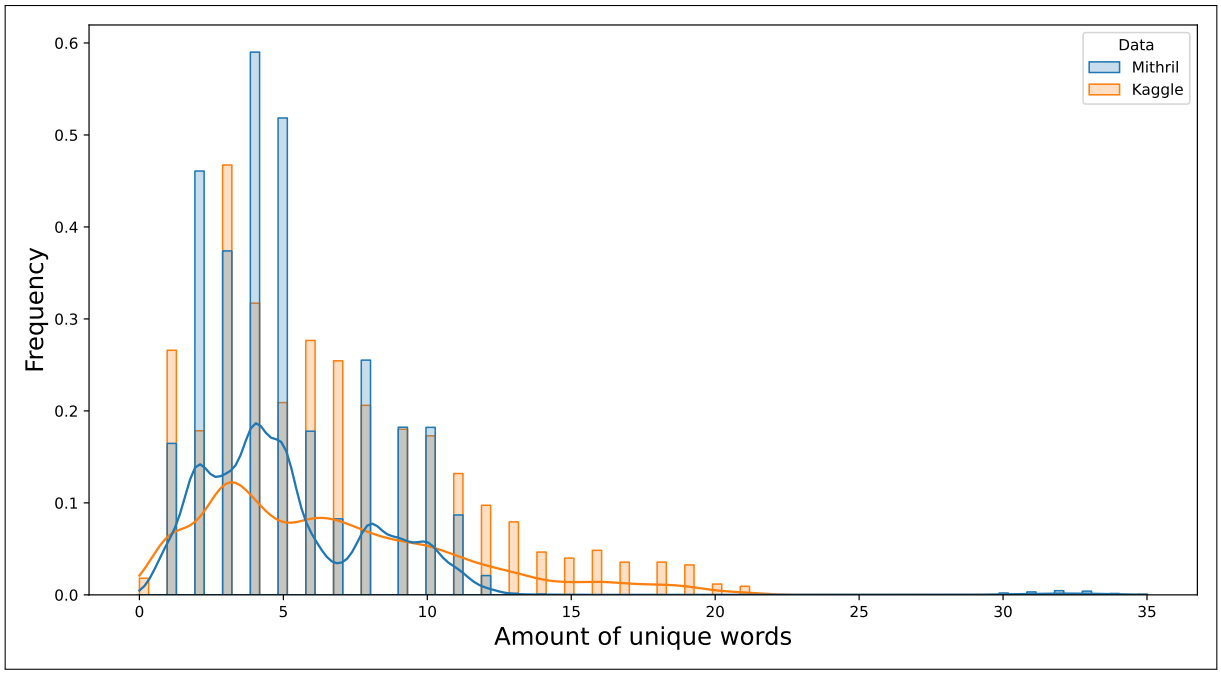


Figure 5.2: Distribution of unique words in the attack class according to each data source.

- Null hypothesis ($H0$): there is no difference between the Kaggle and Mithril data.
- Alternative hypothesis ($H1$): there is a difference between the Kaggle and Mithril data.

The Z-Test conducted for the two data sets, taking the number of distinct words per sample, resulted in a p-value of $1.832710135153467 \cdot 10^{-47}$. Since the obtained p-value was less than 0.05 (corresponding to 5%), the null hypothesis was rejected, indicating that the data actually differed considering the number of distinct words per sample. This led to the conclusion that data integration for these two sources was not feasible.

5.2.4 Data Integration

Despite the fact stated in subsection 5.2.3 regarding the similarity of the data sources and their feasibility for integration, it was decided to integrate the data to respond to the interest in structuring a database that includes data from Mithril.

Upon integrating the data, the resulting database (database 1.1) showed an imbalance in the amount of data per class (SQLi and Non-SQLi), as evidenced in Figure 5.3a. It was

noted that database 1.1 contained duplicate data; the top 5 duplicated payloads are shown in Table 5.2. It was then decided to remove the duplicate data, and the distribution of the resulting database (database 2.1) can be seen in Figure 5.3c.

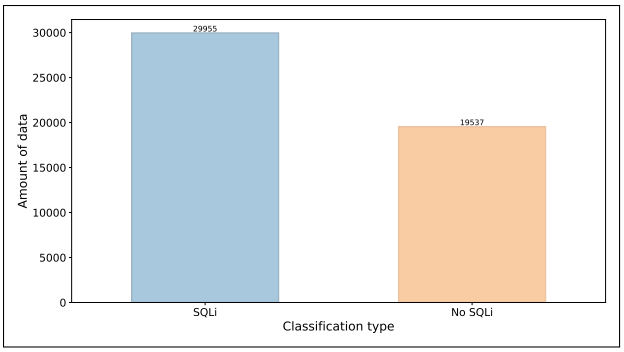
Payload	Number of duplicates
0	972
0 union all select 0 , (@ @ version) - -	771
user @ email	770
0 . 0 e + 0	646
’) or ‘ 0 ‘ = ‘ 0 ‘ ; - - -	576

Table 5.2: The 5 most common duplicated payloads present in database 1.1

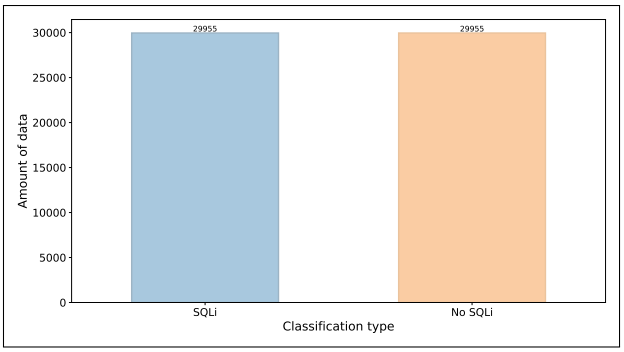
Data balancing is a practice that leads to better results when using the data to build machine learning models [59]. It was decided to employ data balancing using the Python library "imblearn" [60].

Specifically, "Random Over Sampler" was applied to database 1.1, the distribution of the resulting database (database 1.2) can be seen in Figure 5.3b, and "Random Under Sampler" was applied to database 2.1, the distribution of the resulting database (database 2.2) can be seen in Figure 5.3d.

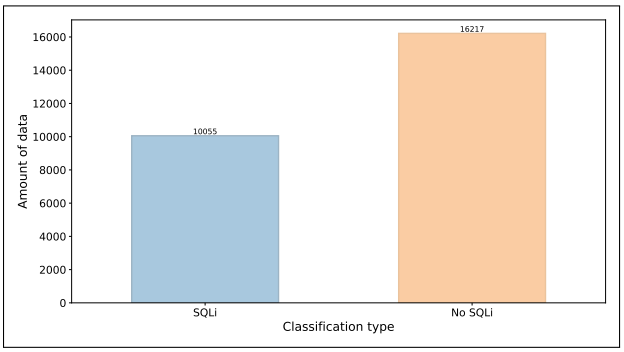
This resulted in four distinct instances of data integration, representing the four future scenarios on which the selected machine learning techniques will be implemented (training and testing). For validation, data from a different implementation of Mithril than the one used previously in subsection 5.2.4 was taken, exclusively SQLi payloads corresponding to March 2024.



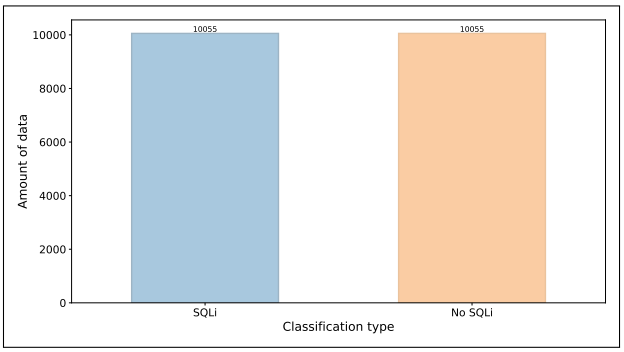
(a) Data obtained after integration (Database 1.1).



(b) Database 1.1 after data balancing (Database 1.2).



(c) Data obtained after removing duplicates in database 1.1 (Database 2.1).



(d) Database 2.1 after data balancing (Database 2.2).

Figure 5.3: Number of data points per classification type according to each resulting database.

5.2.5 Feature Generation

Natural Language Processing (NLP) is a field within artificial intelligence aimed at emulating human understanding of textual data [61]. In the machine learning field, particularly in text analysis, the challenge is to transform textual data into numerical feature vectors. To address this challenge, various techniques have emerged using artificial intelligence (which have proven to be effective [62], [63], [64]), such as the model known as "Bag of Words," where the frequency of word occurrences in a text is used to generate a numerical feature vector for subsequent analysis and processing [61].

After the data preprocessing exposed in the subsection 5.2.2, "Bag of Words" was applied using the "Scikit-learn" library in Python [65], to the four different instances of the database.

The Figure 5.4 shows what was described above during the section 5.2.

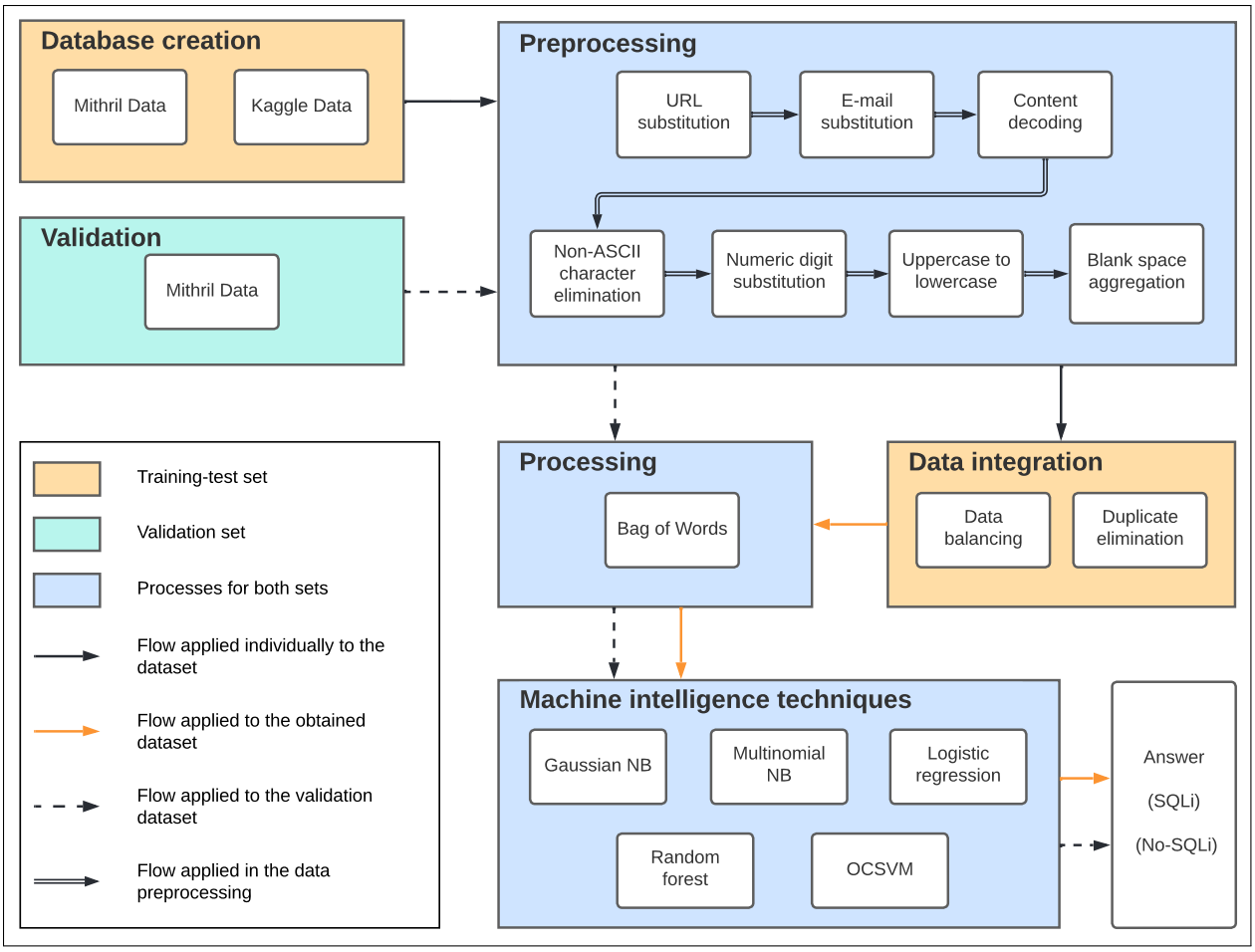


Figure 5.4: Data processing flow for training, testing, and validation sets

5.3 Implementation and Training of Selected Techniques

The implementation of the selected machine learning techniques was carried out using Python as the programming language. This implementation was done using the following specific modules of the "Scikit-learn" library [65] for each technique:

- “GaussianNB” de “sklearn.naive_bayes”: Used to implement the Gaussian Naïve Bayes classifier.
- “MultinomialNB” de “sklearn.naive_bayes”: Used to implement the Multinomial Naïve Bayes classifier.
- “LogisticRegression” de “sklearn.linear_model”: Used to implement Logistic Regression as a binary classifier.

- "RandomForestClassifier" de "sklearn.ensemble": Used to implement the Random Forest classifier.
- "OneClassSVM" de "sklearn.svm": Used to implement Support Vector Machine (SVM) with the approach of performing classification from a single dataset.

5.3.1 Hyper-parameter Tuning

Hyper-parameter tuning is the process of selecting the parameters that an artificial intelligence algorithm requires before training with the data. These parameters significantly affect the model's performance [66]. The goal of tuning is to find the parameter configuration that produces the best results in terms of algorithm performance and effectiveness [66].

Hyper-parameter tuning was carried out based on the literature review, where the selection of the parameters for each of the implementations was made given the effectiveness demonstrated according to the studies present in the state of the art exposed in the Table 4.1. The parameter configuration for each algorithm is listed in the Table 5.3

Algorithm	Hyper-parameters
GaussianNB	(priors=None, var_smoothing=1e-09)
MultinomialNB	(alpha=0.1, force_alpha=True, fit_prior=True)
LogisticRegression	(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)

Continued on the next page

RandomForestClassifier	(n_estimators=23, criterion='gini', max_depth=18, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=-1, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None, monotonic_cst=None)
OneClassSVM	(kernel='rbf', degree=3, gamma='scale', coef0=0.0, tol=0.001, nu=0.01, shrinking=True, cache_size=200, verbose=False, max_iter=-1)

Table 5.3: Hyper-parameters for each algorithm

5.3.2 Results gathering

Once the hyper-parameters for each implemented model were determined, the results of each implementation were compiled in order to calculate performance metrics. It should be noted that there were 4 different scenarios given the four different instances of the database. All models were executed on a laptop with an Intel(R) Core(TM) i7-1255U twelfth-generation processor with 16 GB of RAM. The obtained results are presented in detail in chapter 6.

5.4 Performance evaluation and benchmarking of the techniques implemented

5.4.1 Performance Metrics

When discussing performance metrics for computational intelligence models, it's important to define the basic components that help determine these metrics and provide information on the accuracy, precision, recall, and overall effectiveness of the model, considering that in anomaly classification problems, there are negative (normal) and positive (anomalous) classes [67].

True Positives (TP): number of positive data correctly classified.

False Positives (FP): number of negative data correctly classified.

True Negatives (TN): number of positive data incorrectly classified.

False Negatives (FN): number of positive data incorrectly classified.

Specifically for the problem addressed in this work, the negative class corresponds to non-attack instances, and the positive class corresponds to SQLi attack instances. Now, the selection of performance metrics was based on those widely accepted in the literature from the state of the art presented in Table 4.1. The chosen metrics for comparing the implemented techniques are described below.

Accuracy: is the percentage of instances in which the model made a correct classification. It measures the overall effectiveness of a model in correctly classifying both positive and negative classes and is calculated using Equation 5.1 [68].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

Balanced Accuracy: metric used to evaluate the performance of a classification model, especially when using imbalanced datasets. It is defined as the arithmetic mean of sensi-

tivity (recall) and the true negative rate, calculated according to Equation 5.2 [69]. When calculated on balanced datasets, it becomes the accuracy described above.

$$Balanced\_Accuracy = \frac{1}{2} \cdot \left(\frac{TP}{TP + FN} + \frac{TN}{TN + TP} \right) \tag{5.2}$$

Precision: is the proportion of positive instances correctly classified out of the total positive classifications. It indicates the accuracy of positive classifications made by the model, calculated using Equation 5.3 [67].

$$Precision = \frac{TP}{TP + FP} \tag{5.3}$$

Recall: is the proportion of positive instances correctly classified out of the total true positives. Also known as the true positive rate, it uses Equation 5.4 [67].

$$Recall = \frac{TP}{TP + FN} \tag{5.4}$$

F1-Score: also known as F-measure, it establishes the balance between the precision and recall of a classifier, and it is one of the most commonly used performance metrics in classification problems [70]. It is calculated using a harmonic mean between precision and recall as shown in Equation 5.5.

$$F1Score = 2 \times \frac{\left(\frac{TP}{TP+FP}\right) \cdot \left(\frac{TP}{TP+FN}\right)}{\left(\frac{TP}{TP+FP}\right) + \left(\frac{TP}{TP+FN}\right)} \tag{5.5}$$

Confusion Matrix: a tool that allows visualizing the performance of a computational intelligence classification model, showing correct and incorrect predictions compared to actual classifications [71].

Each column of the matrix represents the number of predictions for each class made by the model, and each row corresponds to instances belonging to the actual classification.

The structure of a confusion matrix is illustrated in Figure 5.5.

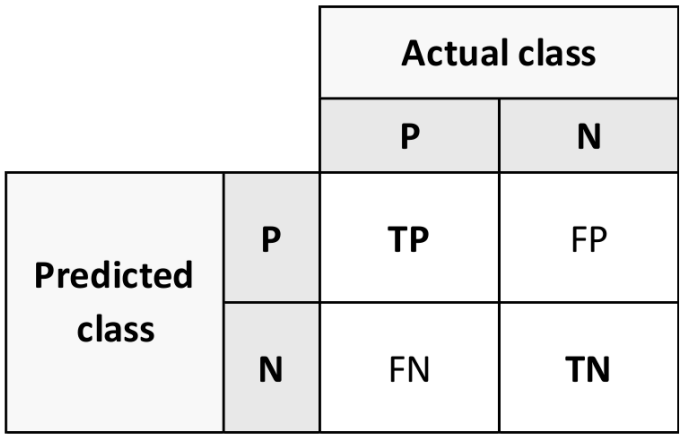


Figure 5.5: General scheme of a confusion matrix [71].

Both the specific results of calculating each metric and additional analyses are detailed in chapter 6, following the standard formulas for each metric defined above. However, it's important to note that the validation set was not composed of data from the two target classes, and due to this limitation, the performance evaluation of each model on this set is performed exclusively using the recall metric.

5.4.2 Benchmark and recommendation

According to the evaluation performed on the different computational intelligence models for SQLi attack detection, the Logistic Regression model emerged as the most effective, achieving a precision of 99.45% and an F1-score of 99.45% using the database 1.2. In comparison, the Multinomial Naïve Bayes model also showed good results with a precision of 98.65% and an F1-score of 98.64%. The OCSVM model presented the worst performance, with a precision of 50.69% and an F1-score of 66.79%, indicating a high rate of false negatives and registering long times in terms of the training and testing stages.

As mentioned earlier, specific results for each scenario are detailed in chapter 6.

6 Results and analysis

6.1 Database 1.1 and 1.2

6.1.1 Training and test set

To analyze the performance of the different implemented models using the database 1.1, the confusion matrices obtained during the training and testing stages are presented in Figure 6.1, along with the performance metrics in Table 6.1. The Logistic Regression model outperformed the other models with an accuracy of 0.9941 (99.41%) and an F1 Score of 0.9952 (99.52%). The Multinomial Naïve Bayes model also achieved good results with an accuracy of 0.9366 (93.66%) and an F1 Score of 0.9496 (94.96%), while the Gaussian Naïve Bayes model and the Random Forest model showed intermediate performances. On the other hand, the One-Class SVM (OCSVM) model had the worst performance with an accuracy of 0.6055 (60.55%) and an F1 Score of 0.7519 (75.19%).

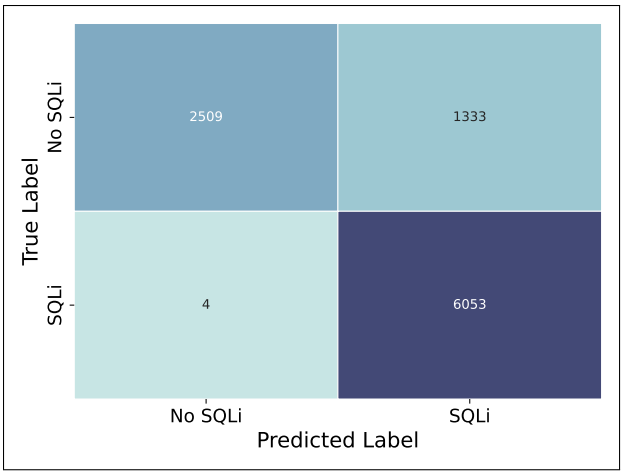
Similarly, Figure 6.2 presents the confusion matrices and Table 6.2 presents the performance metrics obtained when using the database 1.2. Again, Logistic Regression stood out with an accuracy of 0.9945 (99.45%) and an F1 Score of 0.9945 (99.45%), and the Multinomial Naïve Bayes model also achieved good results with an accuracy of 0.9865 (98.65%) and an F1 Score of 0.9864 (98.64%). In this dataset, the Random Forest model showed better performance compared to the implementation using the database 1.1, reaching an accuracy of 0.9135 (91.35%) and an F1 Score of 0.9062 (90.62%).

Model	Accuracy	Balanced Accuracy	Precision	Recall	F1 Score
Gaussian Naïve Bayes	0.8649	0.8262	0.8195	0.9993	0.9005
Multinomial Naïve Bayes	0.9366	0.9247	0.9231	0.9777	0.9496
Logistic Regression	0.9941	0.9942	0.9965	0.9939	0.9952
Random Forest	0.8109	0.7582	0.7665	0.9936	0.8654
One-Class SVM	0.6055	0.4984	0.6111	0.9771	0.7519

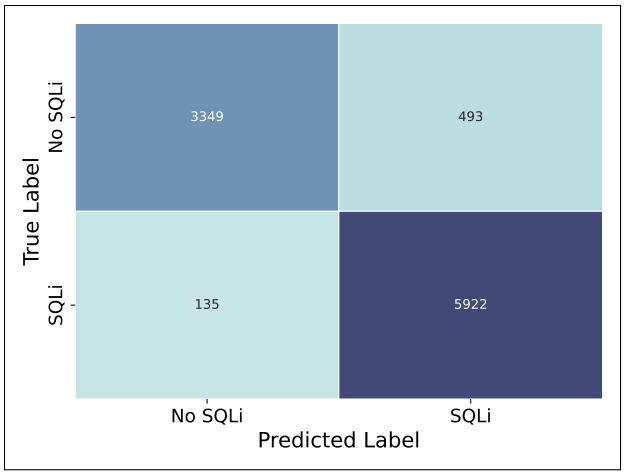
Table 6.1: Performance metrics obtained by implementing each model using database 1.1.

Model	Accuracy	Balanced Accuracy	Precision	Recall	F1 Score
Gaussian Naïve Bayes	0.8764	0.8761	0.8025	0.9995	0.8902
Multinomial Naïve Bayes	0.9865	0.9865	0.9974	0.9755	0.9864
Logistic Regression	0.9945	0.9945	0.996	0.993	0.9945
Random Forest	0.9135	0.9137	0.9923	0.8339	0.9062
One-Class SVM	0.5069	0.5056	0.5042	0.9888	0.6679

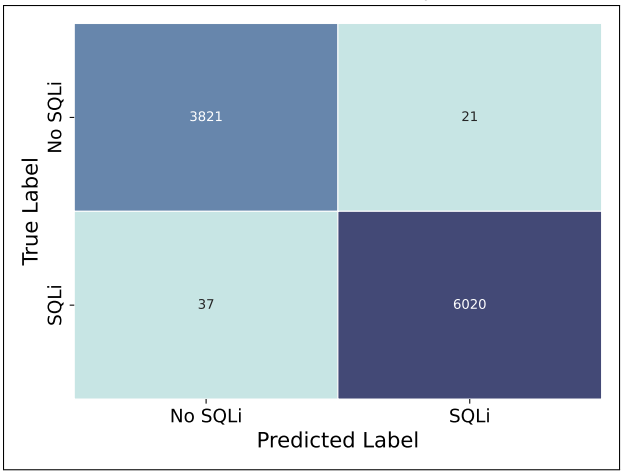
Table 6.2: Performance metrics obtained by implementing each model using database 1.2.



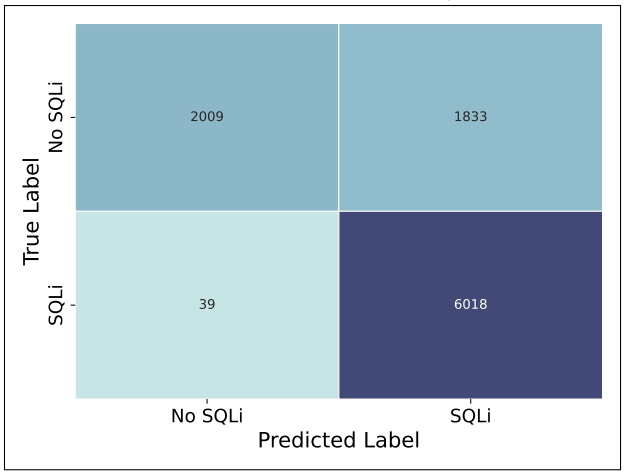
(a) Gaussian Naïve Bayes



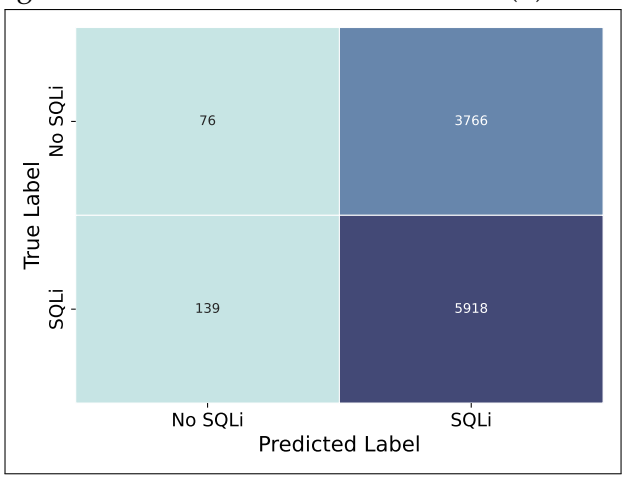
(b) Multinomial Naïve Bayes



(c) Logistic Regression

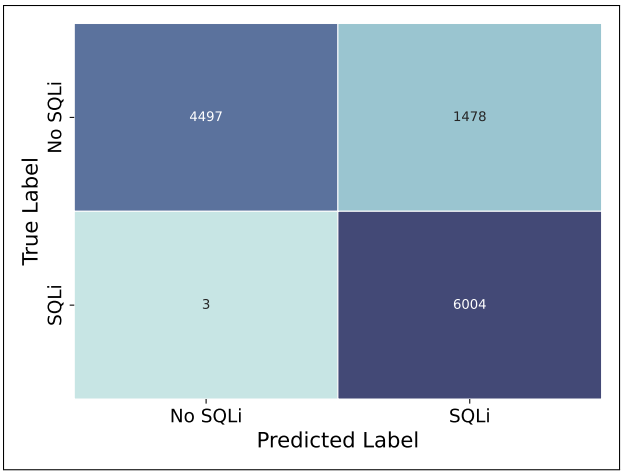


(d) Random Forest

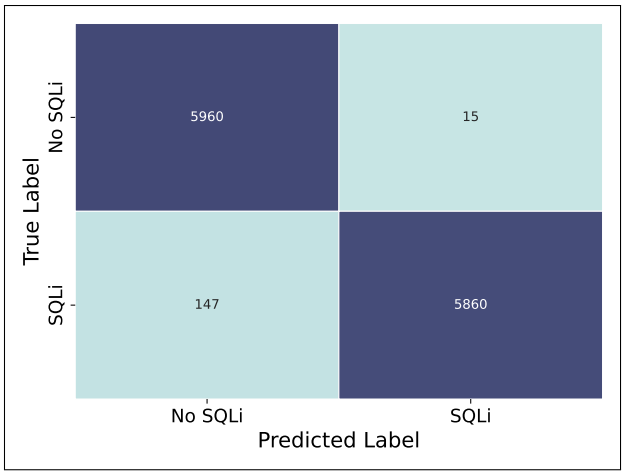


(e) One-Class SVM

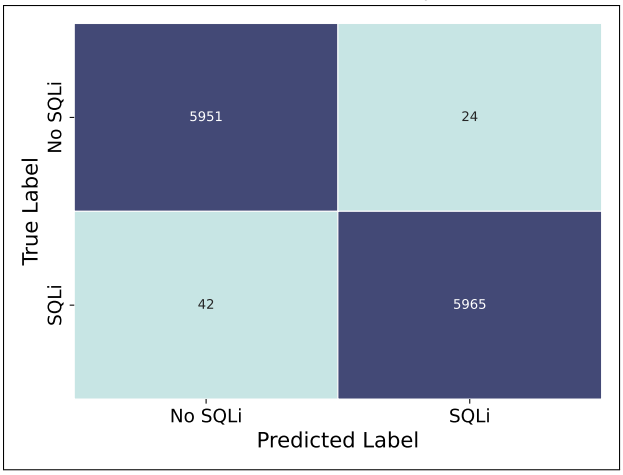
Figure 6.1: Confusion matrices obtained for each model during the training and testing stage using database 1.1.



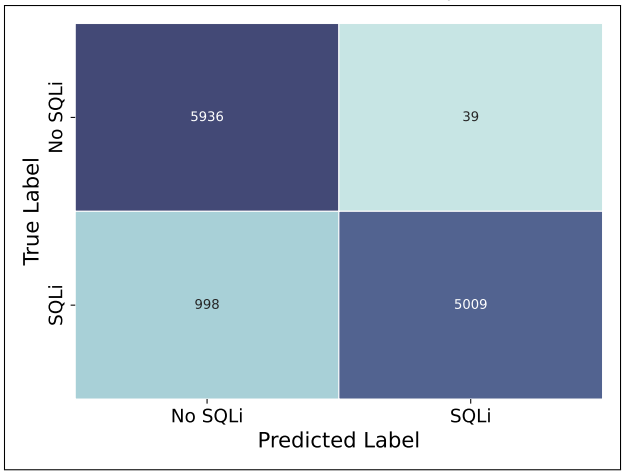
(a) Gaussian Naïve Bayes



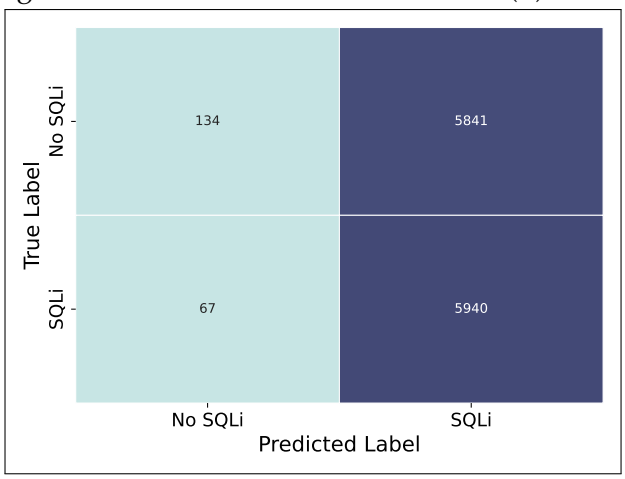
(b) Multinomial Naïve Bayes



(c) Logistic Regression



(d) Random Forest



(e) One-Class SVM

Figure 6.2: Confusion matrices obtained for each model during the training and testing stage using database 1.2.

These results were achieved with different training and testing times for each model, as shown in Table 6.3. In the training process, it was observed that the Random Forest model stood out for its efficiency with 3.4472 seconds in database 1.1 and 8.7517 seconds in database 1.2. On the other hand, the One-Class SVM model had the longest training times, specifically 504.9279 seconds in the database 1.1 and 381.0886 seconds in the database 1.2.

Regarding the testing phase, a similar tendency was observed, where the Random Forest model stood out again for its speed, with a time of 0.3624 seconds when using the database 1.1 and 1.6974 seconds in the database 1.2, while the One-Class SVM algorithm was the slowest in both databases, with a time of 107.2197 seconds in the database 1.1 and 268.3849 seconds in the database 1.2.

Model	Database 1.1		Database 1.2	
	Training Time	Testing Time	Training Time	Testing Time
Gaussian Naïve Bayes	15.6946	2.7145	24.4493	4.7831
Multinomial Naïve Bayes	3.5644	1.2913	8.8071	3.0579
Logistic Regression	20.4303	1.2570	33.0457	3.0664
Random Forest	3.4472	0.3624	8.7517	1.6974
One-Class SVM	504.9279	107.2197	381.0886	268.3849

Table 6.3: Training and testing times for each model using databases 1.1 and 1.2

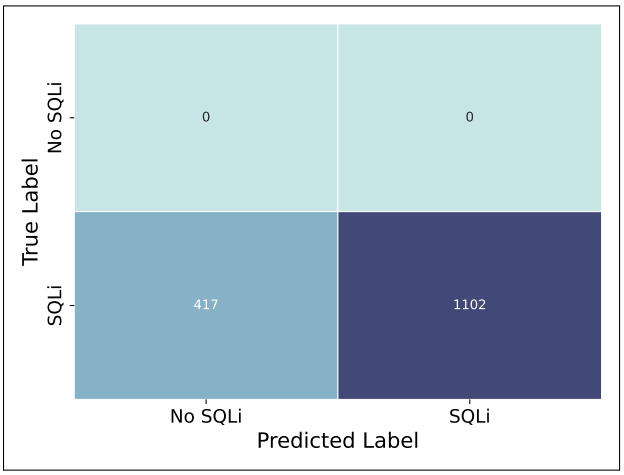
6.1.2 Validation set

The results shown in Table 6.4 correspond to the Recall values obtained for each model when using the database 1.1 as the source for training and testing. It can be observed that the Logistic Regression model presented the highest Recall value with 0.9895 (98.95%), followed by Random Forest with 0.9131 (91.31%). This indicates that both models are highly effective in identifying true positives. On the other hand, Gaussian Naïve Bayes showed the lowest performance with a Recall of 0.9895 (98.95%), suggesting a higher number of false negatives compared to the other models. The confusion matrices in Figure 6.3 corroborate these results, showing that Logistic Regression and Random Forest have a high number of true positives and a low number of false negatives, while Gaussian Naïve Bayes has a higher proportion of false negatives.

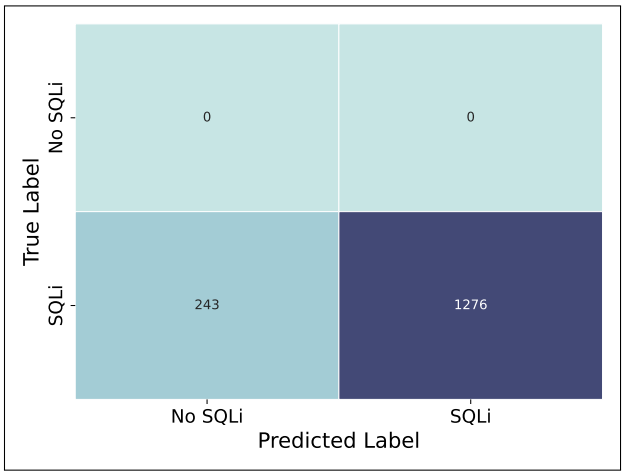
The performance metrics obtained during the validation of each model trained using database 1.2 are presented in Table 6.5. Again, Logistic Regression achieved the best performance with a Recall of 0.9895 (98.95), while the Gaussian Naïve Bayes model maintained a Recall of 0.7255 (0.7255). Notably, the Random Forest model experienced a decrease in its Recall value, with a value equal to 0.8354 (83.54). The confusion matrices shown in Figure 6.4 support the above, showing that Logistic Regression remains effective in minimizing false negatives, while Random Forest shows an increase in false negatives when using database 1.2, which aligns with the decrease in its Recall observed in Table 6.5.

Model	Recall
Gaussian Naïve Bayes	0.7255
Multinomial Naïve Bayes	0.84
Logistic Regression	0.9895
Random Forest	0.9131

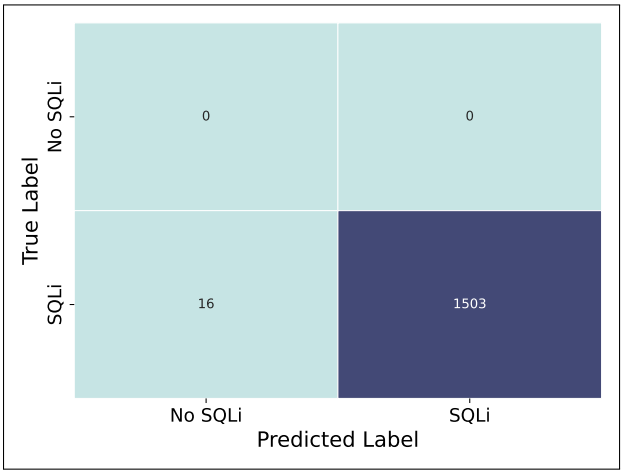
Table 6.4: Performance metrics obtained in the validation for each model trained using the database 1.1.



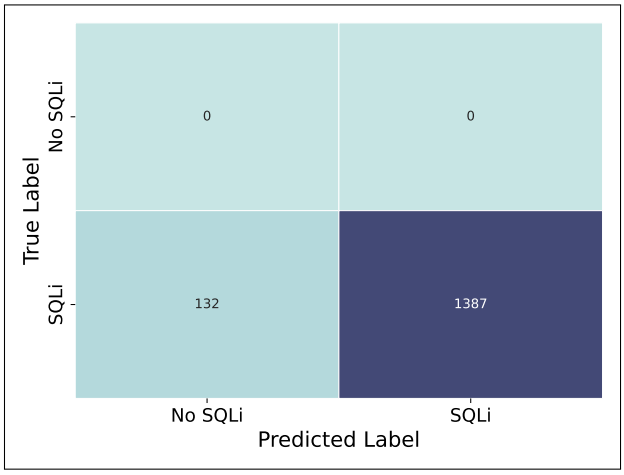
(a) Gaussian Naïve Bayes



(b) Multinomial Naïve Bayes



(c) Logistic Regression

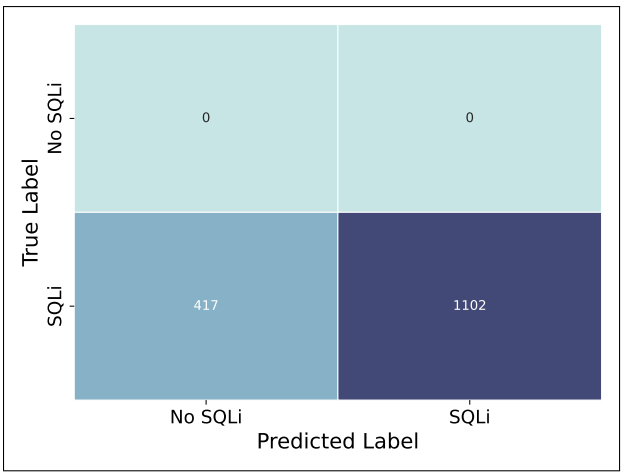


(d) Random Forest

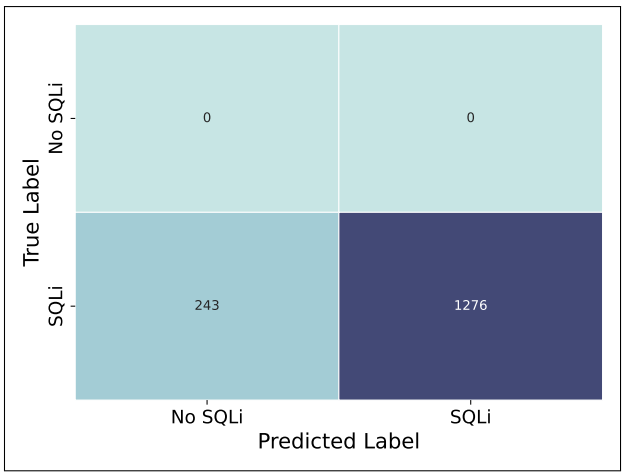
Figure 6.3: Confusion matrices obtained in the validation for each model trained using the database 1.1.

Model	Recall
Gaussian Naïve Bayes	0.7255
Multinomial Naïve Bayes	0.84
Logistic Regression	0.9895
Random Forest	0.8354

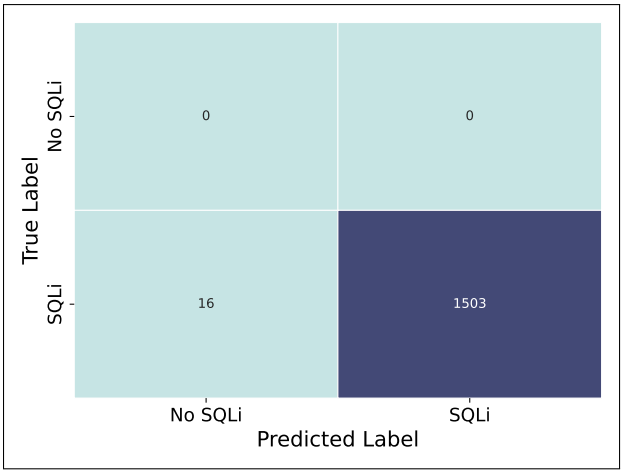
Table 6.5: Performance metrics obtained in the validation for each model trained using the database 1.2.



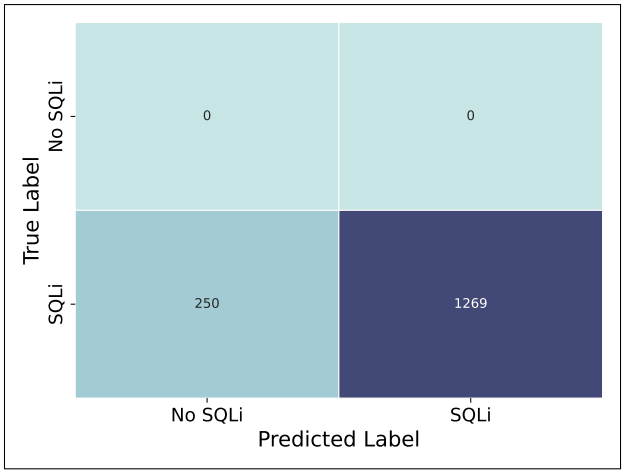
(a) Gaussian Naïve Bayes



(b) Multinomial Naïve Bayes



(c) Logistic Regression



(d) Random Forest

Figure 6.4: Confusion matrices obtained in the validation for each model trained using the database 1.2.

6.2 Database 2.1 and 2.2

6.2.1 Training and test set

To analyze the performance of the different models implemented using database 2.1, the confusion matrices obtained during the training and testing stage are presented in Figure 6.5, as well as the performance metrics in Table 6.6. The Logistic Regression model outperformed the other models with an accuracy of 0.988 (98.8%) and an F1 Score of 0.9846 (98.46%). The Multinomial Naïve Bayes model also achieved good results with an accuracy of 0.9814 (98.14%) and an F1 Score of 0.9756 (97.56%), while the Gaussian Naïve Bayes model and the Random Forest model showed intermediate performances. On the other hand, the One-Class SVM (OCSVM) model had the worst performance with an accuracy of 0.3966 (39.66%) and an F1 Score of 0.5512 (55.12%).

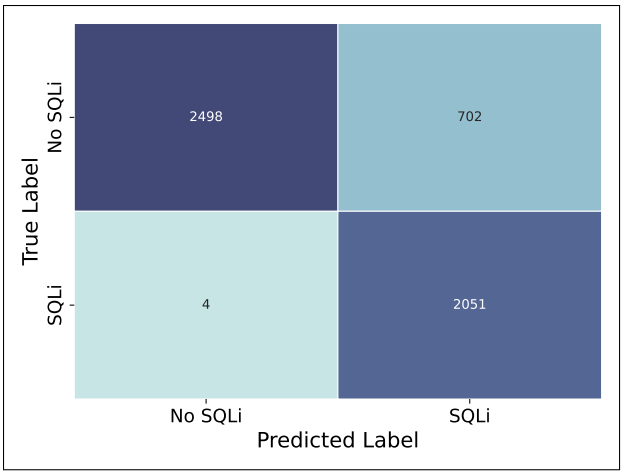
Furthermore, Figure 6.6 presents the confusion matrices, and Table 6.7 the performance metrics obtained when using database 2.2. Again, Logistic Regression stood out with an accuracy of 0.9876 (98.76%) and an F1 Score of 0.9872 (98.72%), while the Multinomial Naive Bayes model also obtained good results with an accuracy of 0.9784 (97.84%) and an F1 Score of 0.9776 (97.76%). In this dataset, the Random Forest model showed improved performance compared to the implementation using database 2.1, reaching an accuracy of 0.9162 (91.62%) and an F1 Score of 0.9086 (90.86%).

Model	Accuracy	Balanced Accuracy	Precision	Recall	F1 Score
Gaussian Naïve Bayes	0.8657	0.8893	0.745	0.9981	0.8532
Multinomial Naïve Bayes	0.9814	0.9766	0.9975	0.9547	0.9756
Logistic Regression	0.988	0.9868	0.9882	0.981	0.9846
Random Forest	0.8422	0.7983	1.0	0.5966	0.7473
One-Class SVM	0.3966	0.4951	0.3886	0.9474	0.5512

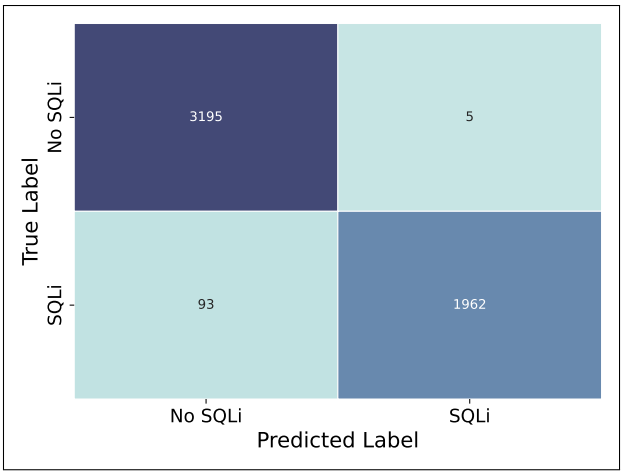
Table 6.6: Performance metrics obtained by implementing each model using database 2.1.

Model	Accuracy	Balanced Accuracy	Precision	Recall	F1 Score
Gaussian Naïve Bayes	0.8836	0.8862	0.8082	0.999	0.8935
Multinomial Naïve Bayes	0.9784	0.9781	0.9891	0.9664	0.9776
Logistic Regression	0.9876	0.9875	0.9923	0.9822	0.9872
Random Forest	0.9162	0.9148	0.9733	0.852	0.9086
One-Class SVM	0.4853	0.4955	0.4865	0.9507	0.6436

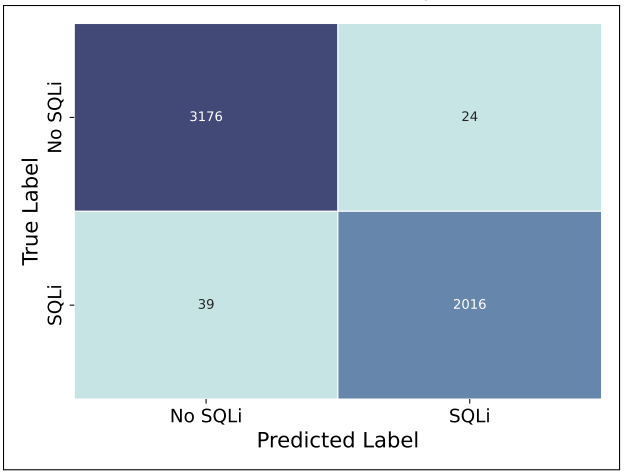
Table 6.7: Performance metrics obtained by implementing each model using database 2.2.



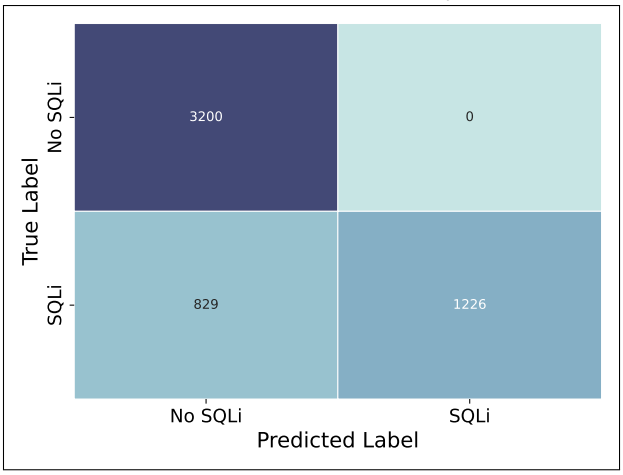
(a) Gaussian Naïve Bayes



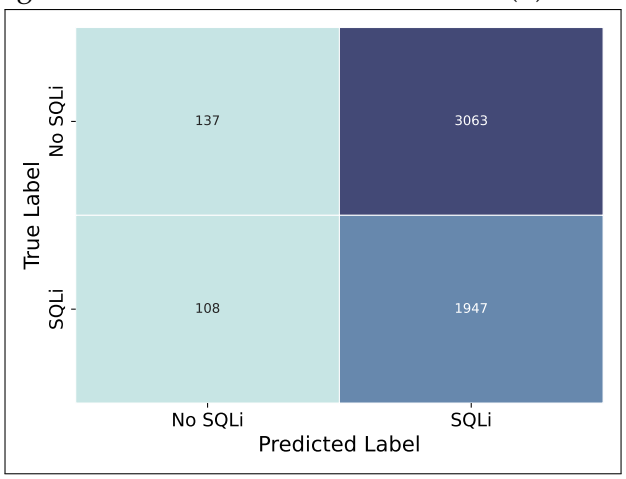
(b) Multinomial Naïve Bayes



(c) Logistic Regression

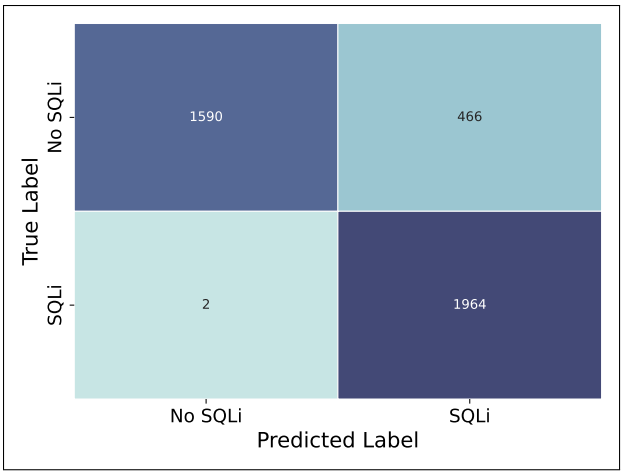


(d) Random Forest

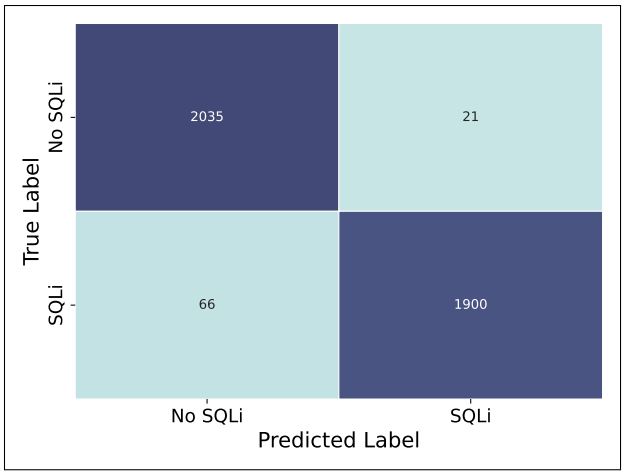


(e) One-Class SVM

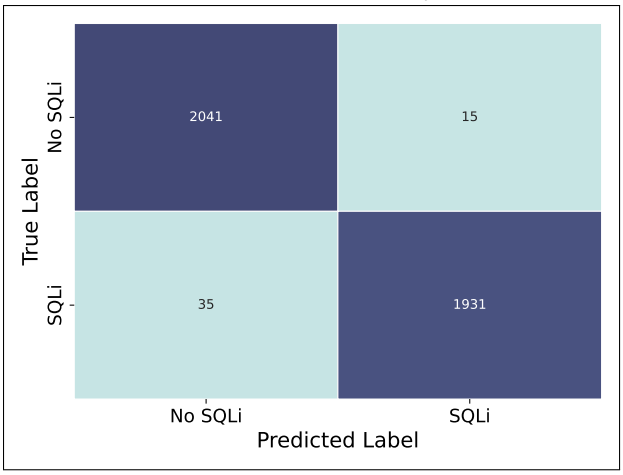
Figure 6.5: Confusion matrices obtained for each model during the training and testing stage using database 2.1.



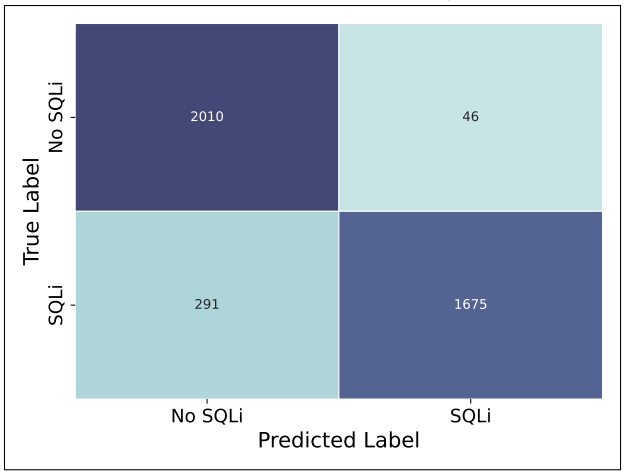
(a) Gaussian Naïve Bayes



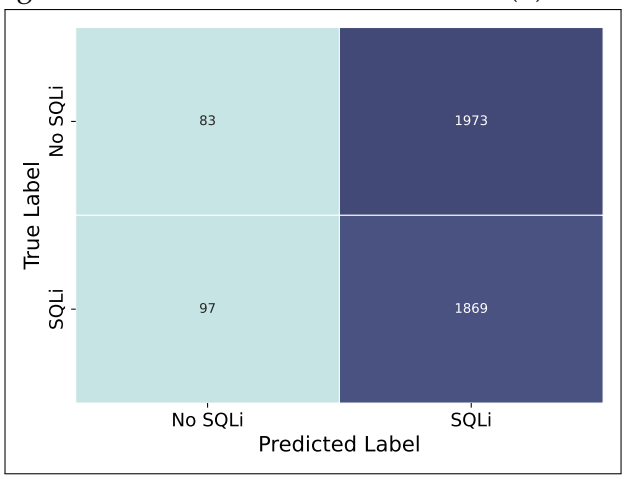
(b) Multinomial Naïve Bayes



(c) Logistic Regression



(d) Random Forest



(e) One-Class SVM

Figure 6.6: Confusion matrices obtained for each model during the training and testing stage using database 2.2.

These results were achieved with different training and testing times for each model, as can be observed in Table 6.8. During the training phase, the Random Forest model was the fastest in both databases, with a time of 1.5125 seconds in database 2.1 and 1.7408 seconds in database 2.2. On the other hand, the One-Class SVM algorithm had the longest training times, specifically 138.7639 seconds in database 2.1 and 99.6006 seconds in database 2.2.

The times also varied according to each model in the testing phase. The Random Forest model was again the fastest in both databases, with a time of 0.2042 seconds in database 2.1 and 0.3699 seconds in database 2.2. Meanwhile, the One-Class SVM model was the slowest in both databases, with a time equal to 57.9170 seconds in database 2.1 and 205.3404 seconds in database 2.2.

Model	Database 1.1		Database 1.2	
	Training Time	Testing Time	Training Time	Testing Time
Gaussian Naïve Bayes	6.8385	1.2266	6.4089	1.2645
Multinomial Naïve Bayes	1.7315	1.0022	2.6571	0.6294
Logistic Regression	12.2442	0.6946	6.8998	0.6761
Random Forest	1.5125	0.2042	1.7408	0.3699
One-Class SVM	138.7639	57.9170	99.6006	205.3404

Table 6.8: Training and testing times for each model using databases 2 y 2.2

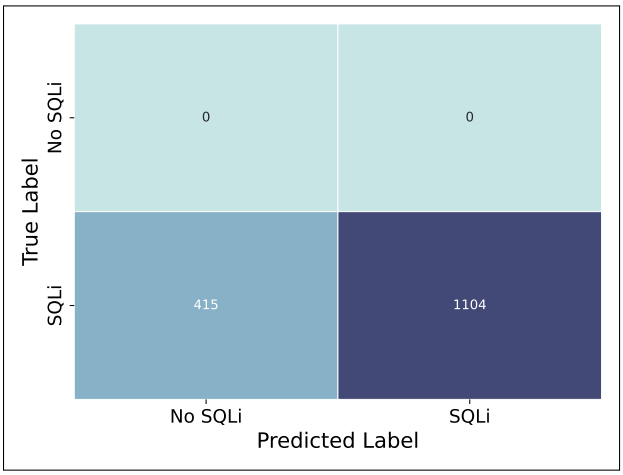
6.2.2 Validation set

The results shown in Table 6.9 correspond to the Recall values obtained for each model when using database 2.1 as the source for training and testing. It can be observed that the Logistic Regression model presented the highest Recall value with 0.9737 (97.37%), followed by Random Forest with 0.8354 (83.54%). This indicates that both models are highly effective in identifying true positives. On the other hand, Gaussian Naïve Bayes showed the lowest performance with a Recall of 0.7268 (72.68%), suggesting a higher number of false negatives compared to the other models. The confusion matrices in Figure 6.7 corroborate these results, showing that Logistic Regression has a high number of true positives and a low number of false negatives, while Gaussian Naïve Bayes has the highest proportion of false negatives.

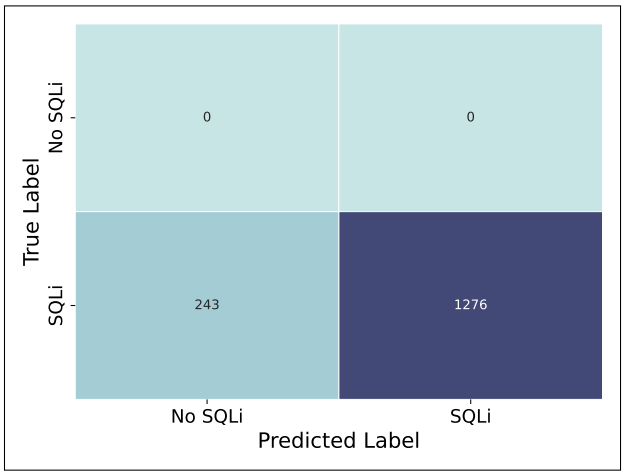
The performance metrics obtained during the validation of each model trained using database 2.2 are presented in Table 6.10. Again, Logistic Regression achieved the best performance with a Recall of 0.9895 (98.95%), while the Gaussian Naïve Bayes model maintained a Recall of 0.7268 (72.68%). Notably, the Random Forest model experienced a slight increase in its Recall value, with a value equal to 0.8492 (84.92%). The confusion matrices shown in Figure 6.4 support the above, showing that Logistic Regression remains effective in minimizing false negatives when using database 1.2.

Model	Recall
Gaussian Naïve Bayes	0.7268
Multinomial Naïve Bayes	0.84
Logistic Regression	0.9737
Random Forest	0.8354

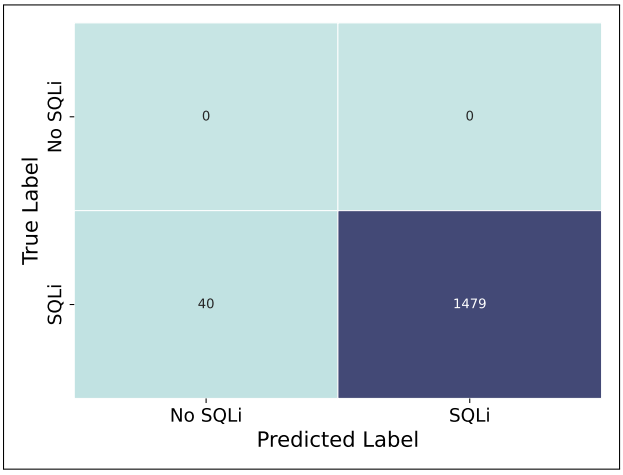
Table 6.9: Performance metrics obtained in the validation for each model trained using the database 2.1.



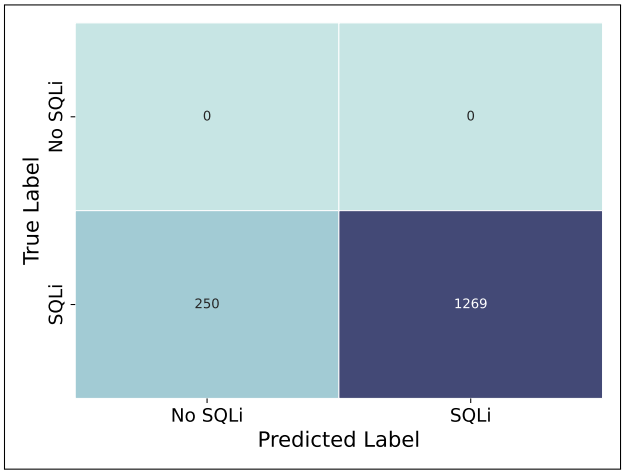
(a) Gaussian Naïve Bayes



(b) Multinomial Naïve Bayes



(c) Logistic Regression

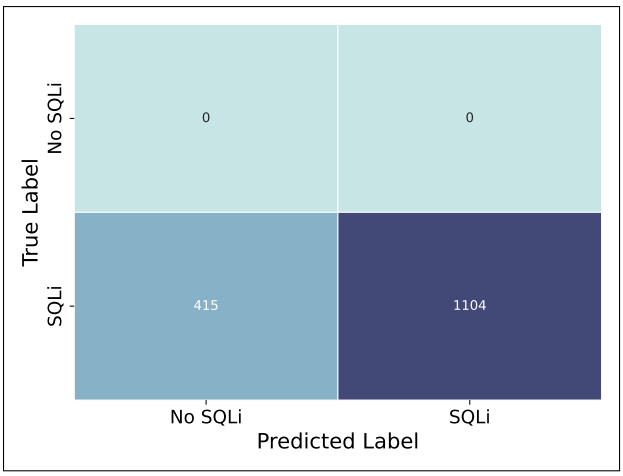


(d) Random Forest

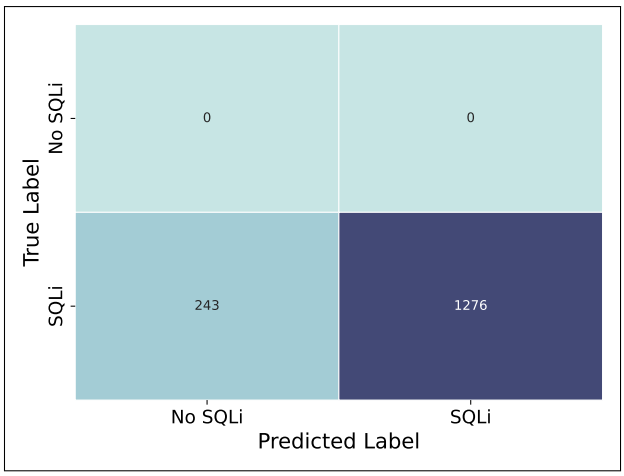
Figure 6.7: Confusion matrices obtained in the validation for each model trained using the database 2.1.

Model	Recall
Gaussian Naïve Bayes	0.7268
Multinomial Naïve Bayes	0.84
Logistic Regression	0.9895
Random Forest	0.8492

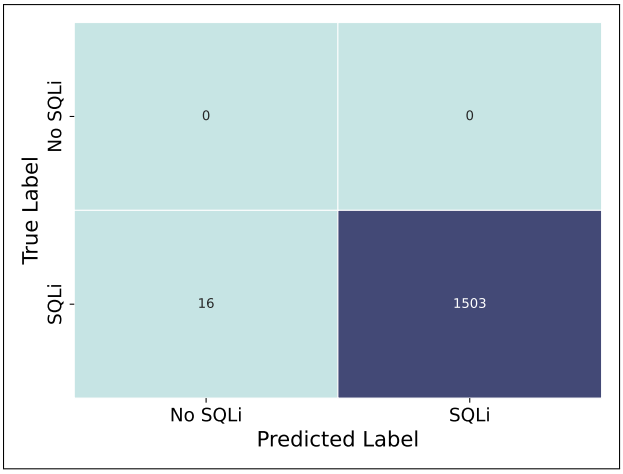
Table 6.10: Performance metrics obtained in the validation for each model trained using the database 2.2.



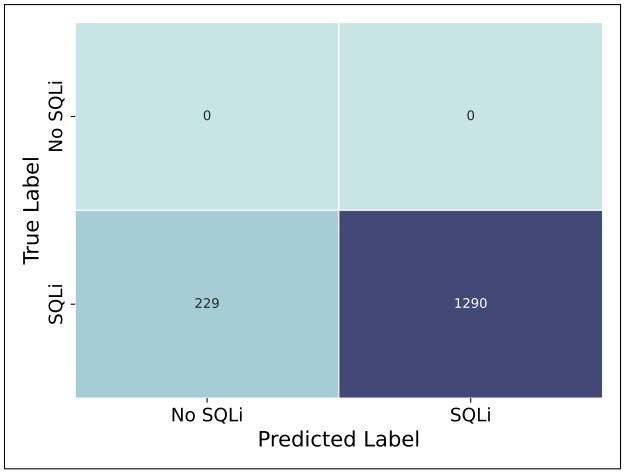
(a) Gaussian Naïve Bayes



(b) Multinomial Naïve Bayes



(c) Logistic Regression



(d) Random Forest

Figure 6.8: Confusion matrices obtained in the validation for each model trained using the database 2.2.

It should be noted that it was decided to exclude the OCSVM model from the validation stage for all the implemented instances. As could be evidenced, due to the nature and specific characteristics of the datasets, the detection task is more effectively addressed with the other machine learning models. Although the OCSVM approach is interesting, it may not fit well with the characteristics of the data used and it is recognized that addressing the model from the data corresponding to attacks becomes complicated. Additionally, the reported times indicated that the computational cost in terms of training and testing of the OCSVM model is significantly higher compared to the other models, without a proportional increase in performance.

7 Conclusions

The main objective of this work was to evaluate strategies for detecting SQL injection (SQLi) attacks based on computational intelligence, with the aim of providing a recommendation to enhance the security of AizoOn Technology Consulting's web application firewall. Below are the conclusions derived from achieving the specific objectives outlined:

Selection of SQLi Detection Techniques

Various SQLi detection techniques based on computational intelligence were identified and selected, such as Naïve Bayes, logistic regression, random forests, and One-Class Support Vector Machines (OCSVM). The selection was based on both the relevance and effectiveness demonstrated by each technique in the scientific literature and the interests expressed by AizoOn Technology Consulting.

Database Structuring

A database was structured by combining public data from the "SQL Injection Dataset" publicly available on the Kaggle platform with specific data generated by AizoOn Technology Consulting's web application firewall, known as Mithril. Preprocessing and conditioning of the data were essential to ensure the quality and relevance of the information used in the detection models. It has been evidenced in the literature that data preprocessing is crucial to achieve a feature set that enables the construction of models with good performance.

It is important to note that from the statistical analysis conducted during the work, it was evident that the integration of the data should not have been performed. However,

this integration proved to be useful in the implementation of the machine learning models. This leads to the conclusion that given the nature of the data, the analysis in terms of the number of distinct words per sample does not fully determine the feasibility of integrating the two different data sources used.

Implementation and Training of Selected Techniques

The selected machine learning techniques were implemented and trained using the resulting integrated dataset. Hyperparameter tuning improved the models' performance and allowed the identification of the best configurations for each one, increasing detection capability and attempting to minimize false positives.

Evaluation and Comparison of Implemented Techniques

The techniques were evaluated and compared using performance metrics such as accuracy, precision, recall, and F1-Score. The results showed that the logistic regression model presented the best performance in all implemented scenarios, specifically achieving a precision of 99.45% and an F1-Score of 99.45% using the database with data balancing applied using the "Random Over Sampler" technique.

Recommendations for Implementation

Based on the findings detailed in this work, it is recommended to implement the logistic regression model in AizoOn Technology Consulting's web application firewall. This model not only demonstrated the highest precision in detecting SQL injection attacks, reaching a value of 99.45% but also showed an F1-Score of 99.45%, which is a metric of great importance in classification problems.

Department of Electronics and Telecommunications Engineering

Evaluation of SQL injection (SQLi) attack detection strategies in web applications using machine learning

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

INTERN: Santiago Taborda Echeverri

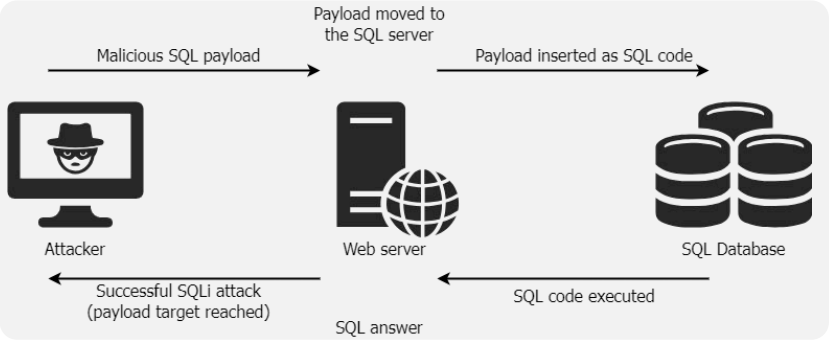
ADVISORS: Jaime Alberto Vergara Tejada, Jhonny Alexander Triana Maldonado

DEGREE: Ingeniería de Telecomunicaciones

Internship Semester: 2023-2

Abstract

SQL injection (SQLi) is one of the most critical vulnerabilities in web applications. It allows attackers to manipulate and execute malicious SQL queries through user input, compromising the integrity, confidentiality, and availability of the data stored in the database. This work evaluates various computational intelligence techniques, such as Naïve Bayes, logistic regression, random forests, and one-class support vector machines (OCSVM), for the detection of SQLi attacks with the aim of strengthening the security of the web application firewall of AizoOn Technology Consulting. A database was structured by combining public sources and specific data processed by the company's firewall, and the evaluation of the models included performance metrics to generate specific recommendations for their future implementation.



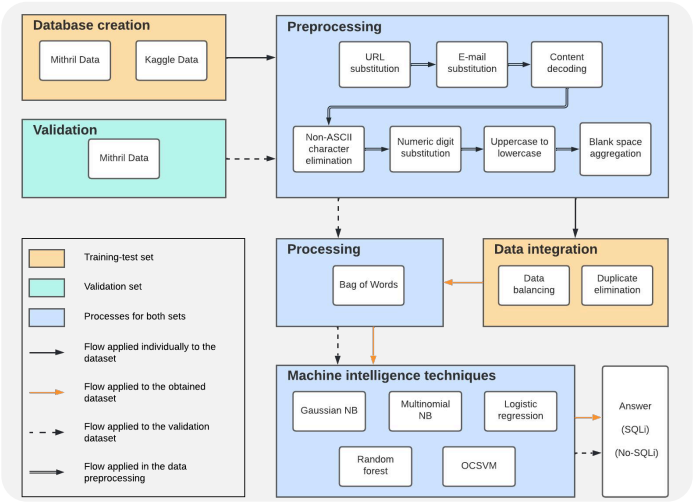
Methodology

Data Collection: Combination of data from the public SQL injection dataset available on Kaggle and data processed by AizoOn Technology Consulting's web application firewall (Mithril)

Data Preprocessing: Data cleaning included the removal of duplicates and balancing the dataset using oversampling and random undersampling techniques.

Model Selection: Various machine learning models were evaluated, including Naïve Bayes, Logistic Regression, Random Forest, and One-Class SVM.

Implementation: Python libraries such as statsmodels and imblearn were used for hypothesis testing and data balancing, respectively.



Objectives

- **Evaluate SQL injection attack detection strategies based on machine learning to strengthen the security of the web application firewall at AizoOn Technology Consulting.**
- Select SQLi detection techniques based on machine learning
- Structure a database by combining public sources and specific data processed by AizoOn Technology Consulting's web application firewall
- Implement and train the selected techniques.
- Evaluate and compare the implemented techniques using performance metrics to recommend the best option.

Conclusions

- Various SQLi attack detection techniques based on machine learning were identified and selected, such as Naïve Bayes, logistic regression, random forests, and one-class support vector machines (OCSVM).
- A database was successfully structured by combining public data from the "SQL Injection Dataset" on Kaggle with specific data processed by AizoOn Technology Consulting's web application firewall
- During the evaluation stage, the Logistic Regression model stood out as the most effective, achieving an accuracy and F1-score of 99.45% with the database when data balancing was implemented using the "Random Over Sampler" technique. In contrast, the OCSVM model showed the worst performance, with an accuracy of 50.69% and an F1-score of 66.79%.
- The findings recommend the use of the Logistic Regression model to enhance the security of AizoOn's web application firewalls against SQL injection attacks.

AUTHOR'S CONTACT INFORMATION: +57 3108983553 santiago.tabordae@udea.edu.co https://www.linkedin.com/in/santiago-taborda-e

Bibliography

[1] International Telecommunication Union (via World Bank), Gapminder, UN, HYDE, and Gapminder (Systema Globalis) – processed by Our World in Data. Number of internet users. https://ourworldindata.org/grapher/number-of-internet-users. Accessed: 2024.

[2] Gallup. Indicators. hybrid work. work locations for u.s. employees with remote-capable jobs. https://www.gallup.com/401384/indicator-hybrid-work.aspx. Accessed: 2024.

[3] The Open Worldwide Application Security Project (OWASP). Owasp foundation. https://owasp.org, 2001. Accessed: 2024.

[4] The Open Worldwide Application Security Project (OWASP). Owasp top ten. https://owasp.org/www-project-top-ten/, 2021. Accessed: 2024.

[5] Common Weakness Enumeration (CWE). Cwe community-developed. https://cwe.mitre.org/about/index.html, 2006. Accessed: 2024.

[6] Common Weakness Enumeration (CWE). 2023 cwe top 25 most dangerous software weaknesses. https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html, 2023. Accessed: 2024.

[7] Hazim Hanif, Mohd Hairul Nizam Md Nasir, Mohd Faizal Ab Razak, Ahmad Firdaus, and Nor Badrul Anuar. The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches. *Journal of Network and Computer Applications*, 179:103009, 2021.

[8] AizoOn Technology Consulting. Aizoon. https://www.aizoongroup.com/home.aspx#intro. Accessed: 2024.

[9] AizoOn Technology Consulting. About us. https://www.aizoongroup.com/aboutus.aspx#intro. Accessed: 2024.

[10] AizoOn Technology Consulting. Technology units - cyber security. https://www.aizoongroup.com/cyber_security.aspx#intro. Accessed: 2024.

[11] Abu Syeed Sajid Ahmed. Sql injection dataset, 2024.

[12] Catherine M. Ricardo. Structured query language. In Hossein Bidgoli, editor, *Encyclopedia of Information Systems*, pages 279–297. Elsevier, New York, 2003.

[13] Mario Heiderich, Eduardo Alberto Vela Nava, Gareth Heyes, and David Lindsay. Chapter 7 - sql. In Mario Heiderich, Eduardo Alberto Vela Nava, Gareth Heyes, and David Lindsay, editors, *Web Application Obfuscation*, pages 177–197. Syngress, Boston, 2011.

[14] Cornelia Gyorödi, Robert Gyorödi, and Roxana Sotoc. A comparative study of relational and non-relational database models in a web-based application. *International Journal of Advanced Computer Science and Applications*, 6(11):78–83, 2015.

[15] Cornelia Győrödi, Robert Győrödi, George Pecherle, and Andrada Olah. A comparative study: Mongodb vs. mysql. In *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*, pages 1–6, 2015.

[16] OWASP Foundation. kingthorin. Sql injection. https://owasp.org/www-community/attacks/SQL_Injection#. Accessed: 2024.

[17] Aditya Rai, MD. Mazharul Islam Miraz, Deshbandhu Das, Harpreet Kaur, and Swati. Sql injection: Classification and prevention. In *2021 2nd International Conference on Intelligent Engineering and Management (ICIEM)*, pages 367–372, 2021.

[18] Malik Qasaimeh Mohammed Nasereddin, Ashaar ALKhamaiseh and Raad Al-Qassas. A systematic review of detection and prevention techniques of sql injection attacks. *Information Security Journal: A Global Perspective*, 32(4):252–265, 2023.

[19] Ines Jemal, Omar Cheikhrouhou, Habib Hamam, and Adel Mahfoudhi. Sql injection attack detection and prevention techniques using machine learning. *International Journal of Applied Engineering Research*, 15(6):569–580, 2020.

[20] Common Weakness Enumeration (CWE) content team. Plover. Cwe-89: Improper neutralization of special elements used in an sql command ('sql injection'). https://cwe.mitre.org/data/definitions/89.html, 2006. Accessed: 2024.

[21] Fahad M Alotaibi and Vassilios G Vassilakis. Toward an sdn-based web application firewall: Defending against sql injection attacks. *Future Internet*, 15(5):170, 2023.

[22] Amirmohammad Sadeghian, Mazdak Zamani, and Suhaimi Ibrahim. Sql injection is still alive: A study on sql injection signature evasion techniques. In *2013 International Conference on Informatics and Creative Multimedia*, pages 265–268, 2013.

[23] Igor Tasevski and Kire Jakimoski. Overview of sql injection defense mechanisms. In *2020 28th Telecommunications Forum (TELFOR)*, pages 1–4, 2020.

[24] Zain Marashdeh, Khaled Suwais, and Mohammad Alia. A survey on sql injection attack: Detection and challenges. In *2021 International Conference on Information Technology (ICIT)*, pages 957–962, 2021.

[25] Balazs Pejo and Nikolett Kapui. Sqli detection with ml: A data-source perspective, 2023.

[26] Raed Abdullah Abobakr Busaeed, Wan Isni Sofiah Wan Din, Quadri Waseem, and Azlee Bin Zabidi. Taxonomy of sql injection: Ml trends & open challenges. In *2023 IEEE 8th International Conference On Software Engineering and Computer Systems (ICSECS)*, pages 382–387, 2023.

[27] Ignacio Samuel Crespo-Martínez, Adrián Campazas-Vega, Ángel Manuel Guerrero-Higueras, Virginia Riego-DelCastillo, Claudia Álvarez Aparicio, and Camino Fernández-Llamas. Sql injection attack detection in network flow data. *Computers & Security*, 127:103093, 2023.

[28] Prince Roy, Rajneesh Kumar, and Pooja Rani. Sql injection attack detection by machine learning classifier. In *2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, pages 394–400, 2022.

[29] Wei Zhang, Yueqin Li, Xiaofeng Li, Minggang Shao, Yajie Mi, Hongli Zhang, Guoqing Zhi, et al. Deep neural network-based sql injection detection method. *Security and Communication Networks*, 2022, 2022.

[30] Ahmed Abadulla Ashlam, Atta Badii, and Frederic Stahl. Multi-phase algorithmic framework to prevent sql injection attacks using improved machine learning and deep learning to enhance database security in real-time. In *2022 15th International Conference on Security of Information and Networks (SIN)*, pages 01–04, 2022.

[31] Eman Hosam, Hagar Hosny, Walaa Ashraf, and Ahmed S. Kaseb. Sql injection detection using machine learning techniques. In *2021 8th International Conference on Soft Computing & Machine Intelligence (ISCMI)*, pages 15–20, 2021.

[32] Ding Chen, Qiseng Yan, Chunwang Wu, and Jun Zhao. Sql injection attack detection and prevention techniques using deep learning. *Journal of Physics: Conference Series*, 1757(1):012055, jan 2021.

[33] Sushant Sharma, Pavol Zavarsky, and Sergey Butakov. Machine learning based intrusion detection system for web-based attacks. In *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, pages 227–230, 2020.

[34] Qi Li, Fang Wang, Junfeng Wang, and Weishi Li. Lstm-based sql injection detection method for intelligent transportation system. *IEEE Transactions on Vehicular Technology*, 68(5):4182–4191, 2019.

[35] Muhammad Noman Khalid, Humera Farooq, Muhammad Iqbal, Muhammad Talha Alam, and Kamran Rasheed. Predicting web vulnerabilities in web applications based on machine learning. In Imran Sarwar Bajwa, Fairouz Kamareddine, and Anna Costa, editors, *Intelligent Technologies and Applications*, pages 473–484, Singapore, 2019. Springer Singapore.

[36] Gustavo Betarte, Álvaro Pardo, and Rodrigo Martínez. Web application attacks detection using machine learning techniques. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1065–1072, 2018.

[37] Andrew R Webb, Keith D Copsey, and Gavin Cawley. *Statistical pattern recognition*, volume 2. Wiley Online Library, 2011.

[38] Yagang Zhang. *New advances in machine learning*. BoD–Books on Demand, 2010.

[39] Jiawei Han, Jian Pei, and Hanghang Tong. *Data mining: concepts and techniques*. Morgan kaufmann, 2022.

[40] Iqbal H Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN computer science*, 2(3):160, 2021.

[41] Maha Alghawazi, Daniyal Alghazzawi, and Suaad Alarifi. Detection of sql injection attack using machine learning techniques: A systematic literature review. *Journal of Cybersecurity and Privacy*, 2(4):764–777, 2022.

[42] Gangadhar Shobha and Shanta Rangaswamy. Chapter 8 - machine learning. In Venkat N. Gudivada and C.R. Rao, editors, *Computational Analysis and Understanding of Natural Languages: Principles, Methods and Applications*, volume 38 of *Handbook of Statistics*, pages 197–228. Elsevier, 2018.

[43] Daniel M. Rice. Chapter 4 - causal reasoning. In Daniel M. Rice, editor, *Calculus of Thought*, pages 95–123. Academic Press, 2014.

[44] Trevor J Hastie. Generalized additive models. In *Statistical models in S*, pages 249–307. Routledge, 2017.

[45] Kalidas Yeturu. Chapter 3 - machine learning algorithms, applications, and practices in data science. In Arni S.R. Srinivasa Rao and C.R. Rao, editors, *Principles and Methods for Data Science*, volume 43 of *Handbook of Statistics*, pages 81–206. Elsevier, 2020.

[46] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Madison, WI, 1998.

[47] Harry Zhang. The optimality of naive bayes. *Aa*, 1(2):3, 2004.

[48] Thomas W. Edgar and David O. Manz. Chapter 4 - exploratory study. In Thomas W. Edgar and David O. Manz, editors, *Research Methods for Cyber Security*, pages 95–130. Syngress, 2017.

[49] R.O. Sinnott, H. Duan, and Y. Sun. Chapter 15 - a case study in big data analytics: Exploring twitter sentiment analysis and the weather. In Rajkumar Buyya, Rodrigo N. Calheiros, and Amir Vahid Dastjerdi, editors, *Big Data*, pages 357–388. Morgan Kaufmann, 2016.

[50] Aleksandra Bartosik and Hannes Whittingham. Chapter 7 - evaluating safety and toxicity. In Stephanie Kay Ashenden, editor, *The Era of Artificial Intelligence, Machine Learning, and Data Science in the Pharmaceutical Industry*, pages 119–137. Academic Press, 2021.

[51] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[52] Klaus-Robert Müller, Sebastian Mika, Koji Tsuda, and Koji Schölkopf. An introduction to kernel-based learning algorithms. In *Handbook of neural network signal processing*, pages 4–1. CRC Press, 2018.

[53] Fa Zhu, Jian Yang, Cong Gao, Sheng Xu, Ning Ye, and Tongming Yin. A weighted one-class support vector machine. *Neurocomputing*, 189:1–10, 2016.

[54] Bernardo Damele Assumpcao Guimaraes and Miroslav Stampar. Sqlmap project. https://www.aizoongroup.com/home.aspx#intro, 2006-2024. Accessed: 2024.

[55] The Open Worldwide Application Security Project (OWASP). Wstg - latest. testing for sql injection. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05-Testing_for_SQL_Injection, frequently change. Accessed: 2024.

[56] The Open Worldwide Application Security Project (OWASP). Automated audit using sqlmap. https://wiki.owasp.org/index.php/Automated_Audit_using_SQLMap, 2013. Accessed: 2024.

[57] Frank Emmert-Streib and Matthias Dehmer. Understanding statistical hypothesis testing: The logic of statistical inference. *Machine Learning and Knowledge Extraction*, 1(3):945–961, 2019.

[58] Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[59] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explor. Newsl.*, 6(1):20–29, jun 2004.

[60] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[61] Mohammad Taher Pilehvar and Jose Camacho-Collados. *Embeddings in natural language processing: Theory and advances in vector representations of meaning*. Morgan & Claypool Publishers, 2020.

[62] Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pages 746–751, 2013.

[63] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[64] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[65] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[66] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. Hyperparameter optimization for machine learning models based on bayesian optimizationb. *Journal of Electronic Science and Technology*, 17(1):26–40, 2019.

[67] Shigeo Abe. *Support vector machines for pattern classification*, volume 2. Springer, 2005.

[68] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[69] Digna R Velez, Bill C White, Alison A Motsinger, William S Bush, Marylyn D Ritchie, Scott M Williams, and Jason H Moore. A balanced accuracy function for epistasis modeling in imbalanced datasets using multifactor dimensionality reduction. *Genetic Epidemiology: the Official Publication of the International Genetic Epidemiology Society*, 31(4):306–315, 2007.

[70] Ajay Kulkarni, Deri Chong, and Feras A. Batarseh. 5 - foundations of data imbalance and solutions for a data democracy. In Feras A. Batarseh and Ruixin Yang, editors, *Data Democracy*, pages 83–106. Academic Press, 2020.

[71] Ashish Tiwari. Chapter 2 - supervised learning: From theory to applications. In Rajiv Pandey, Sunil Kumar Khatri, Neeraj kumar Singh, and Parul Verma, editors, *Artificial Intelligence and Machine Learning for EDGE Computing*, pages 23–32. Academic Press, 2022.