Ensemble learning techniques against structured query language injection attacks

Ammar Odeh, Anas Abu Taleb
Department of Computer Science, Princess Sumaya University of Technology, Amman, Jordan

Article Info

Article history:

Received Mar 14, 2024
Revised Apr 8, 2024
Accepted Apr 13, 2024

Keywords:

Ensemble learning
Information security
Malicious query detection
SQL injection attacks
Web application vulnerability

ABSTRACT

Structured query language (SQL) injection threats pose severe risks to web applications, necessitating robust detection measures. This study introduced DSQLIA, employing ensemble learning algorithms-Bagging, Stacking, and AdaBoost classifiers-for SQL injection detection. Results unveiled the bagging classifier's 84% accuracy with perfect precision (100%) but moderate recall (68%). The stacking classifier achieved 85% accuracy, exceptional precision (99%), and balanced memory (72%), yielding an 83% F1-Score. Remarkably, the AdaBoost classifier outperformed, achieving 99% accuracy, high precision (98%), and outstanding recall (99%), leading to a remarkable 99% F1-Score. These findings highlight AdaBoost's superior ability to identify malicious queries with minimal false positives accurately. Overall, this research underscores the potential of ensemble learning in fortifying web application security against SQL injection attacks, emphasizing the AdaBoost classifier's exceptional performance in achieving precise and comprehensive detection.

Corresponding Author:

Ammar Odeh
Department of Computer Science, Princess Sumaya University of Technology
Amman 1196, Jordan
Email: a.odeh@psut.edu.jo

1. INTRODUCTION

Significant volumes of sensitive information belonging to corporations and institutions are housed within specialized databases, either within their own servers or overseas. Safeguarding this data from unauthorized entry is paramount across diverse industries [1], [2]. Any breach in this data could profoundly affect user privacy and the institution's reputation and financial stability. Consequently, data stands as one of the most critical assets necessitating preservation and protection. As a result, databases have evolved into vital repositories and fundamental components within contemporary organizations [3].

Structured query language (SQL) is the universally accepted language for interacting with and managing databases. Its functionality empowers database administrators to execute a wide array of operations on data, encompassing storage, retrieval, creation, updating, deletion, and more. Information is organized into tables that establish connections within databases, adhering to relational database (RDB) principles [4]. This framework delineates explicit relationships among data points, ensuring clarity in understanding the connections between tables and field types. These relationships are systematically defined within a schema, enhancing the ease of exploration within the data's interconnections [5].

Expanding on this concept, RDB thrive on well-defined structures, allowing efficient querying and manipulation of interconnected data. Examples of prominent RDB systems include Oracle, MySQL, and Microsoft SQL Server, each renowned for their robust capabilities in managing relational data structures effectively [6]. Initially identified in 1998, SQL injections persist as an unresolved and persistent concern affecting web applications and APIs even after over two decades. These vulnerabilities continue to threaten digital security significantly [7]. The open web application security project (OWASP) underscores the gravity of injection flaws by consistently featuring them in the Top 10 lists for web application security risks and API security threats, emphasizing the enduring relevance and critical nature of addressing these vulnerabilities [8].

Despite extensive efforts to fortify cybersecurity measures, SQL injections endure as a prevalent and serious threat within the digital landscape. Their persistent presence underscores the need for continued vigilance and advanced security protocols to mitigate their risks effectively. OWASP's consistent inclusion of injection flaws in their prominent security risk assessments emphasizes the imperative nature of combating these vulnerabilities to safeguard web applications and APIs from potential breaches [9].

Web applications face potential exploitation by attackers who leverage SQL statement injections or unique symbols via user inputs, aiming at the database layer. These attacks primarily aim to access valuable assets [10] illicitly. Vulnerabilities in the validation processes of these applications, often stemming from programming errors, offer attackers avenues to bypass authentication mechanisms. This, in turn, grants them unauthorized entry into databases, enabling the unauthorized retrieval or manipulation of data [10].

Recently, researchers, leveraging machine learning algorithms and deep neural network models, have proposed numerous detection approaches. Deep neural networks, a branch of machine learning often referred to as deep learning; represent a progressive area in artificial intelligence development. These models are designed to comprehend intricate patterns and representations in vast datasets. This ability has proven effective in deciphering various data types, spanning text, images, and audio. Deep learning holds promise in enhancing Web security, demonstrating broad potential across diverse applications [11]. However, a notable drawback of neural networks is their inclination toward overly confident predictions, even when those predictions may be incorrect [12]. SQL injection attacks (SQLIA) come in various forms, each exploiting distinct vulnerabilities in web applications that interact with databases. Figure 1 shows some common types of SQL Injection Attacks.

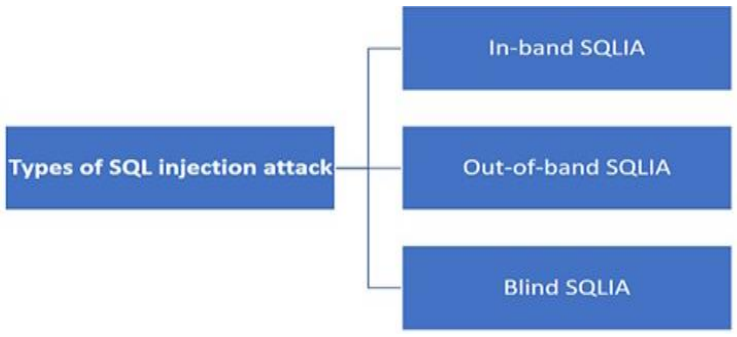

Figure 1. Types of SQL injection attack

The first type is In-band SQL Injection Attacks, also known as error-based and union-based attacks, involve using the same communication channel for both the attack and data retrieval. Error-based attacks exploit error messages returned by the database to extract information. For instance, an attacker might input SQL statements like ' OR 1=1; -- into a login form to provoke an error message disclosing sensitive database details. Union-based attacks involve the UNION SQL operator merging the results of two SELECT queries, enabling the attacker to obtain data from another table in the database [13]. Example SQL Statement:

```
SELECT * FROM Users WHERE Username = 'admin' OR 1=1; --' AND Password = 'password';
```

The second type is Out-of-band SQL Injection Attacks leverage a separate communication channel to extract data from the database. Attackers use these attacks when in-band techniques are not feasible due to firewall restrictions or other limitations. For instance, an attacker might insert malicious queries into the application, causing the database to perform DNS requests or trigger outbound HTTP requests to a controlled server, thereby exfiltrating sensitive data [14]. Example SQL Statement:

```
SELECT * FROM Users; EXEC xp_cmdshell 'nslookup attacker-controlled-domain.com';
```

The last type is Blind SQL Injection Attacks exploit the application's behavior in response to true or false conditions without directly extracting data from the database. Boolean-based attacks involve sending crafted SQL statements and analyzing the application's response to deduce the database's structure and data. Time-based attacks manipulate SQL queries to introduce time delays, revealing data through the application's response times [15]. Example SQL Statement:

```
SELECT * FROM Users WHERE Username = 'admin' AND (1=1) WAITFOR DELAY '0:0:10'; --';
```

2. THE PROPOSED METHODS

In the domain of intelligent transportation, Zhou and Wang [16] introduced an innovative model leveraging the Bayesian network algorithm for detecting XSS attacks. Their study proposed an ensemble learning approach incorporating domain expertise and threat intelligence, resulting in an exceptional accuracy rate of 96.9%. This accuracy surpasses several other algorithms, such as SVM, random forest, and decision tree. Li *et al.* [17] focused on an intelligent system to detect SQL injection by utilizing LSTM networks, achieving an accuracy of 91.79%. Their experimental findings emphasized the effectiveness of a sample positive generation method during data collection, mitigating overfitting issues and significantly enhancing SQL injection detection in transportation systems. Concurrently, Xie *et al.* [18] conducted research on SQL injection detection, employing an elastic-pooling convolutional neural network (EP-CNN) with approximately 4.48 million real-time web log data. Their study revealed that EP-CNN-based detection automatically identifies hidden patterns in SQL attacks, swiftly discerning malicious traffic while evading conventional SQL injection techniques.

Alarfaj and Khan [19] introduced a model that employs a probabilistic neural network (PNN) to identify instances of SQL injection attacks. To optimize the model's performance, the authors utilized the BAT algorithm, a metaheuristic optimization technique, to determine the most effective smoothing parameter. This study utilized a dataset containing 6,000 instances of SQL injections and 3500 normal queries. Features were extracted through tokenization and regular expression-based methods, followed by selection using Chi-Square testing. Kavitha *et al.* [20] introduced an approach to prevent SQL injection by employing four distinct classifiers: Support vector machines (SVM), artificial neural networks (ANN), boosted decision tree, and decision tree. This study utilized a database comprising 1100 vulnerable SQL injection samples. Comparative analysis among the classifiers revealed that the decision tree exhibited superior performance, albeit at the cost of significant processing time.

Muhammad and Ghafory [21] proposed an approach utilising a Naive Bayes classifier built upon role-based access control to address SQL injection. Their experimental findings demonstrated an accuracy of 93.3%, with precision at 1.0% and recall at 0.89%. However, this study specifically focused on a particular type of SQL injection attack. Dawadi *et al.* [22] introduced and implemented a Fuzzy neural network to construct an expert system for detecting SQL injection attacks. While effective, developing expert systems like this is intricate and time-consuming. Ndichu *et al.* [23] proposed REGEX, a regular expression filter approach, for detecting SQL injection attacks, employing a dataset of 20,474 queries. However, the limitation of this method lies in its inability to identify novel SQL injection attacks.

Baliarsingh *et al.* [24] authors introduced CODDLE, an approach employing a Convolutional Deep Neural Network to detect malicious injection attacks by encoding SQL/XSS symbols during the pre-processing stage. The experimental outcomes demonstrated an accuracy of 95%, with precision reaching 99% and a recall value of 92%. Conventional machine learning techniques often necessitate manual feature crafting, whereas deep learning models [25] can learn intricate hierarchical data representations autonomously. Moreover, deep learning exhibits scalability and excels particularly with extensive datasets, continuously enhancing performance with dataset expansion. Zhuo and Wang [16] introduced a novel detection approach utilizing long short-term memory (LSTM) and abstract syntax trees. This method effectively identifies SQLIA within raw query strings, even in scenarios where SQL detection bypasses occur. Inspired by the layered architecture of LSTM, Dawadi *et al.* [21] devised a multi-layered model for detecting various web attacks, achieving an 89.34% accuracy rate in SQLIA detection.

Recent studies have demonstrated the potential of machine learning in medical diagnostics. Anaraki *et al.* [26] utilized convolutional neural networks (CNNs) and genetic algorithms. In another study, Desai and Shah [27] applied multi-layer perceptron neural networks and CNNs, indicating the significant benefits of advanced neural network models. Gandhi *et al.* [28] proposed a hybrid detection mechanism known as CNN-Bi-LSTM , combining convolutional neural networks for feature extraction and Bidirectional LSTMs for capturing long-term data dependencies. This approach attained an experimental accuracy rate of 98%. Meanwhile, Li *et al.* [17]. introduced an SQLIA detection method based on adaptive deep forests [29], effectively addressing the degradation issue in deep forests' original features as layer numbers increase. Their experimental outcomes demonstrated an accuracy rate of 98.75%.

3. METHOD

Figure 2 illustrates the intricate system architecture meticulously crafted to combat the pervasive threat of malicious SQL injections using a cutting-edge ensemble learning approach. This holistic system blueprint embodies a strategic amalgamation of sophisticated components geared toward robust defense mechanisms. Delving deeper into the intricacies depicted in the figure, we discern three pivotal subsystems orchestrated to operate in tandem seamlessly: the data collection and preprocessing module, the ensemble models framework, and the model performance evaluation apparatus. Each of these subsystems plays a pivotal role in fortifying the system's resilience against evolving cyber threats, collectively forming a formidable bulwark safeguarding sensitive databases from nefarious exploits.

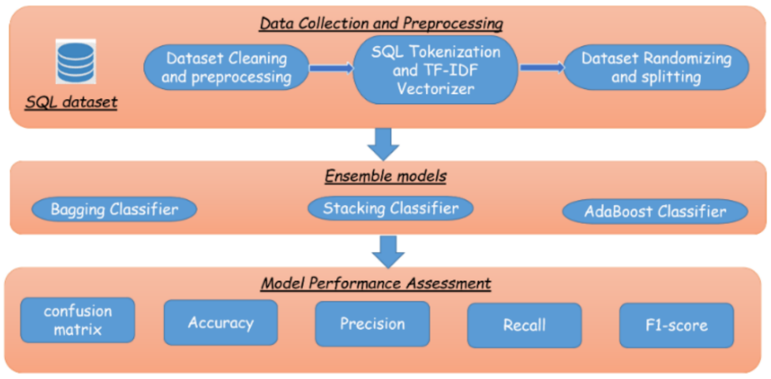
Data Collection and Preprocessing
SQL dataset
Dataset Cleaning and preprocessing
SQL Tokenization and TF-IDF Vectorizer
Dataset Randomizing and splitting
Ensemble models
Bagging Classifier
Stacking Classifier
AdaBoost Classifier
Model Performance Assessment
confusion matrix
Accuracy
Precision
Recall
F1-score

Figure 2. Proposed model architecture

3.1. Data collection and preprocessing

The initial phase of the system is dedicated to a series of pivotal steps aimed at priming the data for subsequent analysis and model training. Commencing with the acquisition of a dataset containing SQL statements, this research employed the Kaggle dataset [30] to train, evaluate, and compare the performance of the ensemble classifier. This dataset compilation involved the collection of diverse SQL injection queries from multiple websites, comprising a total of 30,919 SQL query statements structured around “SELECT FROM” and associated variations. Each statement within the dataset was tagged with a binary label, wherein a value of 1 denoted malicious queries, while 0 indicated benign queries.

Following data acquisition, the subsequent stage involves data cleaning, an essential process for identifying and managing inconsistencies, missing values, or noise present within the dataset. In addressing the creation of a balanced dataset, this step seeks to tackle class imbalance issues. Strategies encompassing oversampling, undersampling, or synthetic data generation are implemented to ensure a more equitable distribution among classes, distinguishing between malicious and non-malicious SQL statements. Figure 3 outlines the dataset before and after we apply the balancing process.

The subsequent phase in the proposed system centers around Tokenization for SQL Statements, a critical procedure responsible for segmenting SQL statements into individual tokens or words. Specifically tailored for SQL, this segmentation process disassembles SQL queries into distinct components such as keywords, table names, column names, and operators. This breakdown serves to streamline further analysis of the data.

Continuing the progression, the application of term frequency-inverse document frequency (TF-IDF) takes precedence. As a transformational technique, TF-IDF converts textual data, in this context, the tokenized SQL statements, into numerical vectors. TF-IDF's weighted approach prioritizes the importance of words within a document concerning an entire document collection, assigning higher weights to terms that exhibit significance within a particular document yet maintain lower frequency across the whole document corpus. This transformation process serves to prepare the tokenized SQL statements in a numerical format compatible with machine learning algorithms.

Upon completion of preprocessing and vectorization, the dataset undergoes segmentation into distinct training and testing subsets. While the training set is designated for machine learning model training, the testing set evaluates the models' performance. Employing randomization ensures unbiased representation and safeguards against the influence of inherent data order during model training and evaluation.

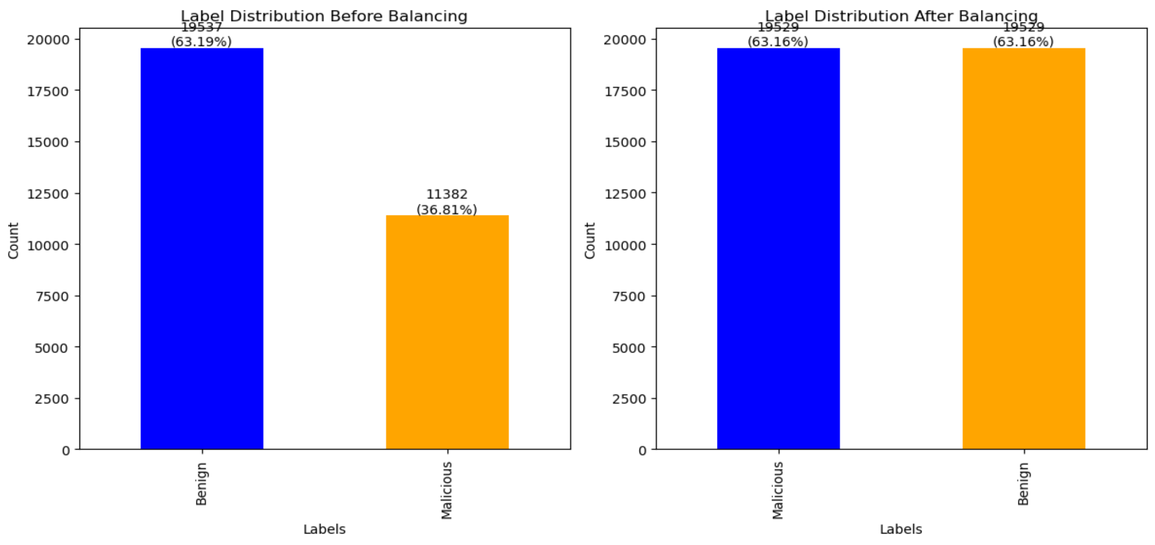

Figure 3. Data balance

3.2. Ensemble classifiers

To detect SQLIA, a series of three distinct ensemble classifiers were methodically employed to ascertain the most effective approach. The initial ensemble model adopted was the bagging classifier, employing decision tree as the base estimator. This classifier operates by creating multiple decision tree models, each trained on a subset of the dataset. Subsequently, these models contribute to the final prediction through aggregation, thereby mitigating overfitting and enhancing the overall robustness of the predictive outcome.

Following the bagging classifier, the stacking classifier was deployed, integrating both decision tree and random forest as base estimators, with logistic regression serving as the final estimator. In this ensemble configuration, the predictions from the base models, namely decision tree and random forest, are utilized as inputs for the logistic regression model, which learns to combine their predictions optimally. By leveraging the strengths of multiple algorithms, the stacking classifier endeavours to exploit diverse decision-making strategies, potentially resulting in improved performance and generalization capabilities.

Lastly, the AdaBoost classifier was introduced into the ensemble framework, employing decision tree as the base estimator with a maximum depth of 1. AdaBoost operates iteratively by assigning weights to incorrectly classified instances in each consecutive training iteration, allowing the subsequent model to focus on these misclassified instances. This iterative process facilitates the creation of a strong learner by sequentially emphasizing the previously misclassified data points, thereby enhancing the overall predictive accuracy and addressing complex decision boundaries effectively.

Each of these ensemble classifiers, from Bagging and Stacking to AdaBoost, offers unique advantages in harnessing the collective power of multiple models to detect intricate patterns indicative of SQL Injection Attacks. By leveraging the complementary strengths of diverse base estimators and ensemble strategies. These models collectively aim to fortify the detection capability and resilience against various SQLIA patterns present within the dataset.

3.3. Model performance assessment

Evaluating Ensemble classifiers is crucial to understanding their performance and recognizing their strengths and weaknesses. This research tested the model using a 5-fold cross-validation technique and assessed it with separate testing datasets. This comprehensive process thoroughly scrutinized and evaluated the model's effectiveness. Standard evaluation metrics were utilized to measure the detection model's efficiency throughout the training, validation, and testing phases. Figure 4 summarizes the standardized performance evaluation metrics.

4. RESULTS AND DISCUSSION

This section highlights the experimental outcomes of the system implemented in the Python environment. Figure 5 shows a confusion matrices for three ensemble machine learning classifiers: Bagging, Stacking, and AdaBoost. For the bagging classifier, the matrix indicates a high number of true positives and true negatives, with a count of 5.282 and 7.887, respectively, showing that the classifier can correctly identify both classes. The false positives are relatively fewer, recorded at 2.444, and the false negatives are minimal, at just 11. Figure 5(a) depicts the Bagging Classifier, which correctly identifies 5.282 True Positives (TP), 7.887 True Negatives (TN), 2444 False Positives (FP) and 11 False Negatives (FN). This classifier demonstrates a robust ability to accurately identify both positive and negative cases, with a particularly low rate of false negatives. In Figure 5(b), the Stacking Classifier is presented, featuring a count of 5.537 TP, 7841 TN, 2.189 FP and 57 FN. The matrix suggests that this classifier performs well in identifying true positives and true negatives, although it experiences a moderate number of false positives when compared to the Bagging Classifier. Figure 5(c) displays the AdaBoost classifier, which has a TP count of 7.652, TN count of 7.739, FP count of 74 and FN count of 159. This classifier is exemplary in minimizing false positives while still maintaining high accuracy in detecting true positives and negatives, indicating its effective application in scenarios where reducing false alarms is critical.

The stacking classifier also exhibits a robust performance with 5.537 true positives and 7.841 true negatives, suggesting its effectiveness in classification. The false positives count is 2.189, and false negatives are 57, slightly higher than the bagging classifier but still indicating a robust predictive ability. Lastly, the AdaBoost classifier shows the highest number of true positives and true negatives among the three, with 7.652 and 7.739, respectively, indicating its potent classification strength. It has the lowest false positives and false negatives, at 74 and 159, respectively, demonstrating a very high prediction accuracy. Table 1 provides a comprehensive summary detailing the performance of the three ensemble models concerning essential evaluation metrics, including accuracy, precision, recall, and F1-Score.

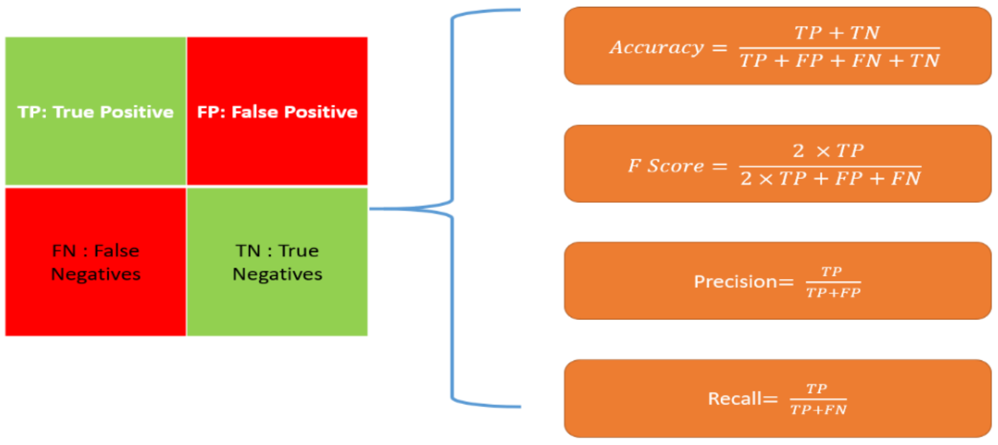
TP: True Positive
FP: False Positive
FN : False Negatives
TN : True Negatives
$Accuracy = \frac{TP+TN}{TP+FP+FN+TN}$
$F\ Score = \frac{2 \times TP}{2\times TP+FP+FN}$
Precision= $\frac{TP}{TP+FP}$
Recall= $\frac{TP}{TP+FN}$

Figure 4. Standard performance assessment indicators

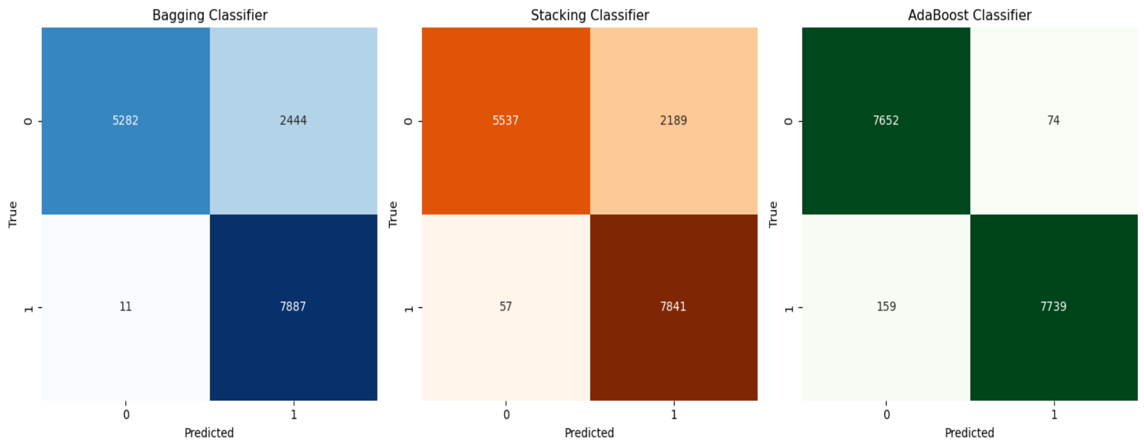

Figure 1. Confusion matrices for three different ensemble machine learning classifiers: (a) bagging, (b) stacking, and (c) AdaBoost from left to right

Table 1. Performance metrics for the proposed models

	Accuracy	Precision	Recall	F1-Score
Bagging classifier	84%	100%	68%	81%
Stacking classifier	85%	99%	72%	83%
AdaBoost classifier	99%	98%	99%	99%

Table 1 outlines the performance evaluation metrics, including accuracy, precision, recall, and F1-Score, for three distinct classifiers utilized in detecting SQL injection attacks. The bagging classifier achieved an accuracy of 84%. It demonstrated perfect precision (100%), indicating that all instances identified as malicious were correct. However, its recall stood at 68%, suggesting a moderate ability to detect actual malicious instances among all positives. The combined F1-Score for precision and recall was computed at 81%.

Meanwhile, the stacking classifier showcased a slightly higher accuracy at 85%. It maintained a high precision rate of 99%, implying minimal false positives in its predictions. However, its recall rate was measured at 72%, indicating a lower ability to capture all actual malicious instances. The resultant F1-Score reached 83%, indicating a good trade-off between precision and recall.

In contrast, the AdaBoost classifier outperformed the others with an accuracy of 99%. It exhibited a high precision of 98%, signifying a low rate of false positive predictions. Moreover, its recall rate stood at 99%, demonstrating a robust capability to identify malicious instances. The resultant F1-Score reached an impressive 99%, denoting an outstanding overall performance by combining precision and recall. These metrics offer a comprehensive understanding of the individual classifiers' capabilities in accurately identifying SQL injection attacks while also shedding light on their respective trade-offs between precision and recall.

Table 2 presents a comprehensive comparison of several prominent machine learning algorithms along with their corresponding accuracy rates. Our proposed AdaBoost classifier stands out prominently among the array of algorithms in terms of accuracy, boasting a remarkable 99% accuracy rate. When compared to other prominent models like the Bayesian network algorithm with 97%, LSTM with 92%, Elastic-pooling convolutional neural network with 98.7%, Naive Bayes classifier with 93.3%, Convolutional deep neural network with 95%, CNN-Bi-LSTM with 98%, and even 'deep forests' with 99%, our AdaBoost classifier surpasses them all. Its ability to consistently achieve such a high accuracy rate positions it as a superior choice, demonstrating its robustness and efficacy in handling complex data sets with exceptional precision.

Table 2. Comprehensive comparison of algorithms for SQL injection techniques

Source	Model	Accuracy
Zhou and Wang [16]	Bayesian network algorithm	97%
Li *et al.* [17]	LSTM	92%
Xie *et al.* [18]	Elastic-pooling convolutional neural network	97.7
Muhammad and Ghafory [21]	Naive Bayes classifier	93.3
Baliarsingh *et al.* [24]	Convolutional deep neural network	95%
Nasereddin *et al.* [28]	CNN-Bi-LSTM	98%
Krishnan *et al.* [29]	deep forests'	98%
Proposed Model	AdaBoost classifier	99%

5. CONCLUSIONS

SQL injection attacks represent a critical security threat to web applications, demanding robust detection mechanisms. In this study, we introduced DSQLIA, an innovative SQL injection detection model harnessing ensemble learning algorithms. Specifically, we integrated three prominent algorithms - bagging classifier employing decision tree, stacking classifier combining decision tree and random forest, and AdaBoost classifier utilizing decision tree with a maximum depth of 1. The simulation results unveil notable insights into the performance of each classifier within our model. The bagging classifier demonstrated an accuracy of 84%, exhibiting perfect precision (100%) in correctly identifying malicious instances but displaying a moderate recall (68%), indicating room for improvement in detecting all actual malicious queries. Its combined F1-Score reached 81%, reflecting the balance between precision and recall.

Meanwhile, the stacking classifier showed slightly higher accuracy (85%) with outstanding precision (99%) and a trade-off in recall (72%). This classifier achieved an F1-Score of 83%, indicating a commendable equilibrium between precision and recall rates. In stark contrast, the AdaBoost classifier emerged as the top-performing model, boasting an exceptional accuracy of 99%. It showcased high precision (98%) and outstanding recall (99%), indicating remarkable capability to identify malicious instances accurately. The resulting F1-Score of 99% underscores its outstanding overall performance by harmonizing precision and recall. These metrics comprehensively evaluate each classifier's capacity to identify SQL injection attacks effectively. Additionally, they shed light on the trade-offs between precision and recall, emphasizing the AdaBoost classifier's superiority in achieving exceptional detection accuracy with minimal false positives and a robust ability to capture malicious queries. The findings from our model serve as a stepping-stone towards bolstering web application security against SQL injection attacks, showcasing the potential of ensemble learning in enhancing detection systems.

REFERENCES

[1] T. M. Allam, "Estimate the performance of cloudera decision support queries," *International journal of online and biomedical engineering*, vol. 18, no. 1, pp. 127–138, Jan. 2022, doi: 10.3991/ijoe.v18i01.27877.

[2] M. S. A. El-Seoud, H. F. El-Sofany, and I. A. T. F. Taj-Eddin, "Mobile applications and semantic-web: A case study on automated course management," *International Journal of Interactive Mobile Technologies*, vol. 10, no. 3, pp. 42–53, Jul. 2016, doi: 10.3991/ijim.v10i3.5770.

[3] M. S. Al Reshan, "IoT-based application of information security triad," *International Journal of Interactive Mobile Technologies*, vol. 15, no. 24, pp. 61–76, Dec. 2021, doi: 10.3991/IJIM.V15I24.27333.

[4] M. L. E. Chang and H. N. Chua, "SQL and NoSQL database comparison: from performance perspective in supporting semi-structured data," in *Advances in Intelligent Systems and Computing*, vol. 886, Springer International Publishing, 2019, pp. 294–310.

[5] L. Sheldon, "Exploring the experiences that prompt data literacies: a mixed methods research study," *The University of Arizona*, 2022.

[6] J. F. Sequeda, W. J. Briggs, D. P. Miranker, and W. P. Heideman, "A Pay-as-you-go methodology to design and build enterprise knowledge graphs from relational databases," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11779 LNCS, Springer International Publishing, 2019, pp. 526–545.

[7] R. U. Rasool, H. F. Ahmad, W. Rafique, A. Qayyum, and J. Qadir, "Security and privacy of internet of medical things: A contemporary review in the age of surveillance, botnets, and adversarial ML," *Journal of Network and Computer Applications*, vol. 201, p. 103332, May 2022, doi: 10.1016/j.jnca.2022.103332.

[8] Sunardi, I. Riadi, and P. A. Raharja, "Vulnerability analysis of E-voting application using open web application security project (OWASP) framework," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 11, pp. 135–143, 2019, doi: 10.14569/IJACSA.2019.0101118.

[9] H. A. Noman and O. M. F. Abu-Sharkh, "Code injection attacks in wireless-based internet of things (IoT): A comprehensive review and practical implementations," *Sensors*, vol. 23, no. 13, p. 6067, Jun. 2023, doi: 10.3390/s23136067.

[10] N. F. Syed, S. W. Shah, R. Trujillo-Rasua, and R. Doss, "Traceability in supply chains: A cyber security analysis," *Computers and Security*, vol. 112, p. 102536, Jan. 2022, doi: 10.1016/j.cose.2021.102536.

[11] Z. Alkhalil, C. Hewage, L. Nawaf, and I. Khan, "Phishing attacks: A recent comprehensive study and a new anatomy," *Frontiers in Computer Science*, vol. 3, Mar. 2021, doi: 10.3389/fcomp.2021.563060.

[12] J. Gawlikowski *et al.*, "A survey of uncertainty in deep neural networks," *Artificial Intelligence Review*, vol. 56, no. S1, pp. 1513–1589, Jul. 2023, doi: 10.1007/s10462-023-10562-9.

[13] N. Singh and P. Tiwari, "SQL injection attacks, detection techniques on web application databases," in *Lecture Notes in Networks and Systems*, vol. 434, Springer Nature Singapore, 2022, pp. 387–394.

[14] A. Rai, M. M. I. Miraz, D. Das, H. Kaur, and Swati, "SQL injection: classification and prevention," in *Proceedings of 2021 2nd International Conference on Intelligent Engineering and Management, ICIEM 2021*, Apr. 2021, pp. 367–372, doi: 10.1109/ICIEM51511.2021.9445347.

[15] J. R. Dora, L. Hluchý, and K. Nemoga, "Ontology for blind SQL injection," *Computing and Informatics*, vol. 42, no. 2, pp. 480–500, 2023, doi: 10.31577/cai_2023_2_480.

[16] Y. Zhou and P. Wang, "An ensemble learning approach for XSS attack detection with domain knowledge and threat intelligence," *Computers and Security*, vol. 82, pp. 261–269, May 2019, doi: 10.1016/j.cose.2018.12.016.

[17] Q. Li, F. Wang, J. Wang, and W. Li, "LSTM-based SQL injection detection method for intelligent transportation system," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4182–4191, 2019, doi: 10.1109/TVT.2019.2893675.

[18] X. Xie, C. Ren, Y. Fu, J. Xu, and J. Guo, "SQL injection detection for web applications based on elastic-pooling CNN," *IEEE Access*, vol. 7, pp. 151475–151481, 2019, doi: 10.1109/ACCESS.2019.2947527.

[19] F. K. Alarfaj and N. A. Khan, "Enhancing the performance of SQL injection attack detection through probabilistic neural networks," *Applied Sciences (Switzerland)*, vol. 13, no. 7, p. 4365, Mar. 2023, doi: 10.3390/app13074365.

[20] M. Kavitha, V. Vennila, G. Padmapriya, and A. R. Kannan, "Prevention of SQL injection attack using unsupervised machine learning approach," *International Journal of Aquatic Science*, vol. 12, no. 03, pp. 1413–1424, 2021.

[21] T. Muhammad and H. Ghafory, "SQL injection attack detection using machine learning algorithm," *Mesopotamian Journal of CyberSecurity*, vol. 2022, pp. 5–17, Feb. 2022, doi: 10.58496/MJCS/2022/002.

[22] Dawadi, B.R.; Adhikari, B.; Srivastava, D.K. Deep Learning Technique-Enabled Web Application Firewall for the Detection of Web Attacks. Sensors 2023, 23, 2073

[23] S. Ndichu, S. Kim, S. Ozawa, T. Misu, and K. Makishima, "A machine learning approach to detection of JavaScript-based attacks using AST features and paragraph vectors," *Applied Soft Computing Journal*, vol. 84, p. 105721, 2019, doi: 10.1016/j.asoc.2019.105721.

[24] S. K. Baliarsingh, S. Vipsita, A. H. Gandomi, A. Panda, S. Bakshi, and S. Ramasubbareddy, "Analysis of high-dimensional genomic data using MapReduce based probabilistic neural network," *Computer Methods and Programs in Biomedicine*, vol. 195, p. 105625, Oct. 2020, doi: 10.1016/j.cmpb.2020.105625.

[25] E. H. A. Rady and A. S. Anwar, "Prediction of kidney disease stages using data mining algorithms," *Informatics in Medicine Unlocked*, vol. 15, p. 100178, 2019, doi: 10.1016/j.imu.2019.100178.

[26] A. K. Anaraki, M. Ayati, and F. Kazemi, "Magnetic resonance imaging-based brain tumor grades classification and grading via convolutional neural networks and genetic algorithms," *Biocybernetics and Biomedical Engineering*, vol. 39, no. 1, pp. 63–74, Jan. 2019, doi: 10.1016/j.bbe.2018.10.004.

[27] M. Desai and M. Shah, "An anatomization on breast cancer detection and diagnosis employing multi-layer perceptron neural network (MLP) and Convolutional neural network (CNN)," *Clinical eHealth*, vol. 4, pp. 1–11, 2021, doi: 10.1016/j.ceh.2020.11.002.

[28] N. Gandhi, J. Patel, R. Sisodiya, N. Doshi, and S. Mishra, “A CNN-BiLSTM based approach for detection of SQL injection attacks,” In *Proceedings of the 2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE),* Dubai, United Arab Emirates, 17–18 March 2021; pp. 378–383.
[29] A. K. S.S., “SQL injection detection using machine learning,” *Revista Gestão Inovação e Tecnologias*, vol. 11, no. 3, pp. 300–310, Jun. 2021, doi: 10.47059/revistageintec.v11i3.1939.
[30] Sajid, “SQL injection dataset,” 2021. https://www.kaggle.com/datasets/sajid576/sql-injection-dataset (accessed Apr. 27, 2023).

BIOGRAPHIES OF AUTHORS

Ammar Odeh received his Ph.D. from University of Bridgeport (UB), USA, in 2015. He is an Associate Professor at the Department of Computer Science, Faculty of King Hussein School of Computing Sciences, Princess Sumaya University for Technology, Amman, Jordan. His research interests include cybersecurity and cryptography, the internet of things (IoT). He can be contacted at email: a.odeh@psut.edu.jo.

Anas Abu Taleb is an associate professor in the Department of Computer Science at Princess Sumaya University for Technology, Amman, Jordan. He received a Ph.D. in Computer Science from the University of Bristol, UK 2010, an MS.c. in Computer Science from the University of the West of England, UK, 2007 and a BS.c. Degree in Computer Science from Princess Sumaya University for Technology, Jordan, 2004. Dr. Abu Taleb has published several journal and conference papers on sensor networks. In addition to sensor networks, Dr. Abu Taleb is interested in network fault tolerance, routing algorithms, and mobility models. He can be contacted at email: a.abutaleb@psut.edu.jo.