Empirical Evaluations of Machine Learning Effectiveness in Detecting Web Application Attacks

Muhusina Ismail[1], Saed Alrabaee[1(✉)], Saad Harous[2], and Kim-Kwang Raymond Choo[3]

[1] Department of Information Systems and Security, CIT, United Arab Emirates University, Al Ain, UAE
{201990139,salrabaee}@uaeu.ac.ae

[2] Department of Computer Science, College of Computing and Informatics, University of Sharjah, Sharjah, UAE
harous@sharjah.ac.ae

[3] Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, USA
raymond.choo@fulbrightmail.org

Abstract. Web applications remain a significant attack vector for cybercriminals seeking to exploit application vulnerabilities and gain unauthorized access to privileged data. In this research, we evaluate the efficacy of eight supervised machine learning algorithms - Naive Bayes, Decision Tree, AdaBoost, Random Forest, Logistic Regression, K-Nearest Neighbor (KNN), Support Vector Machine (SVM), and Artificial Neural Network (ANN) - in detecting and countering web application attacks. Our results indicate that KNN and Random Forest classifiers achieve an accuracy rate of 89% and an area under the curve of 94% on the CSIC HTTP dataset, a commonly used benchmark in the field. Meanwhile, the Naive Bayes classifier proves the most efficient, taking the least computational time when differentiating between malicious and benign HTTP requests. These findings may help direct future efforts towards more efficient, machine learning-driven defenses against web application attacks.

Keywords: Web Vulnerabilities · Web Attacks · Machine Learning

1 Introduction

Web applications (or apps) are pervasive in our current society, ranging from e-commerce to e-government, and so on. These applications enable organizations to collect, analyze, and store confidential user information such as credit card numbers and social security records. These applications interact with online databases, presenting data to users dynamically via a web server application. However, web applications with sensitive data can be targeted by cybercriminals, for example by exploiting vulnerabilities to directly manipulate data on various

D. Perakovic and L. Knapcikova (Eds.): FABULOUS 2023, LNICST 542, pp. 99–116, 2024.
https://doi.org/10.1007/978-3-031-50051-0_8

websites [1]. High-risk web applications include those handling large volumes of user data. Attack methods, such as SQL injection and cross-site scripting (XSS), identified by OWASP as particularly dangerous, can lead to users being directed to phishing pages. Despite existing efforts to secure web applications [2–10], this topic remains a research challenge; hence, the focus of this paper.

Specifically, we introduce a comparative framework for detecting SQL injection and XSS attacks, using eight supervised machine learning algorithms (i.e., Naive Bayes, Decision Tree, AdaBoost, Random Forest, Logistic Regression, K-Nearest Neighbor (KNN), Support Vector Machine (SVM), and Artificial Neural Network (ANN)). We employ seven performance metrics to evaluate these eight algorithms, namely: accuracy, precision, recall, F1-score, ROC, AUC, and computation time. Our model parses HTTP requests to extract features, generates a hashmap key, and establishes a whitelist of standard HTTP queries during learning. Request parameters are encoded as a feature vector, following the technique explained by Kozik et al. [11], and transformed into constant-length histograms to train a classifier. In doing so, we seek to gain a better understanding of the accuracy of existing machine learning approaches in attack detection, and identifying effective algorithm(s) in detection.

In the next section, we'll discuss related work, followed by a brief overview of the essential background materials.

2 Related Work

Signature-based methods such as SCALP, PHP-IDS, and Snort are commonly used to detect web application attacks because they can process vast volumes of data [12,13]. These techniques can quickly match text patterns using specific algorithms, typically PCRE standard phrases [14]. However, they require expertise to define attack patterns and struggle with concealed SQL injection attacks using URL encoding. Kozik et al. proposed using data from client-to-web-server HTTP requests to detect web attacks [15]. Their approach employs a machine learning model with two stages: learning and assessment. The learning phase uses labeled data to model normal machine operation. In the assessment phase, HTTP requests are encoded into vectors providing full request details. This method has been applied to HTTP request data, like those in the Apache Connection Log, from traffic generated by e-commerce web applications, identifying various web application cyberattacks.

S.Sharma et al. used machine learning for intrusion detection, and found the J48 decision tree algorithm to be most effective in terms of True Positive Rate, Precision, and Recall [16]. Oumaima, Chakir, et al. evaluated the efficiency of certain machine learning algorithms for intrusion detection [17]. Offutt et al. demonstrated the effectiveness of Bypass Testing to validate application inputs [18]. Sun et al. showed the feasibility of using system conversion and machine learning to detect in-process cyber-attacks on web applications running on a Java Virtual Machine [19]. M. Cova et al. proposed a strategy to detect and study JavaScript code abuses that lead to drive-by-download threats. They combined malicious JavaScript code for anomaly detection and auto-identification,

and used a HTML unit browser to serve web pages [20]. They developed a method that utilized standard JavaScript code to introduce features using various feature and machine learning techniques.

XSnare, an XSS solution extension for Firefox, uses prior knowledge of web application HTML content to thwart XSS attacks [21]. It incorporates a database with exploit descriptions informed by past CVEs. XSSDS is another system that passively identifies successful XSS attacks through HTTP traffic monitoring [22]. DeepXSS, a deep learning-based system for XSS detection, extracts XSS features' payload using word2vec, mapping each payload to a feature vector [23]. It then employs Long Short-Term Memory (LSTM) recurrent neural networks for training and testing the detection model. Such research informs further efforts to detect and mitigate XSS threats using artificial intelligence methods [24]. G. Kaur et al. proposed a blind XSS detection method using machine learning, notably the Linear Support Vector Machine (LSVM) classifier, to identify potential threats and distinguish between cached and blind XSS [25].

SQLIDS is a real-time SQL injection detection system that monitors Java-based applications [26]. Zhang et al. proposed an adaptive random testing aimed at triggering SQL injections within limited attempts [27]. SEPTIC and SPHERES are systems designed for preventing DBMS attacks and analyzing web traffic respectively, with the latter applying specific filtering rules to requests, website structures, and user sessions [28,29]. Zhuo et al. and Li et al. introduced systems that use long short-term memory and deep forest-based methods to detect SQL Injection Attacks [30,31]. DIAVA is a traffic-based system for detecting SQL injection attacks and analyzing vulnerabilities [32]. Batista et al. proposed a fuzzy rules set for constructing expert systems in cybernetic data attacks, specifically SQL Injection attacks [33]. WOVSQLI and DeepSQLi are systems utilizing word vectors of SQL tokens, long short-term memory neural networks, and deep natural language processing to identify SQL injection attacks and create test cases for detection [34,35]. Chen et al. recently introduced an SQL injection detection system utilizing a natural language processing model and deep learning framework, which does not rely on previous rules [36]. A comparison of these studies using the HTTP CSIC 2010 dataset is provided in Table 1.

Table 1. Comparison of some related work

Study	Technique	Dataset	Problem Domain	Evaluation Metrics
[37]	Generic-Feature-Selection (GeFS)	CSIC 2010 HTTP Dataset	Web Applications Attack	Accuracy
[15]	Naive Bayes, AdaBoost, Part and J48	CSIC 2010 HTTP Dataset	Web Applications Attack	False Positive Rate
[38]	Naïve Bayes, Bayes network, decision stump RBF Network	ECML-PKDD 2007 HTTP, CSIC HTTP 2010	Web Applications Attack	False Positive Rate
[16]	J48, Naïve Bayes, OneR, Decision table	CSIC 2010 HTTP Dataset	Web Applications Attack	Precision, Recall, Accuracy, and F-measure metrics
[17]	KNN, Decision Tree, Multinomial, and Bernoulli Naive Bayes, SVM Linear, Sigmoid, and RBF	CSIC 2010 HTTP Dataset	Web Applications Attack	Accuracy, Recall, Precision, F-value, FPR, and FNR

3 Background

3.1 XSS: Cross-Site Scripting

Cross-site scripting (XSS) is a common hacking technique where malicious code, often JavaScript, is injected into web applications, affecting user's browsers [39, 40]. This can spread via legitimate pages, with XSS attacks categorized as non-persistent or persistent.

Non-Persistent XSS Attack. In a non-persistent XSS attack, users click a malicious link created by hackers, triggering the execution of the attacker's vulnerable code in their browser. Listing 1 demonstrates a typical attack.

```
<?php
$name = $_GET['name'];
echo "Hello Welcome $name <br>";
echo "<a href="http://xssattackdemo.com/">Click to Download
   </a>";
?>
index.php? name = guest<script>alert ('Hacked') </script>
```

Listing 1.1. Script for non-persistent XSS attack

Persistent XSS Attack. In this type of attack, the attacker injects a malicious script into a repository. The malicious string originates from the website's database. This can be done by having the victim's browser parse the following HTML code:

```
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
```

Listing 1.2. Script for a persistent XSS attack

In this form of XSS attack, a script redirects the user's browser to a URL that sends the victim's cookies to the attacker's server. The attacker can then impersonate the victim using these cookies, launching further attacks. This persistent type of XSS attack, aimed at exploiting a website's vulnerability to steal cookies, is generally more damaging than non-persistent ones.

3.2 SQL Injection Attack

In SQL injection attacks, an attacker uses a web page input to introduce harmful SQL commands into an SQL statement, compromising the web system's security. These malicious commands can modify an SQL statement, leading to unauthorized access or other adverse effects. The attack might involve inserting malicious code directly into user input variables or storing it within certain database tables, which is later compiled and added to a dynamic SQL command [41]as displayed in Listing 3.

```
EmployeeDetails = Request. Form ("EmployeeDetails ");
var sql = "select * from OrdersTable where EmployeeDetails =
    '" + EmployeeDetails + "'";
```

Listing 1.3. Script for a basic SQL Injection Attack

The following describes an example of an SQL injection attack. First, the user is asked to write down the name of an employee. When a user enters Steve as the employee name, the query will be a collected script as shown in Listing 4.

```
Select * FROM OrdersTable WHERE EmployeeDetails = 'Steve'
```

Listing 1.4. Query for retrieving details from table

The script merges hardcoded and user-inserted strings to generate an SQL query, retrieving specific employee details. SQL injections typically target URL styles like $http : /www.targetdb.com/index.php?id = 2$. Attackers use certain inputs, like single or double quotes or Boolean expressions, to provoke an SQL Injection attack. For example, a MySQL-backed webpage susceptible to SQL injection might display a data error message like $mysql_num_rows()$ or $mysql_fetch_array()$ on the webpage, indicating vulnerability to such attacks.

4 System Methodology

4.1 Design Choice

This study proposes a method for detecting web-based application attacks using supervised machine learning algorithms, given the availability of labeled data. The approach is applicable to various real-world scenarios like facial recognition, healthcare, Siri, weather forecasting, spam filtering, and fraud detection. The methodology utilizes eight different supervised classification machine learning algorithms to predict attacks. The workflow of the proposed method and details about the utilized algorithms will be elaborated further.

This study implements eight machine learning classification algorithms: Naive Bayes, Decision Tree, AdaBoost, Random Forest, Logistic Regression, K-Nearest Neighbor (KNN), Support Vector Machine (SVM), and Artificial Neural Network (ANN). These fall under three categories: Probabilistic, Tree, and Miscellaneous. The probabilistic category includes Naive Bayes and Logistic Regression, both leveraging the principle of probability. Naive Bayes, a powerful classifier, is based on Bayes' Theorem and assumes feature independence within classes [42]. Logistic Regression, a predictive mathematical model, estimates binary reliability parameters. It predicts the probability of target variable belonging to a particular category, thus effectively used for various analytical problems such as disease detection and malicious attack detection [43]. Despite its simplicity, it has a broad range of applications including spam detection, injection attack detection, XSS attacks detection, etc.

The study also employs tree-based machine learning algorithms: Decision Tree, Random Forest, and AdaBoost. AdaBoost, or Adaptive Boosting, is a popular machine learning technique that collaboratively develops a model, learning from previous errors, and works particularly well with Decision Trees [44,45]. The Decision Tree is a supervised algorithm that can be used for both classification and regression tasks. It employs a tree-like model of decisions, where each node represents a feature, each branch signifies a decision rule, and each leaf node represents an outcome [46]. Decision Trees are often favored for their ease of interpretation and decision-making process representation. In this study, the Decision Tree Classifier model is utilized due to its predictive capability and its ability to evaluate potential outcomes effectively.

The Random Forest (RF) classifier is a supervised machine learning algorithm used in this study to detect web application attacks [47,48]. This ensemble method creates a robust forest by averaging results from multiple decision trees, effectively preventing over-fitting and providing faster results. Other machine learning algorithms employed include the K Nearest Neighbor (k-NN), Support Vector Machine (SVM), and Artificial Neural Network (ANN) [49]. The k-NN algorithm, a non-parametric method used for classification and regression, predicts class membership based on the closest instances in the feature space. The performance of this method depends on whether it is used for isolation or regression.

The K-Nearest Neighbor (k-NN) algorithm is a model-based, or lazy learning method, used mainly for prediction of categorical values. This non-parametric algorithm assigns an item to the class of its most frequent nearest neighbor, with the item's value being the output of the k-NN regression, the value of the closest neighbor. It uses all available data for training and makes no assumptions about the underlying data, and it was used in this study to detect XSS and SQL injection attacks in web applications [50]. The Support Vector Machine (SVM) is a robust supervised machine learning model, introduced in the 1960 s and further refined in the 1990 s [51,52]. This algorithm, effective for both classification and regression, separates classes in multidimensional space via a hyperplane, iteratively refined to reduce errors and avoid overfitting. SVM uses kernels (linear, sigmoid, and polynomial in this study) to transform input data into the desired form, its main goal being the identification of the maximum marginal hyperplane that divides the dataset into classes.

Lastly, the Artificial Neural Network (ANN) [42,53,54], also called a neural network, is a group with a perceptron/neuron in each layer. It is a computational model made up of three layers, generally consisting of an input layer, an output layer, and a hidden layer (which is between the input and output layers). Inputs from the dataset are given to the input layers, which then go through the hidden layers for further processing. The output layer will then generate the results. This network is capable of learning nonlinear functions with the help of an activation function. This can be the ANN's powerhouse. In our study, we used the backpropagation neural network, which works by internally adjusting the weights with a non-linear connection between the input and output. To process the

information and to produce consequential results, it has a substantial processing unit that is connected and works together.

4.2 System Work Flow

The workflow of our methodology encompasses three phases: Phase A: Data Collection, Phase B: Pre-Processing, and Phase C: Building a Model.

Step 1: Data Collection. The first step in the workflow involves collecting the dataset. We conduct an experimental analysis in this study using the CSIC 2010 HTTP dataset. This dataset is used in state-of-the-art existing works. This dataset contains real-life data developed in the Spanish Research National council in 2010 and is a publicly available dataset [55]. Specifically, it encompasses 1,000 HTTP (Hyper Text Transfer Protocol) requests. HTTP is the cornerstone for exchanging information over the World Wide Web. A very large quantity of data regularly flows over this protocol, and attacks towards this protocol are rapidly increasing. In this phase, HTTP traffic is analyzed and detected from within the CSIC 2010 HTTP dataset. Previously, classification of the CSIC 2010 HTTP dataset was used mainly with the purpose of instance and feature selection analysis. This dataset consists of 223,585 instances and 18 attributes. Moreover, it incorporates traffic maliciously targeting e-commerce web applications. The dataset is split into a collective set of normal and anomalous traffic requests. There are approximately 25, 000 anomalous requests and 36, 000 normal requests. We extracted the Cross-site Scripting (XSS) and SQL Injection (SQLI) attacks for our analysis from this dataset, since these two attacks fall under dynamic attacks that could harm users by stealing their confidential information. From the CSIC HTTP 2010 dataset, we eliminate unintentionally illegal requests, static attacks, and some dynamic attacks like buffer overflows and CRLF injection. As such, a total of 3096 instances are contained in our subset dataset, of which 1548 are anomalous (XSS+SQL) requests and 1548 are normal requests.

Despite its age, this dataset continues to be one of the most comprehensive and widely used in our field. There are many recent studies used this dataset for their experimental evaluations [56–62]. It contains a wide array of HTTP requests, both legitimate and malicious, making it ideal for testing and evaluating web application attack detection methods. Moreover, it enables the comparison of our results with those of past researches, contributing to a coherent body of knowledge. However, we understand the need for incorporating more recent datasets and will consider this in future studies to stay updated with the evolving web attack patterns.

Step 2: Pre-Processing. Following the collection of the dataset, the preprocesing step involves training the model. This step is completed using the normalization technique. Data normalization is a procedure by which all attributes are taken under the standard scale to pre-process the data. It is a preliminary processing which renders data compatible for analysis. Data processing is

widely recognized as an important part of anomaly detection [63]. Pre-processing includes all of the tasks performed prior to structured data processing, such as data cleaning, normalization, conversion, extraction, and selection of features [45]. Based on both resources and time, the pre-processing stage necessitates a significant amount of work, so it is necessary to pay attention to this step. Generally, about half the effort of an entire project is used to prepare the data [64]. This preparation step not only influences the efficacy of storing data, but also the effectiveness of the adoption process [63].

Step 3: Build a Model. The normalized data was divided into 80% for training and 20% for testing. The eight supervised machine learning classification algorithms used the training set to create models that could predict web application attacks. Each model's performance was assessed and compared using various evaluation metrics. The entire experiment was performed 50 times, with results averaged for accuracy. K-fold cross-validation also confirmed similar results. The aim was to identify the algorithm that most accurately classified normal and anomalous traffic requests. Results and detailed comparisons are provided in the experimental section.

5 Experimental Results

The experimental results and their evaluation are discussed in this section. Here we use seven performance metrics to determine the best algorithm. While evaluating the machine learning (ML) models, it is imperative to choose precise performance metrics [65]. Different performance criteria are proposed for evaluating ML models in diverse applications. The metrics that we chose to compare the algorithms are classification metrics such as accuracy, precision, recall (sensitivity), F_1-score, and Receiver Operating Characteristic (ROC) curve. Other popular metrics, such as time and Area Under the Curve (AUC), are also evaluated.

Accuracy: This is explained as the proportion of records correctly categorized over the total number of records. $Accuracy = (TP+TN)/(TP+TN+FP+FN)$. An accuracy value of 0.75 means that 75 of every 100 HTTP protocol requests is predicted correctly. Our system works to detect web application attacks based on the CSIC http 2010 dataset, which has a target class of 2 anomalous and normal web requests. According to our proposed definition of accuracy, it is the proportion of correctly identified anomalous and normal requests to the total number in our subset dataset.

Precision (P): This is defined as the proportion of true positive (TP) records divided by the total of true positive (TP) and false positive (FP) records classified. $P = TP/(TP + FP)$. Our intended definition of precision is that it is the ratio of precisely detected anomalous requests to the overall number of positive

predictions. A precision value of 0.77 means that 77 out of 100 HTTP protocol requests were precisely predicted.

Recall (R): This is known as the mean value of actual true positive records divided by the number of classified records of true positives and false negatives. It is also known as sensitivity, or the True Positive Rate. $TPR = Sensitivity = Recall = TP/(TP + FN)$. A recall value of 0.70 means that 30 of every 100 HTTP protocol requests to the web application are missed by the proposed detection system, and 70 are correctly predicted.

F_1 **score (F):** Also known as the F_1 Score and $F-$Measure, this score is defined as the harmonic mean of precision and recall and represents a balance between them. $F = 2 * P * R/(P + R)$.

Specificity: $TrueNegativeRate = TN/(TN + FP)$, where time is the amount of time intervals utilized by the metric system. A specificity value of 60% means that 40 out of every 100 HTTP protocol requests to the web application are mislabeled as anomalous requests and 60 are correctly labeled as normal requests. In our study, we did not use specificity as a performance metric.

ROC Curve: The Receiver Operating Characteristics Curve is a probability curve. It is basically illustrating, for different threshold values, the true positive rate (TPR) against the false positive rate (FPR) on the Y and X axis, respectively. $FPR = 1 - specificity$.

AUC: This is the Area Under the Curve. A higher AUC means that the model is predicting positives as positives and negatives as negatives. In our system, a higher AUC means the model can easily distinguish web requests as anomalous or normal. An AUC of 0.80 means the model can classify between anomalous and normal requests. On the contrary, an AUC of 0.50 means the model is not good at distinguishing web requests.

Time: This refers to computational time, which is the time taken by the system to predict the results.

The given input dataset is split into an 80:20 ratio, which means (80%) for training and (20%) for testing. Labels were assigned to anomalous (1) and normal (0) requests. We empirically evaluated the system by repeating the experiment 50 times on the subset dataset containing XSS and SQL injection anomalous HTTP requests, including balanced normal HTTP requests from the CSIC 2010 HTTP dataset. In the proposed system, the feature of input data is given to several classifier models to build the model and evaluate the performance of each classifier shown in Table 10. As mentioned earlier, we use eight different machine learning models. Table 10 illustrates the eight different supervised machine learning classifier algorithms that are used to classify the 2 target categorical values

of anomalous (bad request) and normal (good request) labels from the input data to predict the results.

Tables 2 through 9 show the results obtained from the experimental evaluations. Each individual table shows the result of different machine learning classifier models with different parameters used to predict web attacks based on web requests. As previously mentioned, we have two types of requests: bad and good, i.e., anomalous and normal, respectively. The classifiers are trained and tested accordingly so that we may obtain the corresponding values of the evaluation metrics. We can thus compare and analyze each model and reach a final conclusion for the classifier that outperforms the others in terms of accuracy, computational complexity, and so on.

Table 2. Naive Bayes Classifier Results. Prec: Precision; Acc: Accuracy

Model	Time(s)	Acc.	Prec	Recall	F-1	AUC
Naive Bayes	0.027	0.75	0.77	0.75	0.75	0.80

Table 2 provides the results obtained using the Naive Bayes classifier. This classifier yields a classification result with an accuracy of 0.75, a precision of 0.77, a recall of 0.75, an F_1 score of 0.75, and an AUC of 0.80. The time taken by the classifier for the prediction was 0.027. In terms of computational time, the Naive Bayes classifier outperforms the other classification models as it is much quicker, as depicted in Table 10.

Table 3. AdaBoost Classifier Results. Prec: Precision; Acc: Accuracy; Estim: Estimators

No. Estim	Time(s)	Acc.	Prec	Recall	F-1	AUC
50	1.98	0.75	0.77	0.75	0.75	0.80
60	2.55	0.75	0.77	0.75	0.75	0.80

The results obtained using the meta-learning method AdaBoost are shown in Table 3. The classification results of this model are evaluated by setting three parameters: the base estimator, the number of estimators, and the learning rate. In this experiment, we set the learning rate as 1 and the base estimator as tree or by default. This method will use the decision tree as the learner model. However, we try to find the different classification results by changing the number of estimators, which is the number of learners to be trained iteratively. We found that as the number of estimators change, there is no variation in accuracy, precision, recall, F1 score, or AUC, but there is a variation in time. As there is a slight difference found in the computational time, we are considering the best result obtained when the number of estimators = 50 at its initial stage, which

yields a value of 0.77 precision, and a value of 0.75 in accuracy, recall, and F1 score, a value of 0.80 for the area under the curve, and a computational time of $1.98s$, as depicted in Table 10.

Table 4 summarizes the results acquired using the random forest classifier, yet another ensemble learning method for both classification as well as regression. We evaluated the random forest classifier model with a different number of trees. We found that as the number of trees increases, there is no significant variation in performance metrics, but there are some variations in overall classification time. The best classification result was obtained with a number of trees equal to 5, which yielded a value of 0.77 precision, and a value of 0.75 for accuracy, recall, and F_1 score, a value of 0.80 for the AUC, and a time of 0.366 (see Table 10).

Table 4. Random Forest Classifier Results. Prec: Precision; Acc: Accuracy

No. Trees	Time(s)	Acc.	Prec	Recall	F-1	AUC
5	0.366	0.75	0.77	0.75	0.75	0.80
10	0.47	0.75	0.77	0.75	0.75	0.80
20	0.577	0.75	0.77	0.75	0.75	0.80

The decision tree classifier results are depicted in Table 5. The decision tree can also be used for both classification and regression problems; however, this method mainly focuses on classification. Here we used the decision tree classifier with different parameters. We used tree depth set at 100, a condition that does not split the subset smaller than 5, and, for the minimum number of instances in leaves, we used different values ranging from 2 to 15. Throughout the experiment, we understood that even if we change the number of instances in leaves, there is no dissimilarity in the classification results of the evaluation metrics and there are negligible changes in the computational time. The estimated results of the classifier model when the number of instances = 4 and time = $0.40s$ provides an accuracy of 0.73, a precision of 0.75, a recall and F_1 score of 0.73, and an AUC of 0.80, as shown in Table 10.

Table 5. Decision Tree Classifier Results. Prec: Precision; Acc: Accuracy; Inst: Instances

No. Inst	Time(s)	Acc.	Prec	Recall	F-1	AUC
2	0.45	0.73	0.75	0.73	0.73	0.80
4	0.40	0.73	0.75	0.73	0.73	0.80

In the logistic regression classifier, the results from Table 6 indicate that the regularization parameter 'C' does not have a significant impact on performance

metrics. Weak and strong values of 'C' show only slight variations in computational time. The accuracy, recall, and F_1 score have values of 0.75, while precision is 0.77, and the AUC is 0.80. The appropriate selection of the regularization parameter allows the classifier to provide a good fit without underfitting or overfitting, depending on the dataset.

Table 6. Logistic Regression Classifier Results. Prec: Precision; Acc: Accuracy

C-value	Time(s)	Acc.	Prec	Recall	F-1	AUC
1000	0.324	0.75	0.77	0.75	0.75	0.80
7	0.319	0.75	0.77	0.75	0.75	0.80
9	0.33	0.75	0.77	0.75	0.75	0.80
10	0.321	0.75	0.77	0.75	0.75	0.80
0.001	0.315	0.73	0.73	0.72	0.72	0.80

Table 7 represents the results obtained when using an Artificial Neural Network (ANN) with different hidden layers. In this study, we used an Artificial Neural Network (ANN) with various hidden layers and the Adam optimization algorithm. Different hidden layers didn't notably affect system classification performance, except for a rise in computational time with increasing layers. Performance metrics showed consistent results across different configurations, with accuracy, precision, recall, and F_1 score all equal to 0.75, and an AUC of 0.80. Hence, using different hidden layers in the ANN had little impact on system performance except for computation time.

The K-Nearest Neighbor (k-NN) classifier, a supervised machine learning algorithm for classification and regression, was used in the experiment. Different integer values of k - the number of nearest data points - were selected. Distance metrics (Euclidean, Manhattan, and Chebyshev) were used to calculate the distance between the testing and each row of training data. The performance evaluation metrics showed minor differences for the k-NN classifier with different k values and distance metrics, with slight impact on system performance. The best results were found with the Manhattan distance metric and showed insignificant impact across different k values. When $k = 7$ or 9, the accuracy, precision, and recall were 0.72 and 0.73 respectively, with an F_1 score of 0.72 and an AUC of 0.78. Some slight but significant changes were noted in computational time across all k values and distance metrics.

In our study, the Support Vector Machine (SVM), a supervised machine learning algorithm for classification and regression, was employed with varying kernels (Linear, Sigmoid, and Polynomial) and C values. For the Linear kernel, we observed a significant change in performance metrics with different C values, but found no such impact for the Sigmoid and Polynomial kernels. The time for model predictions varied slightly. The best results were obtained with a Linear kernel and $C = 0.01$, producing an accuracy of 0.58, precision of 0.60, recall of

Table 7. ANN Classifier Result. Prec: Precision; Acc: Accuracy; Hidd. L: Hidden Layers

Hidd. L	Time(s)	Acc.	Prec	Recall	F-1	AUC
50	0.502	0.75	0.77	0.75	0.75	0.80
80	0.542	0.75	0.77	0.75	0.75	0.80
100	0.563	0.75	0.76	0.75	0.75	0.80

0.58, F_1 score of 0.57, AUC of 0.368, and a time of $1.028s$. However, the SVM did not perform well for classifying anomalous and normal HTTP requests in our subset dataset.

Table 8. K-Nearest Neighbor Classifier Results. Prec: Precision; Acc: Accuracy; Hidd. L: Hidden Layers. K: K-value

K	Metric	Time	Acc	Prec	Recall	F-1	AUC
5	Euclidean	4.035	0.71	0.72	0.71	0.71	0.76
	Manhattan	4.143	0.71	0.72	0.71	0.71	0.76
	Chebyshev	3.217	0.71	0.71	0.71	0.71	0.77
7	Euclidean	3.7	0.71	0.72	0.71	0.71	0.77
	Manhattan	4.244	0.72	0.72	0.71	0.71	0.77
	Chebyshev	3.19	0.72	0.72	0.72	0.72	0.78
9	Euclidean	3.71	0.73	0.73	0.73	0.72	0.78
	Manhattan	4.326	0.73	0.73	0.73	0.72	0.78
	Chebyshev	3.542	0.73	0.73	0.73	0.72	0.78

Table 9. SVM Classifier Results. Prec: Precision; Acc: Accuracy. C: C-value

C	Kernel	Time	Acc	Prec	Recall	F-1	AUC
0.1	Linear	0.76	0.57	0.57	0.57	0.57	0.49
	Sigmod	1.028	0.58	0.60	0.58	0.57	0.368
	Polynomial	.836	0.58	0.61	0.57	0.54	0.487
0.5	Linear	0.829	0.55	0.56	0.55	0.52	0.47
	Sigmod	0.986	0.58	0.60	0.58	0.56	0.36
	Polynomial	0.840	0.56	0.61	0.57	0.53	0.52

Table 10 shows the overall best result obtained using different classifiers after comparing their different parameters. The table also includes the confusion matrix of each classifier, which demonstrates the number of requests out of the 3096 instances in the total subset dataset that were correctly predicted as anomalous or normal requests.

Table 10. Experimental results of proposed work.

Different Classifier Comparison								
Classifier	**Confusion Matrix**			**AUC**	**Classification Accuracy**	**Precision**	**Recall**	**F score**
	Class	anom (1)	norm (0)	0.80	0.75	0.77	0.75	0.75
Logistic Regression	anom (1)	9855	5645					
	norm (0)	1986	13514					
k-NN	anom (1)	10227	5273	0.78	0.73	0.73	0.73	0.72
	norm (0)	3237	12263					
AdaBoost	anom (1)	9855	5645	0.80	0.75	0.77	0.75	0.75
	norm (0)	2016	13484					
Random Forest	anom (1)	9887	5613	0.80	0.75	0.77	0.75	0.75
	norm (0)	2048	13452					
SVM	anom (1)	12318	3182	0.49	0.58	0.61	0.57	0.54
	norm (0)	9695	5805					
Neural Network	anom (1)	9855	5645	0.80	0.75	0.77	0.75	0.75
	norm (0)	2001	13499					
Naive Bayes	anom (1)	9855	5645	0.80	0.75	0.77	0.75	0.75
	norm (0)	2010	13490					
Decision Tree	anom (1)	10227	5275	0.80	0.73	0.75	0.73	0.73
	norm (0)	3235	12263					

6 Conclusion

Eight (8) supervised machine learning classification algorithms were tested on a subset of the CSIC 2010 HTTP dataset to determine their effectiveness in detecting web application attacks, focusing on XSS and SQLI dynamic attacks. Experimental results showed that Random Forest, Neural Networks, Naive Bayes, Logistic Regression and AdaBoost classifiers performed best based on metrics like accuracy, precision, recall, F1 Score, AUC, ROC, and time [42–44, 48]. SVM and KNN appeared to achieve lower accuracy and performance. Naive Bayes, Logistic Regression and Decision Tree algorithms were quickest, with SVM only faster when using a linear kernel. These results offer satisfactory solutions for detecting web application attacks.

In summary, our findings show KNN and Random Forest classifiers achieve 89% accuracy and 94% AUC on the CSIC dataset, while the Naive Bayes classifier excels in computational efficiency when distinguishing between malicious and benign HTTP requests.

Future work could involve unsupervised machine learning, deep learning algorithms, or new publicly available datasets.

Acknowledgment. This work was supported by grant number 12R170.

References

1. Mikheeva, O.I., Gatchin Yuri, A., Savkov, S.V., Khammatova, R.M., et al.: Search methods for abnormal activities of web applications. J. Sci. Tech. Inf. Technol. Mech. Optics **126**(2), 233–242 (2020)
2. Holz, T., Marechal, S., Raynal, F.: New threats and attacks on the world wide web. IEEE Secur. Priv. **4**(2), 72–75 (2006)
3. Moshchuk, A., Bragin, T., Deville, D., Gribble, S.D., Levy, H.M.: SpyProxy: Execution-based detection of malicious web content. In: USENIX Security Symposium, pp. 1–16 (2007)
4. Tekerek, A.: A novel architecture for web-based attack detection using convolutional neural network. Comput. Secur. **100**, 102096 (2021)
5. Huang, Y., Li, T., Zhang, L., Li, B., Liu, X.: JSContana: malicious javascript detection using adaptable context analysis and key feature extraction. Comput. Secur. **104**, 102218 (2021)
6. Phung, N.M., Mimura, M.: Detection of malicious javascript on an imbalanced dataset. Internet of Things **13**, 100357 (2021)
7. Nithya, V., Pandian, S.L., Malarvizhi, C.: A survey on detection and prevention of cross-site scripting attack. Int. J. Secur. Its Appl. **9**(3), 139–152 (2015)
8. Tariq, I., Sindhu, M.A., Abbasi, R.A., Khattak, A.S., Maqbool, O., Siddiqui, G.F.: Resolving cross-site scripting attacks through genetic algorithm and reinforcement learning. Expert Syst. Appl. **168**, 114386 (2021)
9. Jeitner, P., Shulman, H.: Injection attacks reloaded: tunnelling malicious payloads over DNS. In: 30th {USENIX} Security Symposium ({USENIX} Security 21), pp. 3165–3182 (2021)
10. Kc, G.S., Keromytis, A.D., Prevelakis, V.: Countering code-injection attacks with instruction-set randomization. In: Proceedings of the 10th ACM conference on Computer and communications security, pp. 272–280 (2003)
11. Hazel, P.: Perl compatible regular expressions, The University of Cambridge, p. 114 (2012)
12. Erlacher, F., Dressler, F.: On high-speed flow-based intrusion detection using snort-compatible signatures. IEEE Trans. Dependable Secur. Comput
13. Fredj, O.B., Cheikhrouhou, O., Krichen, M., Hamam, H., Derhab, A.: An OWASP top ten driven survey on web application protection methods. In: Garcia-Alfaro, J., Leneutre, J., Cuppens, N., Yaich, R. (eds.) CRiSIS 2020. LNCS, vol. 12528, pp. 235–252. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-68887-5_14
14. Perl-compatible regular expressions (PCRE), http://www.pcre.org (2021)
15. Kozik, R., Choraś, M., Renk, R., Hołubowicz, W.: A proposal of algorithm for web applications cyber attack detection. In: Saeed, K., Snášel, V. (eds.) CISIM 2014. LNCS, vol. 8838, pp. 680–687. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45237-0_61
16. Sharma, S., Zavarsky, P., Butakov, S.: 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), IEEE, pp. 227–230 (2020)
17. Oumaima, C., Abdeslam, R., Yassine, S., Abderrazek, F.: Experimental study on the effectiveness of machine learning methods in web intrusion detection. In: Maleh, Y., Alazab, M., Gherabi, N., Tawalbeh, L., Abd El-Latif, A.A. (eds.) ICI2C 2021. LNNS, vol. 357, pp. 486–494. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-91738-8_44

18. J. Offutt, Y. Wu, X. Du, H. Huang, Bypass testing of web applications. In: 15th International Symposium on Software Reliability Engineering, IEEE, pp. 187–197 (2004)
19. Sun, F., Zhang, P., White, J., Schmidt, D., Staples, J., Krause, L.: A feasibility study of autonomically detecting in-process cyber-attacks. In: 2017 3rd IEEE International Conference on Cybernetics (CYBCONF), IEEE, pp. 1–8 (2017)
20. Cova, M., Kruegel, C., Vigna, G.: Detection and analysis of drive-by-download attacks and malicious JavaScript code. In: Proceedings of the 19th international conference on World wide web, pp. 281–290 (2010)
21. Pazos, J.C., Légaré, J.-S., Beschastnikh, I.: XSnare: application-specific client-side cross-site scripting protection. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, pp. 154–165 (2021)
22. Johns, M., Engelmann, B., Posegga, J., Xssds: Server-side detection of cross-site scripting attacks. In: Annual Computer Security Applications Conference (ACSAC). IEEE, vol. 2008, pp. 335–344 (2008)
23. Fang, Y., Li, Y., Liu, L., Huang, C.: DeepXSS: cross site scripting detection based on deep learning. In: Proceedings of the 2018 International Conference on Computing and Artificial Intelligence, pp. 47–51 (2018)
24. Rodríguez, G.E., Torres, J.G., Flores, P., Benavides, D.E.: Cross-site scripting (XSS) attacks and mitigation: a survey. Comput. Netw. **166**, 106960 (2020)
25. Kaur, G., Malik, Y., Samuel, H., Jaafar, F.: Detecting blind cross-site scripting attacks using machine learning. In: Proceedings of the 2018 International Conference on Signal Processing and Machine Learning, pp. 22–25 (2018)
26. Kemalis, K., Tzouramanis, T.: SQL-IDS: a specification-based approach for SQL-injection detection. In: Proceedings of the 2008 ACM symposium on Applied computing, pp. 2153–2158 (2008)
27. Zhang, L., Zhang, D., Wang, C., Zhao, J., Zhang, Z.: ART4SQLI: the art of SQL injection vulnerability discovery. IEEE Trans. Reliab. **68**(4), 1470–1489 (2019)
28. Medeiros, I., Beatriz, M., Neves, N., Correia, M.: SEPTIC: detecting injection attacks and vulnerabilities inside the DBMS. IEEE Trans. Reliab. **68**(3), 1168–1188 (2019)
29. Fredj, O.B.: SPHERES: an efficient server-side web application protection system. Int. J. Inf. Comput. Secur. **11**(1), 33–60 (2019)
30. Zhuo, Z., Cai, T., Zhang, X., Lv, F.: Long short-term memory on abstract syntax tree for SQL injection detection. IET Softw. **15**(2), 188–197 (2021)
31. Li, Q., Li, W., Wang, J., Cheng, M.: A SQL injection detection method based on adaptive deep forest. IEEE Access **7**, 145385–145394 (2019)
32. Gu, H., et al.: DIAVA: a traffic-based framework for detection of SQL injection attacks and vulnerability analysis of leaked data. IEEE Trans. Reliab. **69**(1), 188–202 (2019)
33. Batista, L.O.: Fuzzy neural networks to create an expert system for detecting attacks by SQL injection, arXiv preprint arXiv:1901.02868
34. Fang, Y., Peng, J., Liu, L., Huang, C.: WOVSQLI: detection of SQL injection behaviors using word vector and LSTM. In: Proceedings of the 2nd International Conference on Cryptography, Security and Privacy, pp. 170–174 (2018)
35. Liu, M., Li, K., Chen, T.: DeepSQLi: deep semantic learning for testing SQL injection. In: Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 286–297 (2020)
36. D. Chen, Q. Yan, C. Wu, J. Zhao, Sql injection attack detection and prevention techniques using deep learning. J. Phys. Conf. Series **1757**, 012055 IOP Publishing (2021)

37. Nguyen, H.T., Torrano-Gimenez, C., Alvarez, G., Petrović, S., Franke, K.: Application of the generic feature selection measure in detection of web attacks. In: Herrero, Á., Corchado, E. (eds.) CISIS 2011. LNCS, vol. 6694, pp. 25–32. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21323-6_4
38. Yavanoglu, O., Aydos, M.: A review on cyber security datasets for machine learning algorithms. In: IEEE International Conference on Big Data (big data). IEEE, vol. 2017, pp. 2186–2193 (2017)
39. Kascheev, S., Olenchikova, T.: The detecting cross-site scripting (XSS) using machine learning methods. In: Global Smart Industry Conference (GloSIC). IEEE, vol. 2020, pp. 265–270 (2020)
40. Mereani, F.A., Howe, J.M.: Detecting cross-site scripting attacks using machine learning. In: Hassanien, A.E., Tolba, M.F., Elhoseny, M., Mostafa, M. (eds.) AMLTA 2018. AISC, vol. 723, pp. 200–210. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74690-6_20
41. Halfond, W.G., Viegas, J., Orso, A., et al.: A classification of SQL-injection attacks and countermeasures. In: Proceedings of the IEEE International Symposium on Secure Software Engineering, IEEE, vol. 1, pp. 13–15 (2006)
42. Saritas, M.M., Yasar, A.: Performance analysis of ANN and naive Bayes classification algorithm for data classification. Int. J. Intell. Syst. Appl. Eng. **7**(2), 88–91 (2019)
43. Garg, A., Roth, D.: Understanding probabilistic classifiers. In: De Raedt, L., Flach, P. (eds.) ECML 2001. LNCS (LNAI), vol. 2167, pp. 179–191. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44795-4_16
44. Kulkarni, C.C., Kulkarni, S.: Human agent knowledge transfer applied to web security. In: 2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT), IEEE, pp. 1–4 (2013)
45. Zhang, H.: The optimality of naive Bayes. Aa **1**(2), 3 (2004)
46. Myles, A.J., Feudale, R.N., Liu, Y., Woody, N.A., Brown, S.D.: An introduction to decision tree modeling. A J. Chemom. Soc. **18**(6), 275–285 (2004)
47. Liaw, A., Wiener, M., et al.: Classification and regression by randomforest. R News **2**(3), 18–22 (2002)
48. Howe, J., Mereani, F.: Detecting cross-site scripting attacks using machine learning. In: Advances in Intelligent Systems and Computing 723
49. Zhang, Z.: Introduction to machine learning: k-nearest neighbors. Anna. Transl. Med. **4**(11)
50. Bhor, R., Khanuja, H.: Analysis of web application security mechanism and attack detection using vulnerability injection technique. In: 2016 International Conference on Computing Communication Control and automation (ICCUBEA), IEEE, pp. 1–6 (2016)
51. Jakkula, V.: Tutorial on support vector machine (SVM), School of EECS, Washington State University 37
52. Rawat, R., Shrivastav, S.K.: SQL injection attack detection using SVM. Int. J. Comput. Appl. **42**(13), 1–4 (2012)
53. Braspenning, P.J., Thuijsman, F., Weijters, A.J.M.M. (eds.): Neural Network School 1999. LNCS, vol. 931. Springer, Heidelberg (1995). https://doi.org/10.1007/BFb0027019
54. Manzoor, I., Kumar, N., et al.: A feature reduced intrusion detection system using ANN classifier. Expert Syst. Appl. **88**, 249–257 (2017)
55. CSIC 2010 Dataset, https://petescully.co.uk/research/csic-2010-http-dataset-in-csv-format-for-weka-analysis/ (2021)

56. Bhatnagar, M., Rozinaj, G., Yadav, P.K.: Web intrusion classification system using machine learning approaches. In: International Symposium ELMAR. IEEE, vol. 2022, pp. 57–60 (2022)
57. Ramos Júnior, L.S., Macêdo, D., Oliveira, A.L.I., Zanchettin, C.: Detecting Malicious HTTP Requests Without Log Parser Using RequestBERT-BiLSTM. In: Xavier-Junior, J.C., Rios, R.A. (eds) Intelligent Systems. BRACIS 2022. LNCS(), vol 13654 . Springer, Cham (2022). https://doi.org/10.1007/978-3-031-21689-3_24
58. Ghazal, S.F., Mjlae, S.A.: Cybersecurity in deep learning techniques: Detecting network attacks. Int. J. Adv. Comput. Sci. Appl. **13**(11)
59. Li, W., Zhang, X.Y.: GBLNet: Detecting Intrusion Traffic with Multi-granularity BiLSTM. In: Groen, D., de Mulatier, C., Paszynski, M., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M.A. (eds) Computational Science – ICCS 2022. ICCS 2022. LNCS, vol 13353. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-08760-8_32
60. Tan, S., Sun, R., Liang, Z.: Detection of malicious web requests using neural networks with multi granularity features. In: Proceedings of the 5th International Conference on Big Data Technologies, pp. 83–89 (2022)
61. Shaheed, A., Kurdy, M.: Web application firewall using machine learning and features engineering, Secur. Commun. Netw. (2022)
62. Toprak, S., Yavuz, A.G.: Web application firewall based on anomaly detection using deep learning. Acta Infologica **6**(2), 219–244 (2022)
63. J. J. Davis, A. J. Clark, Data preprocessing for anomaly based network intrusion detection: a review. Comput. Secur. **30**(6–7), 353–375 (2011)
64. Kotsiantis, S.B., Kanellopoulos, D., Pintelas, P.E.: Data preprocessing for supervised leaning. Int. J. Comput. Sci. **1**(2), 111–117 (2006)
65. Performance metrics, https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics1ca3e282a2ce (2021)