InPrime: Indonesian Journal of Pure and Applied Mathematics
Vol. 7, No. 1 (2025), pp. 1 – 15, doi: 10.15408/inprime.v7i1.41025
p-ISSN 2686-5335, e-ISSN: 2716-2478

Machine Learning for Cybersecurity: Web Attack Detection (Brute Force, XSS, SQL Injection)

Muhammad Irfa'issurur[1] and Bony Parulian Josaphat[2]*
[1]Badan Pusat Statistik Kabupaten Pasaman, West Sumatra, Indonesia
[2]Department of Statistical Computing, Politeknik Statistika STIS, Jakarta, Indonesia
Email: *bonyp@stis.ac.id

Abstract

Security is a top priority in system development, as web portals serve as critical entry points that are frequently targeted by cyber-attacks. Common attack methods include SQL Injection, Cross-Site Scripting (XSS), and Brute Force. The application of machine learning in cybersecurity is growing due to its effectiveness in detecting such threats. This study employs supervised machine learning with six algorithms: K-Nearest Neighbors (KNN), Random Forest, Naïve Bayes, AdaBoost, LightGBM, and XGBoost. The research utilizes the CICIDS2017 and CSE-CICIDS2018 datasets, which contain network traffic data labeled with four categories: Benign, Brute Force, XSS, and SQL Injection. To address the dataset imbalance issue, this study applies Synthetic Minority Oversampling Technique (SMOTE) in conjunction with Principal Component Analysis (PCA) for dimensionality reduction. Performance evaluation is conducted using accuracy, precision, recall, and F1-score metrics, as well as K-Fold Cross Validation, AUC-ROC, and Learning Curve analysis. The results indicate that the Random Forest algorithm achieves the highest classification performance, with an accuracy of 97.77%, precision of 84.07%, recall of 91.96%, and an F1-score of 87.28%. This research contributes by demonstrating the applicability of machine learning in real-time web attack detection, highlighting the advantages of ensemble-based models in handling cybersecurity threats. Additionally, it underscores the importance of dataset preprocessing techniques in enhancing classification performance. Future improvements should focus on optimizing hyperparameters, integrating real-time network traffic analysis, and exploring hybrid models that combine traditional machine learning with deep learning approaches to further enhance detection capabilities.

Keywords: Machine learning; Cybersecurity; Web attack detection; Random forest; SMOTE; PCA.

Abstrak

Keamanan merupakan prioritas utama dalam pengembangan sistem, karena portal web berfungsi sebagai titik masuk penting yang sering menjadi sasaran serangan siber. Metode serangan umum meliputi SQL Injection*,* Cross-Site Scripting (XSS)*, dan* Brute Force. *Penerapan machine learning dalam keamanan siber semakin berkembang karena efektivitasnya dalam mendeteksi ancaman tersebut. Studi ini menggunakan supervised machine learning dengan enam algoritma:* K-Nearest Neighbors (KNN), Random Forest, Naïve Bayes, AdaBoost, LightGBM, *dan* XGBoost. *Penelitian ini memanfaatkan kumpulan data* CICIDS2017 *dan* CSE-CICIDS2018*, yang berisi data lalu lintas jaringan yang diberi label dengan empat kategori:* Benign, Brute Force, XSS, *dan* SQL Injection. *Untuk mengatasi masalah ketidakseimbangan kumpulan data, studi ini menerapkan* Synthetic Minority Oversampling Technique (SMOTE) *bersama dengan* Principal Component Analysis (PCA) *untuk pengurangan dimensionalitas. Evaluasi kinerja dilakukan dengan menggunakan metrik akurasi, presisi, recall, dan skor F1, serta* K-Fold Cross Validation*,* AUC-ROC*, dan analisis* Learning Curve. *Hasilnya menunjukkan bahwa algoritma* Random Forest *mencapai kinerja klasifikasi tertinggi, dengan akurasi 97,77%, presisi 84,07%, recall 91,96%, dan skor F1 87,28%. Penelitian ini berkontribusi dengan menunjukkan penerapan machine learning dalam deteksi serangan*

*) Corresponding author

Submitted August 22nd, 2024, Revised February 19th, 2025,
Accepted for publication February 24th, 2025, Published Online February 27th, 2025

web real-time, menyoroti keunggulan model berbasis ensemble dalam menangani ancaman keamanan siber. Selain itu, penelitian ini menggarisbawahi pentingnya teknik praproses dataset dalam meningkatkan kinerja klasifikasi. Peningkatan di masa mendatang harus difokuskan pada pengoptimalan hiperparameter, pengintegrasian analisis lalu lintas jaringan real-time, dan eksplorasi model hybrid yang menggabungkan machine learning tradisional dengan pendekatan deep learning untuk lebih meningkatkan kemampuan deteksi.

Kata Kunci: *Pembelajaran mesin; Keamanan siber; Deteksi serangan web; Random forest;* SMOTE*;* PCA.

2020MSC: 68T05

1. INTRODUCTION

As the internet continues evolving, the web has become the primary portal for users to access public and private information through browsers. The increasing use of web application systems has led to potential security vulnerabilities that can be exploited by attacks such as SQL Injection, XSS, and Brute Force [1,2,3]. System security is a top priority in protecting data, necessitating methods capable of detecting various cyber-attacks [4].

Machine learning is widely used in data analysis because it can create models that can predict values by learning patterns from data. As a branch of artificial intelligence, machine learning algorithms can understand the normal behavior patterns of users and systems and detect suspicious behavior or activities [5]. This enables security systems to quickly detect attacks, even before they reach a damaging stage. Machine learning can assist humans in performing large-scale attack detection analyses rapidly, which would be impossible to do manually. Implementing machine learning in web security provides real-time automation in monitoring network behavior [6]. This allows continuous, uninterrupted monitoring to protect systems from any attack.

As cyber threats become more sophisticated, the application of machine learning is a promising step in detecting ever-evolving and increasingly complex cyberattacks [7]. This study classifies web attack data (Brute Force, XSS, SQL Injection) using six selected algorithms: KNN, Naïve Bayes, Random Forest, AdaBoost, LightGBM, and XGBoost.

Much research has been conducted in detecting cyber-attacks because maintaining data security is crucial to avoiding cyber-attacks. Previous studies, such as the work by Sharafaldin et al. (2018), have generated reliable intrusion detection datasets like CICIDS2017, which were designed to address the limitations of older datasets such as DARPA98 and KDD99 [8]. CICIDS2017 covers many modern attacks, including SQL Injection, Brute Force, and XSS, offering a realistic representation of network traffic and attack patterns. While prior works have utilized machine learning for web attack detection, most have focused on a limited set of models, often neglecting the impact of data imbalance on performance.

This study differentiates itself by employing six machine learning algorithms—K-Nearest Neighbors (KNN), Naïve Bayes, Random Forest, AdaBoost, LightGBM, and XGBoost—on the CICIDS2017 and CSE-CICIDS2018 datasets. To improve classification accuracy, we incorporate Synthetic Minority Oversampling Technique (SMOTE) to address dataset imbalance and Principal Component Analysis (PCA) for dimensionality reduction.

This research's key contribution lies in its more comprehensive application of machine learning techniques, with special attention given to the data imbalance problem that can affect model performance. This study enhances real-time web attack detection using techniques like SMOTE [9] and PCA [10], offering a potential foundation for developing more robust cybersecurity systems.

The main objective of this study is to develop a machine learning model that can effectively classify network traffic into benign and web attack classes (Brute Force, XSS, SQL Injection). The study aims to identify which machine learning algorithm is the most effective in achieving this classification. Using the CICIDS2017 and CSE-CICIDS2018 datasets, containing four labeled classes—Benign, Brute Force, XSS, and SQL Injection—provides a strong foundation for training and evaluating the models. By achieving these objectives, this study contributes to developing robust machine learning models for web attack detection and offers insights into how machine learning can be better applied to address the evolving challenges in cybersecurity.

2. METHODS

2.1. Machine Learning Algorithms

a. K-Nearest Neighbor

K-Nearest Neighbor (KNN) is a non-parametric algorithm used for regression and classification. KNN is also a lazy learning algorithm, meaning it does not use training data points to create a model[11]. The KNN algorithm classifies data based on the characteristics of its neighbors by calculating the proximity distance between the k closest data points using Euclidean, Manhattan, Hamming, or Minkowski distances. The workflow of the KNN algorithm can be summarized as follows [12].

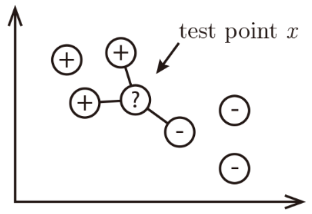

Figure 1. K-Nearest Neighbor Classifiation [12].

Given point x is the target to be classified. Define the set of the k nearest neighbors of x as S_x. Formally S_x is defined as $S_x \subseteq D$ s.t. $|S_x| = k$ and $\forall(x', y') \in D/S_x$ [13],

$$dist(x, x') \geq \max_{(x", y") \in S_x} dist(x, x"). \quad (1)$$

We can then define the classifier $h()$ as a function returning the most common label in S_x:

$$h(x) = mode(\{y : (x", y") \in S_x\}). \quad (2)$$

The k-nearest neighbor classifier fundamentally relies on a distance metric. The better that metric reflects label similarity, the better the classification. The most common choice is the Minkowski distance.

$$dist(x, z) = \left(\sum_{r=1}^{d} |x_r - z_r|^p\right)^{\frac{1}{p}}. \quad (3)$$

b. Random Forest

The Random Forest algorithm combines multiple decision trees using the bagging method [14]. This involves creating various subsets of the training data randomly, which are then used to train the machine learning model. The classification process in Random Forest is done by voting on each decision tree and then combining the results to select the class with the most votes. Random Forest is an ensemble learning method that constructs multiple decision trees and aggregates their outputs through majority voting [14]. We use the following hyperparameters: n_estimators = 100, max_depth = None, min_samples_split = 2, bootstrap = True, and criterion = 'gini'.

c. Naïve Bayes

Naive Bayes is a very popular classification method in machine learning. Naive Bayes applies Bayes' theorem and the strong assumption that each feature is conditionally independent given the target class. Although this assumption is often violated in practice, the method can still compete with others regarding accuracy [15]. Naive Bayes is also referred to as a conditional probability model, which can be described as follows [16].

$$P(C_k|x) = \frac{P(C_k)P(x|C_k)}{P(x)}, \tag{4}$$

where $P(C_k|x)$ is posterior probability i.e. the probability of class C_k given the observed data x, $P(C_k)$ is prior probability i.e. the probability of class C_k before observing any data. It represents our initial belief about the class distribution, $P(x|C_k)$ is likelihood i.e. the probability of observing the data xxx given that it belongs to class C_k, $P(x)$ is evidence (marginal likelihood) i.e. the overall probability of observing x, considering all possible classes. This equation can be written as

$$posterior = \frac{prior \times likelihood}{evidence}. \tag{5}$$

The class decision is determined by the highest posterior probability as follows:

$$\hat{C} = \underset{k \epsilon \{1,\dots,K\}}{\operatorname{argmax}} P(C_k) \prod_{i=1}^{n} P(x_i|C_k). \tag{6}$$

Naïve Bayes assumes independence among features and applies Bayes' theorem for classification [15]. The Gaussian Naïve Bayes model is used, as our dataset contains continuous numerical features. The key parameter in this model is var_smoothing = 1e-9, which prevents zero probability issues by adding a small value to variance calculations.

d. AdaBoost

Adaptive Boosting, or AdaBoost, is an ensemble algorithm that applies the boosting technique in machine learning by reweighting each misclassified data point. AdaBoost works by repeatedly training multiple weak learners, such as decision trees, over a specified number of iterations until the model becomes stronger [17].

According to [18], first assign the initial weights to the data by giving each data point a weight $w_i^t = D(i) = 1/N$, where i indicates the position of the ith data point, and t indicates the iteration. The weight distribution is given by

$$p^t = \frac{w^t}{\sum_{i=1}^{N} w_i^t}. \tag{7}$$

Apply the weak learner with the associated weighting p^t and then calculate the error of the weak learner with

$$h_t : \varepsilon_t = \sum_{i=1}^{N} p_i^t |h_i(x_i) - y_i|, \tag{8}$$

where $|h_i(x_i) - y_i|$ is a function that takes a value of 0 or 1. Calculate α_t, which represents the strength of the weak learner. It is given by $\beta_t = \varepsilon_t/(1-\varepsilon_t)$ and $\alpha_t = \frac{1}{2}\log\left(\frac{1}{\beta_t}\right)$. Next is to update the weights. Misclassified data points receive a larger weight $w_i^{t+1} = w_i^t \times e^{\alpha_t}$ while correctly classified data points receive a smaller weight $w_i^{t+1^*} = w_i^t \times e^{-\alpha_t}$. Output hypothesis given by

$$h_f(x) = \begin{cases} 1, & if\ \sum_{t=1}^{T} (\log\frac{1}{\beta_t})\, h_t(x) \geq \frac{1}{2}\sum_{t=1}^{T} \log\frac{1}{\beta_t}\ , \\ 0, & \text{elsewhere.} \end{cases} \tag{9}$$

The base estimator for this study is Decision Tree (max_depth=1) to ensure weak learners, as AdaBoost relies on sequential improvements. Other parameters include n_estimators = 20 and learning_rate = 1.0.

e. LightGBM

LightGBM is a machine learning algorithm Microsoft developed based on decision trees using gradient boosting. It implements the Gradient Boosting Decision Tree (GBDT), popular in the machine learning community. However, GBDT has faced various challenges in handling high-dimensional and large-scale data because, for each feature, GBDT needs to scan all the data to estimate the information gained from all potential split points. To address these issues, Ke, Guolin, et al. [19]. proposed two techniques: Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB). Key hyperparameters include num_leaves = 31, max_depth = -1, learning_rate = 0.01, boosting_type = 'gbdt', and feature_fraction = 1.

f. XGBoost

XGBoost is an implementation of Gradient Boosting Decision Tree (GBDT) that is highly popular in the machine learning community [20]. The algorithm begins by initializing the prediction using log-odds or logit [21]:

$$\hat{y}_i^{(0)} = \frac{1}{n}\sum_{i=1}^{n} y_i = p, \tag{10}$$

$$\text{logit}(p) = \log\left(\frac{1}{1-p}\right), \tag{11}$$

where $\hat{y}_i^{(0)}$ is the initial predicted value for the i-th instance before boosting starts, n is the total number of training samples, y_i is the actual label (ground truth) of the iii-th training sample, p is the average of the labels, which represents the initial probability estimate (prior probability) of the positive class, and $\text{logit}(p)$ is the log-odds transformation of the probability p.

Then, it proceeds with the Gradient Boosting iterations, where the gradient and hessian are calculated using the previous iteration's predicted probabilities

- Gradient

$$g_i^{(t)} = \frac{\partial L\left(y_i, \hat{y}_i^{(t-1)}\right)}{\partial \hat{y}_i^{(t-1)}} = \hat{p}_i^{(t-1)} - y_i. \tag{12}$$

- Hessian

$$h_i^{(t)} = \frac{\partial^2 L(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)2}} = \hat{p}_i^{(t-1)}\left(1 - \hat{p}_i^{(t-1)}\right), \tag{13}$$

where $\hat{p}_i^{(t-1)}$ represents the predicted probability at iteration $(t-1)$th and

$$\hat{p}_i^{(t-1)} = \frac{1}{1+\exp\left(-\hat{y}_i^{(t-1)}\right)}. \tag{14}$$

Following this, a new model (typically a decision tree) fit. The predictions are then updated to compute the branch values in the decision tree

$$w_j = -\frac{\sum_{i\epsilon I_j} g_i}{\sum_{i\epsilon I_j} h_i+\lambda}, \tag{15}$$

and the predicted values $\hat{y}_i^t$ are further adjusted with the learning rate η.

$$\hat{y}_i^t = \hat{y}_i^{(t-1)} + \eta \cdot w_j. \tag{16}$$

These steps are repeated for a set number of iterations, during which the gradients and Hessians are recalculated, a new tree is fit, and the predictions are updated accordingly. The parameters used are n_estimators = 100, max_depth = 4, learning_rate = 0.1, subsample = 1, colsample_bytree = 1, and gamma = 0.

2.2. Nunamaker and Chen

The system is developed using the Nunamaker and Chen system development method [22], as shown in Figure 2. According to Figure 2, this method consists of five iterative stages: creating a conceptual framework, designing the system architecture, system analysis and design, building a prototype, and system evaluation.

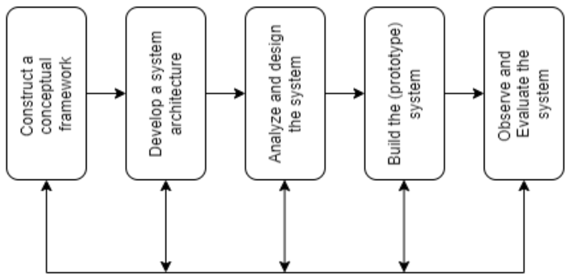
Construct a conceptual framework
Develop a system architecture
Analyze and design the system
Build the (prototype) system
Observe and Evaluate the system

Figure 2. Nunamaker and Chen system development research process [23].

a. Construct a conceptual framework

At this stage, the foundational structure of the research is outlined. It defines the core components of the study, such as the dataset, machine learning models, and evaluation methods. This framework provides an overarching view of the research, explaining the key elements involved and how they interact with the research process. In essence, it is a theoretical design that outlines how the study will proceed conceptually, as shown in Figure 3.

b. Develop a system architecture

The System Architecture goes further by detailing how the framework is implemented practically. At this stage, an architectural diagram shows the specific processing steps involved. It explains how the system is structured technically, outlining the stages of data collection, preprocessing, model training, and evaluation. Unlike the Conceptual Framework, which focuses on the big-picture design, the System Architecture focuses more on the system's technical structure and practical implementation, as shown in Figure 4.

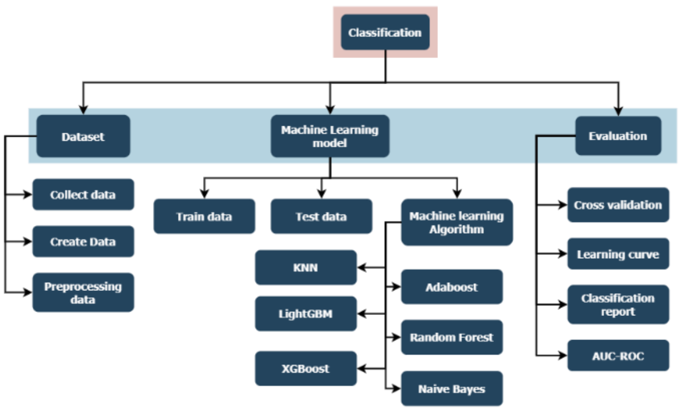

Figure 3. Conceptual framework.

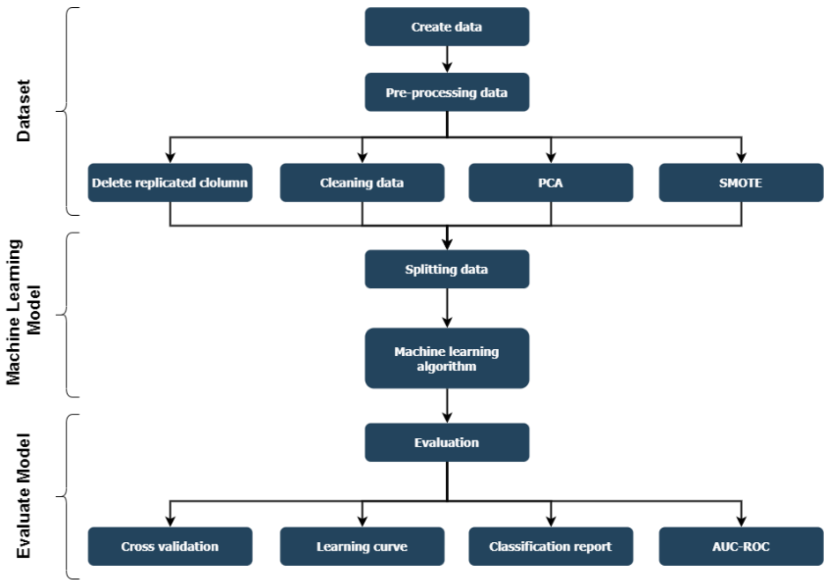

Figure 4. System architecture.

c. Analyze and design the system

This section outlines the preparation steps before implementing the system, including data collection, analysis, preprocessing, splitting, model training, and model evaluation, as shown in Figure 5.

d. Build the system

The system design that has been created is then implemented at this stage.

e. Observe and evaluate the system.

After implementation, the model is evaluated using cross-validation, an AUC-ROC curve, a learning curve, and a classification report. The following outlines the process of building the machine learning model based on the predefined system design.

The datasets used in this study are sourced from the University of New Brunswick, Canadian Institute for Cybersecurity, namely CICIDS2017 [8] and CSE-CICIDS2018 [24]. The dataset used in this research combines CICIDS2017 and CSE-CICIDS2018 (Attack labeled). This dataset consists of

77 attributes/variables, one attack label attribute, and 171,159 rows of data. The label attribute has four types: Benign, Brute Force, XSS, and SQL Injection.

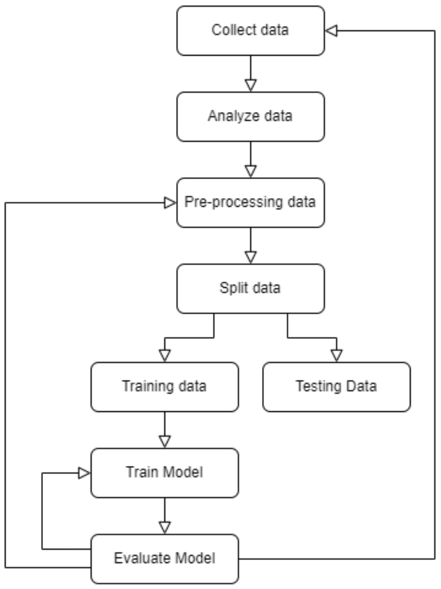

Figure 5. Flowchart of developing a machine learning model.

Research Stages:

1. Preprocessing data
   The obtained dataset is then subjected to data preprocessing, where it is prepared before modeling according to the analysis requirements. The processes involved include adjusting variable names, removing duplicate attributes, encoding data, checking missing values, applying PCA (Principal Component Analysis) to reduce dimensions, and employing SMOTE (Synthetic Minority Oversampling Technique) to address data imbalance. After completion, the dataset will be divided into class X (77 attributes/variables) and class Y (attack labels).
2. Splitting data
   At this stage, the dataset will be divided into training and testing, with a ratio of 90% for training data and 10% for testing data.
3. Classifying data
   At this stage, classification will be conducted using the predetermined machine learning algorithms: KNN, Naïve Bayes, Random Forest, AdaBoost, LightGBM, and XGBoost.
4. Evaluating
   Evaluating machine learning models is crucial for determining their performance and effectiveness in detecting cyber-attacks. The primary goal of model evaluation is to measure how well a model performs in classifying or predicting data based on specific metrics. The following metrics are used to assess the models applied in this research comprehensively:
   a) Classification Report
      The classification report is a document that presents evaluation metrics of the classification model's performance against target classes. It typically includes the confusion matrix, precision, recall, F1-score, and support. These evaluation metrics allow us to understand a

model's accuracy and reliability in real-world scenarios. In cybersecurity, where detecting attacks is critical, measuring precision, recall, and other metrics ensures that the model is theoretically sound and practically useful.

b) K-Fold Cross Validation
K-Fold Cross Validation is a technique that divides the dataset into k random parts, using k-1 parts for training and one part for testing, then repeating this process k times. The accuracy results from k repetitions are averaged to produce a single estimation. Through techniques like cross-validation, we can detect whether a model is overfitting (too closely tuned to the training data) or underfitting (failing to capture important patterns in the data). Overfitting leads to poor generalization, which is dangerous in cybersecurity when models encounter new, unseen threats.

c) AUC-ROC
AUC-ROC (Area Under the Curve—Receiver Operating Characteristics) is a measurement technique used to determine how well a model can distinguish classes at various threshold settings. The higher the AUC level, the better the model predicts different classes. We plot the ROC curve and calculate TPR (True Positive Rate) and FPR (False Positive Rate).

d) Learning Curve
The learning curve is a graph that compares a model's performance for training and testing with different numbers of training objects, thereby indicating whether the model can learn from the data as the training data increases.

Utilizing appropriate evaluation techniques—such as classification reports, cross-validation, AUC-ROC, and learning curves—ensures that machine learning models are effective, reliable, and scalable for real-world cybersecurity applications. These methods provide valuable insights into the model’s strengths and limitations, facilitating the selection and refinement of the best-performing models.

5. RESULTS

Figure 6 illustrates the distribution of web attack label types to assess data balance. The figure shows the imbalance in data points across different types of web attacks. Further processing is performed to achieve data balance.

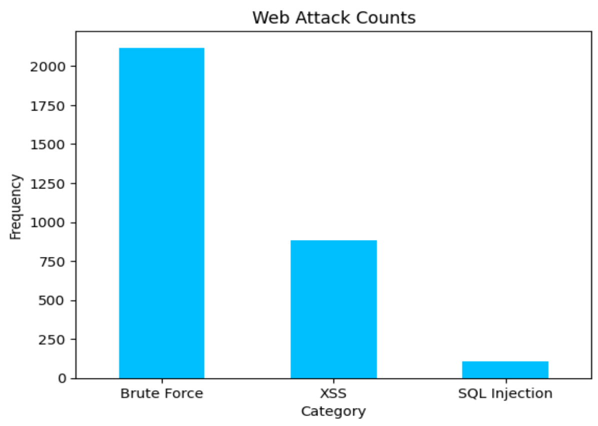

Figure 6. Web attack labeled data.

1. Preprocessing data

The process consists of three steps: (1) removing null values, NaN values, and duplicate columns, (2) applying PCA for dimensionality reduction, as shown in Figure 7, where seven principal components are identified to achieve 99% cumulative variance, and (3) using SMOTE to balance the dataset, generating 20,000 instances per attack type, excluding the original data.

2. Splitting data

After data preprocessing, the dataset is split into two parts: 90% for training, consisting of benign data and SMOTE-resampled web attack data, and 10% for testing, which includes both benign and web attack data (see Figure 8).

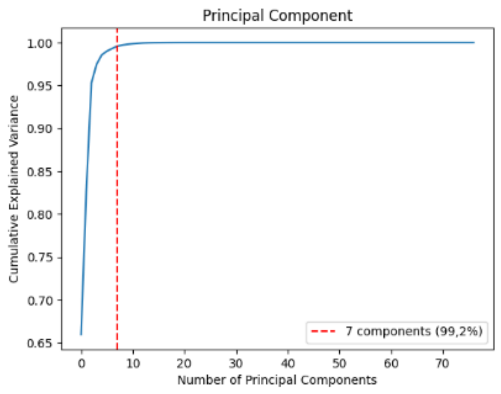

Figure 7. The PCA's Result.

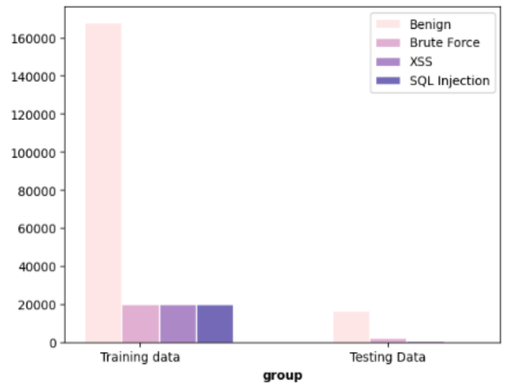

Figure 8. Splitting data: training and testing.

3. Training model

At this stage, six machine learning algorithms are applied to the training data to develop a web attack classification model.

4. Evaluating

In the evaluation stage, the model's performance and accuracy are assessed using predefined methods, with the results displayed for comparison. Table 1 presents the classification report.

Table 1. Classification report

No	Algorithm	Accuracy	Precision	Recall	F1-score
1.	KNN	96.96%	80.72%	89.29%	84.17%
2.	Random Forest	97.77%	84.07%	91.96%	87.28%
3.	Naïve Bayes	17.40%	28.76%	33.32%	12.22%
4.	Adaboost	88.99%	57.83%	65.29%	53.70%
5.	LightGBM	94.68%	69.72%	84.01%	64.41%
6.	XGBoost	94.79%	70.37%	85.15%	73.56%

Based on Table 1, models such as Random Forest and KNN achieved high precision, recall, and F1 scores above 80%, demonstrating their ability to balance false positives and false negatives, making them well-suited for real-world attack detection. In contrast, Naïve Bayes showed poor performance, with low precision (28.76%) and an F1-score of 12.22%, indicating a high false positive rate and reduced reliability for attack detection. Meanwhile, LightGBM and XGBoost delivered consistently strong performance with high precision and recall, highlighting the effectiveness of boosting algorithms in handling imbalanced data. This suggests boosting models like LightGBM and XGBoost are more resilient to data imbalance than probabilistic methods such as Naïve Bayes. Other boosting algorithms, such as AdaBoost, exhibited lower performance than LightGBM and XGBoost, underscoring the importance of parameter tuning to optimize boosting model performance.

Table 2 presents the results of K-Fold Cross Validation using $k = 10$. Based on the table, Random Forest and KNN models demonstrate consistent performance across almost every fold, with minimal accuracy variation (96–97%). This stability indicates that these models effectively learn from different data subsets and are not overly sensitive to data splits. In contrast, Naïve Bayes consistently yielded low accuracy (~17%) across all folds, suggesting underfitting. This implies that the model lacks the complexity to capture underlying data patterns, potentially due to its feature independence assumption, which may not align with real-world data. Boosting algorithms like XGBoost and LightGBM performed well, with average accuracies exceeding 94% and 88%, respectively. However, both exhibited slight fluctuations across folds, highlighting the need for hyperparameter tuning to ensure optimal and stable performance. Meanwhile, AdaBoost showed greater accuracy variability (84–89%) than other algorithms, indicating potential sensitivity to outliers or noise. Significant fluctuations in K-Fold results may suggest that the model lacks generalization or is highly sensitive to data variations.

Table 2. K-Fold Cross Validation

No	KNN	Random Forest	Naïve Bayes	AdaBoost	LightGBM	XGBoost
1.	97.07%	97.70%	17.60%	84.59%	89.01%	94.77%
2.	97.08%	97.78%	17.36%	88.91%	89.03%	94.89%
3.	96.96%	97.75%	17.40%	84.74%	88.89%	94.68%
4.	97.07%	97.84%	16.74%	84.84%	89.04%	94.88%
5.	97.01%	97.81%	16.75%	88.92%	89.03%	94.92%
6.	97.12%	97.78%	16.88%	84.64%	88.83%	94.71%
7.	96.95%	97.67%	16.98%	88.47%	88.86%	94.74%
8.	96.98%	97.77%	17.25%	88.79%	88.91%	94.85%
9.	96.95%	97.66%	17.62%	88.78%	88.78%	94.62%
10.	96.96%	97.77%	17.40%	88.99%	88.99%	94.79%
Average	97.02%	97.75%	17.20%	87.17%	88.94%	94.78%

Figure 9 presents the AUC-ROC results. As shown in Figure 9, models using ensemble algorithms such as Random Forest, XGBoost, and LightGBM achieve AUC values close to 1, indicating excellent discriminatory capability. This suggests that these models effectively differentiate between malicious and normal traffic. In contrast, Naïve Bayes may exhibit an AUC close to 0.5, reflecting poor performance. This suggests that Naïve Bayes struggles with datasets containing feature dependencies, reinforcing the importance of selecting a model suited to the data characteristics.

Additionally, models using the AdaBoost algorithm may show lower or more variable AUC values across experiments. Since boosting algorithms can be sensitive to outliers, extensive tuning is often required. This sensitivity may lead to higher false positive rates at certain thresholds, particularly in noisy datasets.

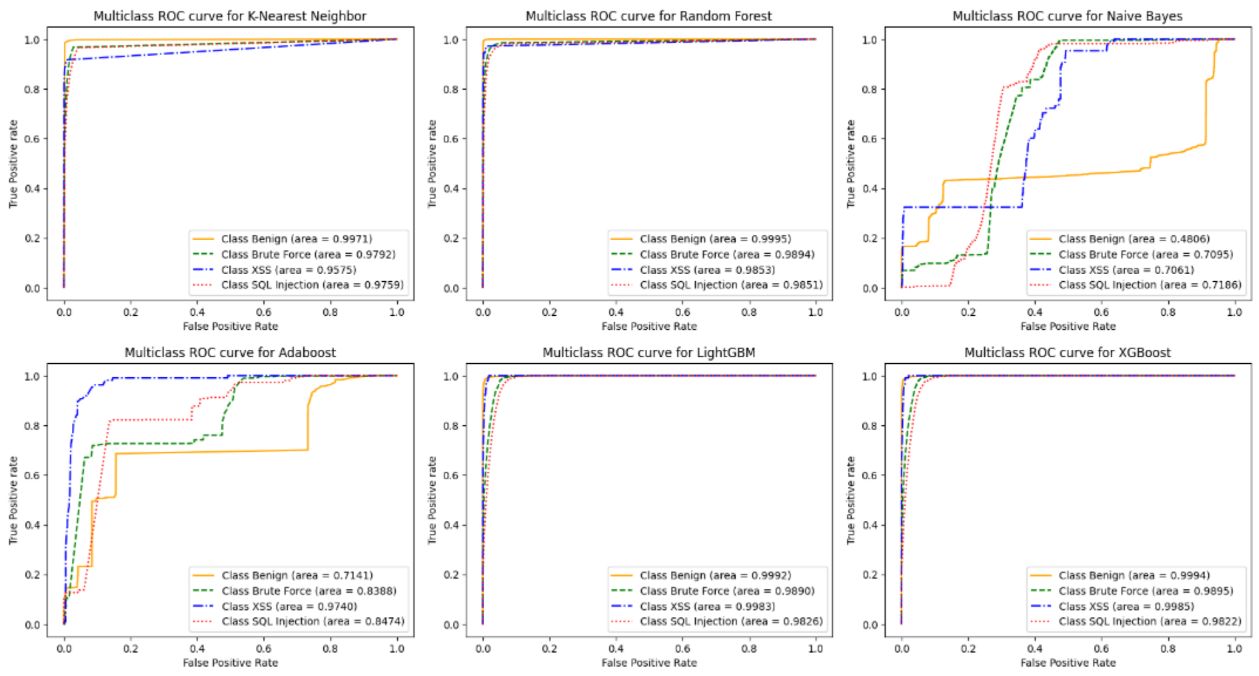

Figure 9. The AUC-ROC result.

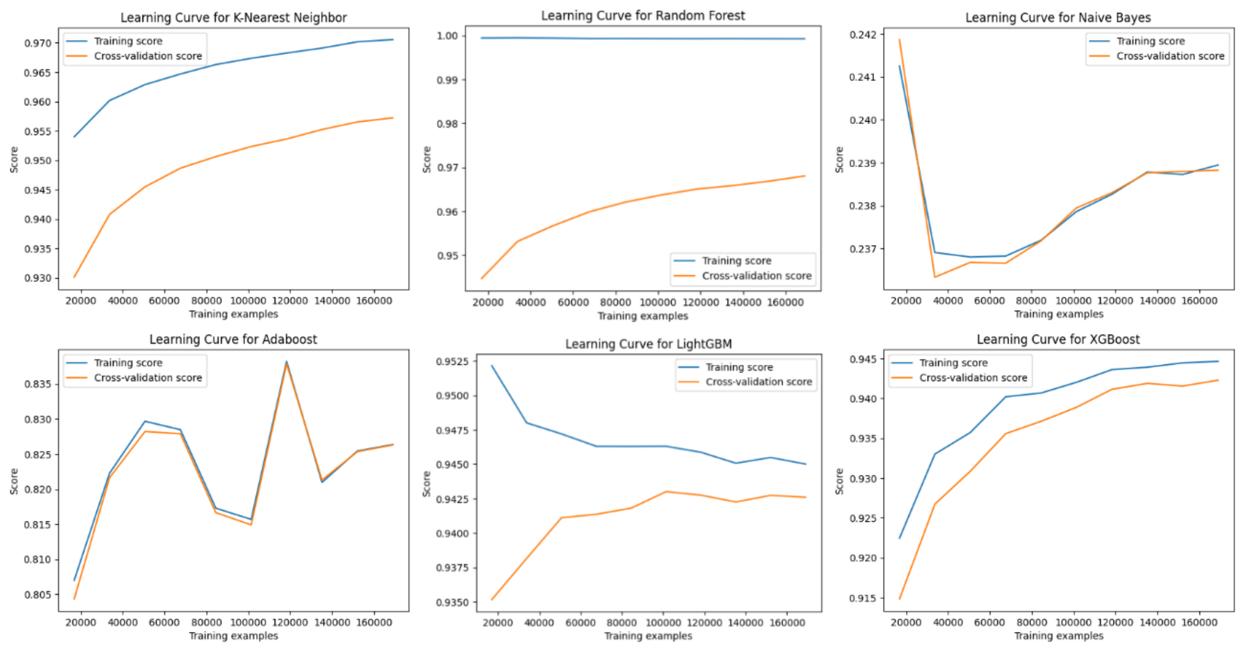

Figure 10. Learning Curves.

Models with a learning curve that consistently improves as more data is added are less likely to experience underfitting, and their performance can benefit from additional data. Figure 10 presents the Learning Curve results. Based on the figure, KNN, Random Forest, XGBoost, and LightGBM exhibit strong learning curves, with the testing performance steadily improving as training data increases. This indicates that ensemble methods possess good generalization capabilities, even with larger datasets. Underfitting is evident when the training and testing curves remain low, as seen in Naïve Bayes. This suggests that the model is too simplistic to capture complex data patterns. Conversely, AdaBoost may be prone to overfitting, especially if not properly tuned or if the dataset contains noise.

5. DISCUSSION

This study evaluated the application of machine learning algorithms for detecting web attacks, such as Brute Force, XSS, and SQL Injection, demonstrating these techniques' potential in cybersecurity. The evaluation result shows the effectiveness of these methods, particularly the Random Forest algorithm. The main strength of this study is the use of relevant datasets, CICIDS2017 and CSE-CICIDS2018, which provide a realistic representation of network traffic and attack patterns [8].

The imbalanced nature of the original datasets necessitates the use of balancing techniques such as SMOTE oversampling, which may introduce synthetic bias [9]. The large number of features can be a curse of dimensionality, which makes it challenging to perform analysis, so it is necessary to transform or simplify the data using certain methods such as PCA [10]. Our results are consistent with related research by Sharafaldin et al. [8], which demonstrated the superiority of ensemble methods in intrusion detection tasks. However, our implementation using LightGBM and XGBoost also showed significant performance [25] , suggesting that boosting algorithms can be an alternative to traditional techniques such as Random Forest. This indicates a potential avenue for enhancing detection abilities.

The evaluation of the models, using techniques like K-Fold Cross Validation, showed that both Random Forest and KNN were stable, with only minor fluctuations in accuracy across different data subsets. On the other hand, boosting models like XGBoost and LightGBM performed well overall but exhibited slight variations, suggesting that more fine-tuning of their hyperparameters could improve their reliability.

One limitation of this study is the use of static datasets, which may not fully represent emerging attack patterns in dynamic cybersecurity environments. So, future research could explore integrating real-time network traffic data and advanced feature engineering techniques to improve model adaptability. Moreover, combining deep learning with traditional machine learning models in hybrid approaches could further boost detection accuracy.

Overall, this study illuminates how machine learning can enhance web attack detection, providing valuable insights into the balance between model complexity, interpretability, and performance in real-world cybersecurity scenarios [25].

6. CONCLUSIONS

This study used six different algorithms to explore the application of machine learning algorithms for detecting web attacks, such as Brute Force, XSS, and SQL Injection. The results demonstrate that models like Random Forest and KNN performed best, offering high accuracy and robustness in real-time web attack detection. However, some limitations must be acknowledged. The datasets utilized—CICIDS2017 and CSE-CICIDS2018—present an imbalance between benign and attack instances. Although SMOTE was applied for oversampling, creating synthetic data may introduce bias and not fully represent real-world scenarios, potentially reducing model effectiveness when deployed in naturally imbalanced environments. Additionally, while this study concentrated on specific algorithms, the model's performance is influenced by parameter settings that were not thoroughly optimized. More in-depth hyperparameter tuning could further enhance model precision and reliability. Moreover, the features used in this study were based on predefined network attributes, which may restrict the model's adaptability to evolving threats. Future research could focus on dynamic feature extraction and real-time data collection for continuous model improvement. By addressing these limitations, future work can strengthen the applicability and reliability of machine learning models in cybersecurity, especially in managing diverse and evolving cyber threats.

ACKNOWLEDGMENT

This research was not funded by any grant.

REFERENCES

[1] Z. Liu, Y. Fang, C. Huang, and Y. Xu, "MFXSS: An effective XSS vulnerability detection method in JavaScript based on multi-feature model," *Comput. Secur.*, vol. 124, p. 103015, 2023, doi: https://doi.org/10.1016/j.cose.2022.103015.

[2] A. Buja, "An Online SQL Vulnerablility Assessment Tool and It's Impact on SMEs," *Int. J. Adv. Res. Comput. Sci.*, vol. 13, no. 5, pp. 23–28, 2022, doi: 10.26483/ijarcs.v13i5.6903.

[3] M. M. Najafabadi, T. M. Khoshgoftaar, C. Kemp, N. Seliya, and R. Zuech, "Machine Learning for Detecting Brute Force Attacks at the Network Level," in *2014 IEEE International Conference on Bioinformatics and Bioengineering*, 2014, pp. 379–385. doi: 10.1109/BIBE.2014.73.

[4] A. Priandoyo, "Vulnerability Assessment untuk Meningkatkan Kesadaran Pentingnya Keamanan Informasi," *J. Sist. Inf.*, vol. 1, no. 2, pp. 73–83, 2006.

[5] R. Moskovitch, Y. Elovici, and L. Rokach, "Detection of unknown computer worms based on behavioral classification of the host," *Comput. Stat. Data Anal.*, vol. 52, no. 9, pp. 4544–4566, 2008, doi: 10.1016/j.csda.2008.01.028.

[6] S. Vijayakumar, K. S. P. Gowtham, N. Nigam, and R. V. R. Singh, "An Novel Approach in Designing a Security Workbench with Deep Learning Capabilities and Process Automation," *IEEE Reg. 10 Annu. Int. Conf. Proceedings/TENCON*, vol. 2019-Octob, pp. 263–268, 2019, doi: 10.1109/TENCON.2019.8929691.

[7] C. Virmani, T. Choudhary, A. Pillai, and M. Rani, "Applications of machine learning in cyber security," *Res. Anthol. Mach. Learn. Tech. Methods, Appl.*, pp. 621–641, 2022, doi: 10.4018/978-1-6684-6291-1.ch033.

[8] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," *ICISSP 2018 - Proc. 4th Int. Conf. Inf. Syst. Secur. Priv.*, vol. 2018-Janua, no. Cic, pp. 108–116, 2018, doi: 10.5220/0006639801080116.

[9] A. Fernández, S. García, F. Herrera, and N. V. Chawla, "SMOTE for Learning from Imbalanced Data: Progress and Challenges, Marking the 15-year Anniversary," *J. Artif. Intell. Res.*, vol. 61, pp. 863–905, 2018, doi: 10.1613/jair.1.11192.

[10] T. Kurita, "Principal component analysis (PCA)," in *Computer vision: a reference guide*, Springer, 2021, pp. 1013–1016.

[11] LP2M Universitas Medan Area, "Algoritma K-Nearest Neighbors (KNN) – Pengertian dan Penerapan," 2016. https://lp2m.uma.ac.id/2023/02/16/algoritma-k-nearest-neighbors-knn-pengertian-dan-penerapan/ (accessed May 28, 2024).

[12] Cornell University, "Lecture 2: k-nearest neighbors." https://www.cs.cornell.edu/courses/cs4780/2017sp/lectures/lecturenote02_kNN.html#:~:text=The k-NN algorithm&text=Formally Sx is defined,furthest point in Sx). (accessed Aug. 11, 2024).

[13] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. 13, no. 1, pp. 21–27, 1967, doi: 10.1109/TIT.1967.1053964.

[14] L. Breiman, "Random Forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.

[15] G. I. Webb, E. Keogh, and R. Miikkulainen, "Naive Bayes," *Encycl. Mach. Learn.*, vol. 15, no. 1, pp. 713–714, 2010.

[16] M. Murty and V. Devi, *Pattern recognition. An algorithmic approach*. 2011. doi: 10.1007/978-0-85729-495-1.

[17] R. E. Schapire, "The strength of weak learnability," *Mach. Learn.*, vol. 5, no. 2, pp. 197–227, 1990, doi: 10.1007/BF00116037.

[18] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997, doi: https://doi.org/10.1006/jcss.1997.1504.

[19] G. Ke *et al.*, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in *Advances in Neural Information Processing Systems*, 2017, vol. 30. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

[20] J. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," *Ann. Stat.*, vol. 29, 2000, doi: 10.1214/aos/1013203451.

[21] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, vol. 13-17-Augu, pp. 785–794, 2016, doi: 10.1145/2939672.2939785.

[22] J. F. Nunamaker, M. Chen, and T. D. M. Purdin, "Systems Development in Information Systems Research," *J. Manag. Inf. Syst.*, vol. 7, no. 3, pp. 89–106, Feb. 1990, [Online]. Available: http://www.jstor.org/stable/40397957

[23] J. Anderberg and N. Fathullah, "A machine learning approach to enhance the privacy of customers," 2019.

[24] "A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018)." https://registry.opendata.aws/cse-cic-ids2018 (accessed Feb. 21, 2024).

[25] R. Gunawan, Erik Suanda Handika, and Edi Ismanto, "Pendekatan Machine Learning Dengan Menggunakan Algoritma Xgboost (Extreme Gradient Boosting) Untuk Peningkatan Kinerja Klasifikasi Serangan Syn," *J. CoSciTech (Computer Sci. Inf. Technol.*, vol. 3, no. 3, pp. 453–463, 2022, doi: 10.37859/coscitech.v3i3.4356.