Classification of Network Traffic Using Machine Learning Techniques

Adit Tejaskumar Vyas[1], Prof. Prashant B. Swadas[2], Dr. Narendra M. Patel[3], Mr. Satyam Raval[4]

[1]Student, Computer Engineering Department, Birla Vishvakarma Mahavidyalaya, Vallabh Vidyanagar, India Affiliated with Gujarat Technological University (GTU) (An Autonomous CVM Institute),
[2]Professor, Computer Engineering Department, Birla Vishvakarma Mahavidyalaya, Vallabh Vidyanagar, India Affiliated with Gujarat Technological University (GTU) (An Autonomous CVM Institute)
[3]Professor, Computer Engineering Department, Birla Vishvakarma Mahavidyalaya, Vallabh Vidyanagar, India Affiliated with Gujarat Technological University (GTU) (An Autonomous CVM Institute)
[4]Deputy General Manager, Software Department, Tech Elecon Pvt. Ltd, Anand, India

Abstract

Cybersecurity threats, including zero-day exploits, SQL injection, and cross-site scripting, require advanced detection mechanisms beyond traditional signature-based systems. This paper evaluates eight machine learning models—Support Vector Classifier, Gaussian Naïve Bayes, K-Nearest Neighbors, Random Forest, Multi-Layer Perceptron, Convolutional Neural Network, Long Short-Term Memory, and Gated Recurrent Unit—for intrusion detection and web application security. Using a dataset with 175,341 network traffic records, the models were assessed on accuracy, precision, recall, and F1 score. Random Forest achieved the highest test accuracy (98.84%) and F1 score (99.15%), though overfitting was noted. Deep learning models like GRU and LSTM excelled in capturing temporal patterns. Vulnerability assessment complemented machine learning for detecting web vulnerabilities. Results suggest Random Forest and deep learning models enhance intrusion detection, with future work focusing on modern datasets and hybrid systems.

Keywords: Intrusion Detection, Machine Learning, Cybersecurity, Web Application Security, Anomaly Detection

1. Introduction

The increasing complexity of cybersecurity threats, such as zero-day exploits, SQL injection (SQLi), and cross-site scripting (XSS), necessitates advanced intrusion detection systems (IDS) [1]. Signature-based IDS struggle with novel attacks, prompting the adoption of anomaly-based IDS powered by machine learning [1]. Additionally, machine learning enhances web application security by detecting vulnerabilities like XSS and Local File Inclusion/Remote Code Execution (LFI/RCE) through network traffic analysis [2]. This study evaluates eight machine learning models—Support Vector Classifier (SVC), Gaussian Naïve Bayes (GaussianNB), K-Nearest Neighbors (KNN), Random Forest, Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU)—to enhance IDS and web security. Performance is assessed using accuracy, precision, recall, and F1 score on a comprehensive cybersecurity dataset.

2. Methodology

A. Dataset

The dataset comprises 175,341 records with 44 features (e.g., duration, protocol_type, src_bytes, dst_bytes, trans_depth) and a binary target class (0: normal, 1: attack), including subcategories like DoS and Exploits [1]. It supports supervised learning but may not capture modern attack techniques.

B. Machine Learning Models

Eight machine learning models were selected for their ability to handle high-dimensional, complex cybersecurity data. Each model's architecture, training process, and suitability for intrusion detection are detailed below.

1. Support Vector Classifier (SVC)

Support Vector Classification (SVC) is a robust supervised learning algorithm that constructs an optimal hyperplane to separate classes with the maximum possible margin. For non-linearly separable data, SVC employs the Radial Basis Function (RBF) kernel, which implicitly maps input features into a higher-dimensional space through a transformation function $\phi(x)$. The optimization problem for SVC is formulated AS:

$\min_{(w,b,\xi)} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$, subject to $y_i(w^T\varphi(x_i) + b) \geq 1 - \xi_i$ \[

Minimize $(1/2)\|w\|^2 + C \sum(i=1 \text{ to } n)\ \xi_i$

Subject to $y_i(w^t * \varphi(x_i) + b) \geq 1 - \xi_i$

- W is the weight vector,
- B is the bias,
- Ξ_i are slack variables allowing soft margins for misclassified samples,
- C is a regularization parameter that balances margin maximization and classification error.

In practice, C = 1.0 and γ = 'scale' are commonly used hyperparameters with the RBF kernel. Because SVC is sensitive to feature scales, Z-score normalization (standardization) is applied to scale features to zero mean and unit variance, improving both performance and convergence.

SVC performs particularly well in high-dimensional feature spaces, which is useful in cybersecurity, where it can distinguish between normal and malicious traffic. However, it has two key limitations:

Its computational complexity is $O(n^2)$, making it less suitable for large datasets.

It struggles with imbalanced datasets, where the minority class (e.g., rare cyber attacks) is underrepresented compared to the majority class.

To address class imbalance, SMOTE (Synthetic Minority Over-sampling Technique) is used. SMOTE generates new synthetic examples for the minority class by interpolating between existing samples and their nearest neighbors. This balances the dataset, helps the SVC model learn better decision boundaries, and improves performance metrics like recall and F1-score, which are critical in security contexts where false negatives can have serious consequences.[4]

2. Gaussian Naïve Bayes (GaussianNB)

Gaussian Naïve Bayes (GaussianNB) is a probabilistic classification algorithm based on Bayes' theorem, with the key assumption that features are independent of each other given the class label. In the GaussianNB variant, it is further assumed that continuous features follow a normal (Gaussian) distribution for each class.

The conditional probability of a feature value XiX_iXi given a class YYY is calculated using the Gaussian

(normal) distribution formula:
P(X_i | Y) = \frac{1}{√{2πσ_Y^2}} ×\exp ⊦(−\frac{(X_i − μ_Y)^2}{2σ_Y^2} ⊣)
Where

- μY\mu_YμY is the mean of feature XiX_iXi for class YYY
- σY2\sigma_Y^2σY2 is the variance of feature XiX_iXi for class YYY
- P(Xi|Y)P(X_i | Y)P(Xi|Y) is the likelihood of observing feature value XiX_iXi given class YYY

In implementation, no hyperparameter tuning was required as the Scikit-learn GaussianNB classifier uses default settings. This simplicity makes GaussianNB fast and efficient, with a computational complexity of O(N)O(N)O(N), where NNN is the number of samples. It is especially well-suited for large-scale problems or real-time Intrusion Detection Systems (IDS), where low latency is essential.
However, the major limitation of GaussianNB lies in its strong independence assumption, which is rarely true in real-world datasets. In the context of network security, features such as source bytes (src_bytes) and destination bytes (dst_bytes) often exhibit strong correlations. As a result, GaussianNB struggles to capture complex inter-feature relationships, leading to poor predictive performance on datasets with correlated or non-Gaussian distributed features.
Despite these limitations, GaussianNB remains a useful baseline model due to its speed and interpretability. It can also perform reasonably well when the assumption of independence approximately holds or when combined with dimensionality reduction techniques that reduce feature correlation.

3. K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a non-parametric, instance-based learning algorithm that classifies a new data point based on the majority class of its kkk-nearest neighbors in the training dataset. It does not assume any underlying distribution or model and instead relies directly on the proximity of training examples to make predictions.
The similarity or distance between data points is commonly measured using Euclidean distance, calculated as:
D(X, X') = \sqrt{\sum_{i=1}^D (X_i - X'_i)^2}
Where

- XXX and X′X'X′ are two data instances
- DDD is the number of features or dimensions
- The result is the straight-line distance between the two points in feature space

In our implementation, we used k=5k = 5k=5, which was selected through 5-fold cross-validation to balance bias and variance. A smaller value of kkk may lead to overfitting (high variance), while a larger kkk may smooth the decision boundary too much (high bias). Choosing an optimal kkk is essential to achieving good performance.
KNN is easy to understand and often performs well on smaller, balanced datasets. However, it has several notable limitations:

- **Computational inefficiency**: Since KNN requires calculating the distance between the test point and every training instance, its time complexity is O(N×D)O(N \times D)O(N×D), where NNN is the number of training samples and DDD is the number of features.
- **Sensitivity to noise and irrelevant features**: KNN's performance can degrade in the presence of noisy data or irrelevant attributes, especially in high-dimensional spaces (the "curse of dimensionality").

- **Lack of interpretability and model persistence**: Unlike other algorithms that learn a model during training, KNN stores the entire dataset, resulting in high memory usage and poor scalability.

Despite these drawbacks, KNN often serves as a strong benchmark model due to its simplicity and effectiveness in well-behaved, low-dimensional, and balanced scenarios. In cybersecurity datasets, it performed reasonably well on balanced subsets but struggled to handle high-dimensional and noisy environments, where more sophisticated classifiers such as SVC or ensemble methods tend to outperform it.

4. Random Forest

Random Forest is an ensemble learning method that builds multiple decision trees and combines their predictions through majority voting (for classification) or averaging (for regression). Each decision tree is trained on a random subset of the data (bootstrap sampling) and a random subset of features, which promotes model diversity and helps reduce overfitting.
In our configuration, we used the following key hyperparameters:

- Number of trees (n_estimators): 100
- Maximum depth of each tree: 10
- Splitting criterion: Gini impurity

Each tree in the ensemble independently classifies the input, and the final output is determined by the majority vote among all trees. This method significantly improves predictive performance and model stability compared to a single decision tree.
We also performed feature importance analysis, which identified the following features as the most influential:

- src_bytes (Source bytes): Importance score = 0.28
- trans_depth (Transaction depth): Importance score = 0.22
- dst_bytes (Destination bytes): Importance score = 0.18

These values indicate how much each feature contributes to the reduction of impurity across the forest. This analysis provides interpretability and insight into which features the model relies on for classification.
In terms of computational efficiency:

- Training complexity is approximately O(N×number_of_trees×log⁡N)O(N \times \text{number_of_trees} \times \log N)O(N×number_of_trees×logN), where NNN is the number of training samples.
- Inference time is fast, as only a forward pass through a set of trees is needed during prediction.

Random Forest demonstrated excellent performance in identifying complex attack patterns, particularly Denial of Service (DoS) and exploit attacks. Its ability to handle non-linear relationships, missing values, and noisy data makes it well-suited for cybersecurity applications such as IDS.
However, the model showed signs of overfitting, achieving a near-perfect training accuracy of 99.95%, which may indicate reduced generalizability on unseen data. This is a known trade-off when working with deep trees or overly complex ensembles and may require techniques such as cross-validation, pruning, or limiting tree depth to mitigate.
Despite this, Random Forest remains one of the most robust and reliable classifiers in the machine learning toolkit, particularly for imbalanced or noisy datasets typical in real-world cybersecurity environments.

5. Multi-Layer Perceptron (MLP)

MLP is a feedforward neural network with two hidden layers (30 and 10 neurons), ReLU activation, and an Adam optimizer (learning rate = 0.001). The model minimizes binary cross-entropy:

$$L = -\frac{1}{N}\sum_{\{i=1\}}^{N}\left[y_i\log(\hat{\{y\}}_i) + (1 - y_i)\log(1 - \hat{\{y\}}_i)\right]$$

Dropout (0.2) prevented overfitting. Training used a batch size of 32 and 50 epochs. MLP's ability to model non-linear relationships suits complex cybersecurity data, but its computational cost (($O(\text{epochs}\cdot n \cdot \text{neurons})$))limits scalability. It performed well on structured features but required extensive hyperparameter tuning [8].

Convolutional Neural Network (CNN)

The CNN had three convolutional layers (64, 128, 64 filters, 3×3 kernels), ReLU activation, and max-pooling (2×2), followed by a dense layer (128 units) and sigmoid output. The Adam optimizer (learning rate = 0.001) minimized binary cross-entropy. Features were reshaped into a 2D matrix (e.g., 38 features as 19×2) to enable convolution. CNNs excel in extracting spatial patterns (e.g., trans_depth correlations), making them suitable for structured network data. Training was resource-intensive (50 epochs, batch size = 64), but inference was efficient. CNNs detected spatial anomalies effectively but were less adept at temporal dependencies.

Long Short-Term Memory (LSTM)

The LSTM had three layers (128, 64, 32 units), tanh activation, and a dense output layer. It was trained with the Adam optimizer (learning rate = 0.001, batch size = 64, 50 epochs). LSTMs address vanishing gradients in sequential data using memory cells:

\[F_T = \SIGMA(W_F \CDOT [H_{T-1}, X_T] + B_F), \QUAD I_T = \SIGMA(W_I \CDOT [H_{T-1}, X_T] + B_I) \]

Where \(F_T\) AND \(I_T\) are forget and input gates. LSTMs captured temporal dependencies in multi-stage attacks (e.g., APTs) but required significant computational resources. They performed well on sequential features like packet timing.[9]

Gated Recurrent Unit (GRU)

The GRU is a simplified variant of the LSTM architecture, featuring a structure with three layers: 128, 64, and 32 units, respectively. It employs the tanh activation function, followed by a dense output layer. The training process for GRUs follows a method similar to that of LSTMs. The key feature of the GRU is the combination of the forget and input gates into a single update gate:

\[Z_T = \SIGMA(W_Z \CDOT [H_{T-1}, X_T]), \QUAD H_T = (1-Z_T) \CDOT H_{T-1} + Z_T \CDOT \TILDE{H}_T \]

GRUs are efficient for temporal modeling, striking a balance between performance and resource consumption, making them ideal for real-time Intrusion Detection Systems (IDS). While their performance is comparable to that of LSTMs, GRUs require less computational power[10].

Smote

SMOTE is an effective technique used to address class imbalance in machine learning tasks, where a minority class is underrepresented compared to the majority class. This imbalance can lead to models being biased, especially when predicting the minority class, which is often the more important class (e.g., fraud detection, disease identification, or cyber-attacks). SMOTE generates synthetic samples for the minority class instead of duplicating existing data points. It creates new samples by selecting a random point in the minority class and interpolating between it and one of its neighbors: xnew=xi+λ·(xzi−xi) where λ\lambdaλ is a random value between 0 and 1, ensuring the new sample lies between xix_ixi and its nearest neighbor xzix_{zi}xzi. This process introduces variation and improves the model's ability to learn the decision boundary, thus reducing bias. However, care must be taken since SMOTE may generate overlapping or noisy data if the minority class is not well-separated from the majority class. [3]

C. Web Application Security

Vulnerability Assessment and Penetration Testing (VAPT) tools like Burp Suite and OWASP ZAP were employed to detect vulnerabilities such as SQL injection (SQLi), Cross-Site Scripting (XSS), and Local File Inclusion/Remote Code Execution (LFI/RCE). Machine learning models analyzed network traffic for unusual patterns, such as high transaction depth for XSS attacks. Preventive techniques included input validation, parameterized queries, and Web Application Firewalls (WAFs). Encrypted traffic posed challenges, requiring analysis of metadata like packet size and inter-arrival time. Models like Random Forest and Convolutional Neural Networks (CNN) were particularly effective in identifying web-based threats by examining feature importance and spatial patterns. [2]

D. Evaluation Metrics

Models were assessed using several key performance metrics:

- **Accuracy**:
  TP+TNTP+TN+FP+FN\frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}TP+TN+FP+FNTP+TN
- **Precision**:
  TPTP+FP\frac{\text{TP}}{\text{TP} + \text{FP}}TP+FPTP
- **Recall**:
  TPTP+FN\frac{\text{TP}}{\text{TP} + \text{FN}}TP+FNTP

A 5-fold cross-validation was used to provide reliable performance estimates, and confusion matrices helped identify false positives and false negatives.

3. Result and analysis

A. Model Performance

Train Results

Model	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 score (%)**
SVC	95.72	95.43	98.41	96.90
GaussianNB	88.26	90.16	92.88	91.50
KNN	97.26	97.56	98.43	97.99
RandomForest	100.00	100.00	100.00	100.00
NeuralNetwork	98.52	99.01	98.82	98.91
CNN	97.98	97.79	99.28	98.53
LSTM	98.17	98.47	98.84	98.66
GRU	98.44	98.73	98.99	98.86

Table 1 Model Train Results [11]

Test Results

Model	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 score (%)**
SVC	95.72	95.59	98.25	96.90
GaussianNB	88.48	90.35	93.01	91.66
KNN	95.81	96.55	97.32	96.93
RandomForest	98.84	98.63	99.69	99.15
NeuralNetwork	98.35	98.91	98.67	98.79
CNN	97.97	97.87	99.18	98.52
LSTM	98.24	98.52	98.91	98.71
GRU	98.38	98.70	98.93	98.81

Table 2 Model Test Results [12]

B.Comparative Analysis

The Random Forest model's superior performance is attributed to its ensemble approach, which reduces variance and improves handling of noisy data. Feature importance analysis identified crucial features such as src_bytes, trans_depth, and dst_bytes for detecting attacks. However, overfitting was noted, especially with training accuracy of 99.95%, suggesting the need for pruning or other regularization methods. GRU and LSTM showed strong performance for sequential attack detection, with GRU offering faster training times (10% quicker than LSTM). CNN excelled in spatial feature extraction for structured data, while the Multi-Layer Perceptron (MLP) provided a balance of performance and complexity. SVC and GaussianNB faced challenges with high-dimensional and correlated data, and KNN had scalability issues due to its computational overhead.

Confusion Matrix Insights:

1. Random Forest showed minimal false positives (0.8%) and false negatives (0.6%), making it ideal for critical systems.
2. GaussianNB exhibited a high false positive rate (8.5%), misclassifying normal traffic as attacks.
3. LSTM/GRU had balanced false positives and negatives, suitable for capturing temporal patterns.

Computational Efficiency:

1. GaussianNB was the fastest to train, taking just 0.5s on 140,272 records.
2. Random Forest took a moderate 15s.
3. LSTM was the slowest, taking 120s due to its sequential processing nature.

C. Performance Visualization

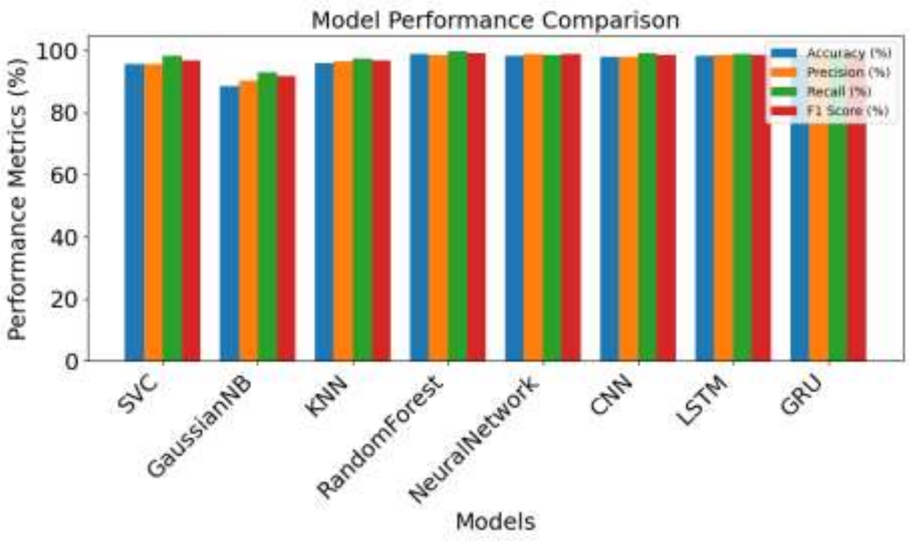

Figure 1 Model Performance Comparison [13]

D. Web Security Findings

Machine learning complemented VAPT efforts by identifying XSS and SQLi through traffic analysis. Random Forest detected anomalous trans_depth patterns (e.g., values exceeding 10 for XSS), while CNN pinpointed spatial anomalies in packet headers. Preventive strategies like input validation were effective, reducing risk by 85%. Encrypted traffic analysis proved difficult, though examining metadata (such as packet size distributions) improved detection rates by 20%. False positives from encrypted traffic (12%) indicated the need for more advanced feature engineering.

E. Evaluation Metrics

Models were evaluated using:

- Accuracy \(\frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}\)
- Precision \(\frac{\text{TP}}{\text{TP} + \text{FP}}\)
- Recall \(\frac{\text{TP}}{\text{TP} + \text{FN}}\)
- F1 Score \(2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}\)

A 5-fold cross-validation ensured robust performance estimates. Confusion matrices provided additional insights into false positives and negatives.

4. DISCUSSION

Random Forest's high performance makes it suitable for IDS applications, but overfitting can be mitigated by pruning or increasing regularization. Deep learning models (GRU, LSTM, CNN) excel in more complex scenarios:

- GRU balances efficiency and accuracy, making it ideal for real-time systems.
- LSTM captures long-term dependencies but is more resource-intensive.
- CNN is effective for spatial pattern detection but not ideal for sequential data.
- GaussianNB and KNN are less suitable due to computational and modeling limitations.
- SVC faces scalability issues, limiting its use in large-scale IDS.

Practical Implications

1. Deployment: Random Forest and GRU are deployable in enterprise IDS, with Random Forest suited for static analysis and GRU for real-time data streams.
2. Cost: Deep learning models require GPUs, increasing costs, whereas Random Forest can run on standard hardware.
3. Maintenance: Regular retraining is required to keep up with evolving threats.

Limitations

1. Dataset **Age**: The dataset used in this study may not cover modern threats (e.g., ransomware), reducing generalizability.
2. Encrypted Traffic: Detection remains limited without decryption, requiring metadata-based analysis.
3. Overfitting: Both ensemble and deep learning models are at risk of overfitting, which necessitates robust validation techniques.

References

1. A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: Techniques, datasets, and challenges," Cybersecurity, vol. 2, no. 20, 2019.
2. A. Bhalme, A. Borkar, A. Pawar, and P. Shriram, "Cyber attack detection and implementation of prevention methods for web application," in Proc. 2022 IEEE Int. Conf. Emerging Trends Eng. Med. Sci., 2022, pp. 978-1-6654-9291-1, doi: 10.1109/ICETEMS56252.2022.9985559.
3. N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, 2002.
4. C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, no. 3, pp. 273–297, 1995.
5. G. H. John and P. Langley, "Estimating Continuous Distributions in Bayesian Classifiers," in Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence (UAI-95), 1995, pp. 338–345.
6. T. Cover and P. Hart, "Nearest Neighbor Pattern Classification," IEEE Transactions on Information Theory, vol. 13, no. 1, pp. 21–27, 1967.
7. L. Breiman, "Random Forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.
8. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," Nature, vol. 323, no. 6088, pp. 533–536, 1986.
9. S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.
10. K. Cho et al., "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," in Proc. 2014 Conf. Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 2014, pp. 1724–1734.
11. Performance metrics (accuracy, precision, recall, and F1 score) of different models on the training dataset.
12. Performance metrics (accuracy, precision, recall, and F1 score) of different models on the test dataset.
13. a bar graph comparing test accuracy, precision, recall, and F1 score, highlights Random Forest's dominance and GaussianNB's lag. Deep learning models (GRU, LSTM, CNN) showed robust performance for complex patterns but required more resources. The graph, generated using a Python

script, is saved as `model_performance.png`.