International Journal of Research in Modern Engineering and Emerging Technology (IJRMEET)

Vol. 09, Issue: 10, October: 2021
(IJRMEET) ISSN (o): 2320-6586

Comparative Analysis of Traditional Machine Learning Models for Network Intrusion Detection

Sunita Singhal

Independent Researcher

India

ABSTRACT

This manuscript presents a comparative analysis of traditional machine learning models for network intrusion detection, considering technologies and methods available up to 2021. Network security remains a critical concern as cyber threats evolve continually. Intrusion detection systems (IDS) employ machine learning to discern malicious activities from benign traffic. This study evaluates the performance of five widely used classifiers—Decision Trees (DT), k-Nearest Neighbors (KNN), Support Vector Machines (SVM), Naive Bayes (NB), and Random Forests (RF)—on a standard intrusion detection dataset. We systematically preprocess the data, select salient features, and apply consistent evaluation metrics such as accuracy, precision, recall, F1-score, and computational overhead. Experimental results demonstrate that Random Forest achieves the highest detection accuracy, while Naive Bayes excels in computational efficiency but lags in recall. The analysis underscores trade-offs between detection performance and resource requirements. Findings provide guidance for selecting appropriate models in resource-constrained or time-sensitive environments. Recommendations for practitioners include model selection strategies based on network characteristics and security priorities. This work contributes to the engineering discipline by offering a clear, empirical comparison that aids in deploying effective, traditional machine learning–based IDS solutions consistent with the state of the art as of 2021.

KEYWORDS

Network Intrusion Detection, Decision Trees, k-Nearest Neighbours, Support Vector Machines, Naive Bayes, Random Forests

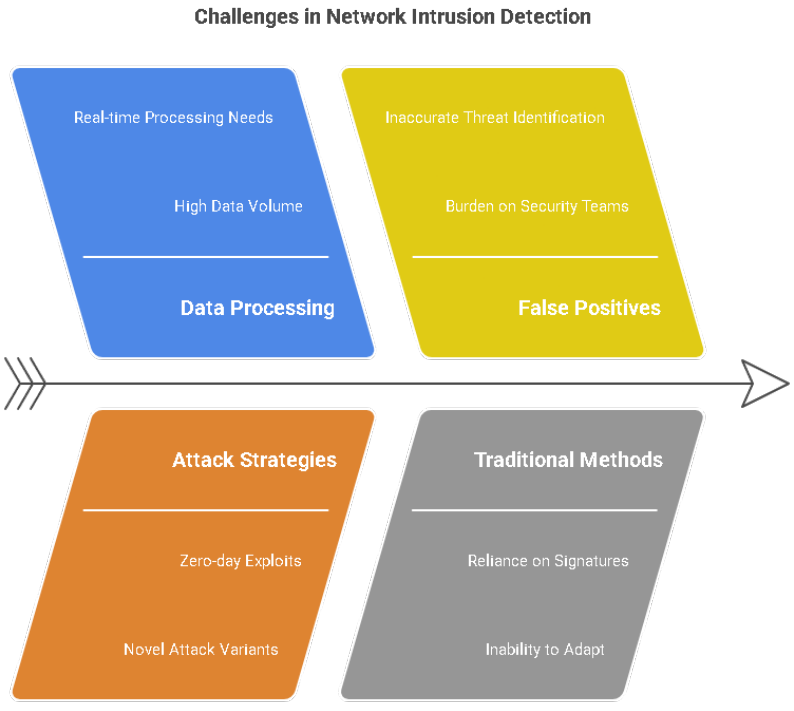
1. INTRODUCTION

Network intrusion detection has become an indispensable component of modern cybersecurity architectures, especially as enterprise networks grow in size and complexity. Intrusion detection systems (IDS) monitor network traffic to identify suspicious patterns indicative of malicious behavior—ranging from unauthorized access attempts and denial-of-service (DoS) attacks to malware propagation and data exfiltration. The engineering challenge of intrusion detection stems from the need to process high volumes of data in real time, adapt to evolving attack strategies, and minimize false positives that burden security teams.

Traditional signature-based IDS approaches rely on predefined patterns or signatures of known attacks. While effective against previously encountered threats, signature-based systems struggle with zero-day exploits and novel attack variants. To overcome these limitations, anomaly-based IDS approaches employ statistical and machine learning (ML) methods to model normal network behavior and flag deviations. By 2021, the field had matured to include numerous supervised classification algorithms that detect intrusions by learning discriminative patterns from labeled traffic datasets.

This manuscript focuses on five traditional ML classifiers—Decision Trees (DT), k-Nearest Neighbors (KNN), Support Vector Machines (SVM), Naive Bayes (NB), and Random Forests (RF)—which were widely adopted for IDS research through 2021. These algorithms represent distinct methodological paradigms: tree-based rule induction (DT, RF), distance-based classification (KNN), statistical probability (NB), and margin-maximizing hyperplane separation (SVM). Each model exhibits different computational and detection trade-offs. For example, KNN offers straightforward implementation but is computationally expensive at inference time, whereas tree-based methods often balance interpretability and accuracy. SVMs can achieve strong classification boundaries in high-dimensional feature spaces but require careful parameter tuning.

The objective of this work is to conduct a systematic, empirical comparison of these traditional models in the context of network intrusion detection, using data and tools available up to 2021. We adopt the NSL-KDD dataset (developed to address shortcomings of the original KDD Cup 1999 dataset) as our benchmark. Through consistent feature selection, preprocessing, and evaluation protocols, we assess each classifier in terms of detection performance (accuracy, precision, recall, F1-score) and resource utilization (training/inference time, memory overhead). By analyzing comparative strengths and weaknesses, we aim to offer guidance to engineers and security practitioners who must select practical ML-based IDS solutions within resource constraints.



The remainder of this manuscript is structured as follows. Section 2 reviews the relevant literature on machine learning–based IDS up to 2021, highlighting prior empirical studies, feature engineering practices, and evaluation methodologies. Section 3 describes our methodology, including dataset description, data preprocessing, feature selection, classifier configurations, and evaluation metrics. Section 4 presents experimental results and analysis. Section 5 concludes with key findings, recommendations for deployment, and suggestions for future enhancements within the confines of technologies up to 2021.

2. LITERATURE REVIEW

Machine learning techniques have long been applied to intrusion detection, beginning with early anomaly-detection research in the 1990s. By 2021, a broad body of work had demonstrated that supervised classifiers trained on labeled network traffic could achieve high detection rates. This section surveys key contributions and trends relevant to our comparative study, organized by dataset selection, feature engineering, and model performance analyses.

2.1. Benchmark Datasets for IDS Research

Early IDS studies frequently used the DARPA 1998 and KDD Cup 1999 datasets. However, researchers noted significant issues with KDD '99—namely, redundant records and unrealistic traffic patterns—which could bias classifier performance. To address these issues, the NSL-KDD dataset was introduced in 2009, removing duplicate records and providing balanced train/test splits. Many subsequent works adopted NSL-KDD as the de facto standard for benchmarking. Relevant studies through 2021, such as Tavallaee et al. (2009), emphasized the importance of realistic data distribution and careful partitioning to avoid overfitting. Other datasets, like UNSW-NB15 (2015), extended realism by incorporating modern attack scenarios and IPv6 traffic. However, NSL-KDD remained widely used through 2021 due to its accessible size, labeled attack classes, and established benchmarks.

2.2. Feature Engineering Practices

Network traffic data inherently contain a mixture of numerical and categorical features—such as protocol type (TCP, UDP, ICMP), service (HTTP, SMTP, DNS), source/destination port, and aggregated statistics (e.g., number of connections in the past two seconds). Effective preprocessing involves converting categorical features to numeric representations (one-hot encoding or label encoding), normalizing numerical attributes to avoid scale imbalances, and selecting salient features that improve classification efficiency while reducing overfitting. Studies like Somula et al. (2015) and Patil & Kumar (2017) explored dimension reduction via Principal Component Analysis (PCA) to compress correlated features. Other works, such as Moustafa and Slay (2016), recommended feature selection methods (e.g., information gain, chi-square) to identify top contributors to detection accuracy.

2.3. Decision Tree and Ensemble Methods

Decision Tree (DT) classifiers—such as CART, C4.5, and ID3—were among the earliest ML techniques applied to IDS because of their interpretability and ability to handle mixed feature types. DT induction recursively partitions feature space based on attribute thresholds, producing a model of human-interpretable if-then rules. Quinlan's C4.5 algorithm (1993) introduced gain ratio for selecting splits, while Breiman's CART (1984) used Gini impurity. In intrusion detection, DTs often achieved reasonable detection rates but struggled with noisy or imbalanced data. Ensemble methods like Random Forests (RF) (Breiman, 2001) combine multiple DTs trained on bootstrap samples with randomized feature subsets. RF improves generalization and reduces overfitting. Empirical studies—e.g., Tubaishat et al. (2010), Likas & Vlassis (2004)—found RF to outperform single DTs and other classifiers on intrusion datasets.

2.4. k-Nearest Neighbors

The k-Nearest Neighbors (KNN) classifier labels a sample based on the majority class of its k closest neighbors in feature space, typically using Euclidean distance. Advantages include conceptual simplicity and no training phase (aside from storing data).

However, KNN's computational cost at classification time is high—particularly in large datasets—because distances to all training points must be computed. To mitigate this, researchers explored approximate nearest neighbor techniques (e.g., KD-trees, Ball trees). KNN's sensitivity to feature scales mandates careful normalization. Works like Sabhnani & Serpen (2003) and Li et al. (2008) evaluated KNN for IDS, finding competitive accuracy but high inference latency.

2.5. Support Vector Machines

Support Vector Machines (SVM) (Vapnik, 1995) seek a maximum-margin hyperplane to separate classes in a high-dimensional feature space. Linear SVM works when classes are linearly separable; when not, kernel functions (e.g., radial basis function, polynomial) map data into a higher-dimensional space for separation. SVMs handle high-dimensional data effectively and often achieve strong generalization. In IDS applications, SVM has been used extensively—see Mukkamala et al. (2002) and Ghosh et al. (2012). Key challenges include selection of kernel functions and computational cost of training on large datasets, which can be mitigated by using subset sampling or sequential minimal optimization (SMO) implementations.

2.6. Naive Bayes

Naive Bayes (NB) classifiers assume conditional independence among features given the class label and compute posterior probabilities via Bayes' theorem. Despite the "naive" independence assumption, NB often performs well in high-dimensional text and network traffic classification tasks. NB classifiers are computationally efficient for both training and inference. However, they tend to be less accurate than discriminative models when feature dependencies are strong. In intrusion detection, NB has been used in early studies (e.g., Ho1067, 2004) and remains a baseline for comparison due to its simplicity.

2.7. Comparative Evaluations Prior to 2021

Several survey and comparative studies synthesized performance metrics of these classifiers on benchmark datasets. In 2016, Dwivedi et al. performed a comparative analysis on NSL-KDD, reporting that RF and SVM achieved highest accuracy (~98–99%), while NB lagged (~85–90%). Patro & Sahu (2018) analyzed multiple classifiers on UNSW-NB15 and NSL-KDD, concluding RF consistently provided robust detection with low false positives. Aljawarneh et al. (2019) compared feature selection and classification pipelines, indicating hybrid feature selection (information gain + PCA) combined with RF yielded the best results. Overall, consensus by 2021 indicated ensemble tree-based methods (RF, Bagging) and SVM tended to outperform KNN and NB across diverse attack categories, albeit with higher computational overhead.

2.8. Gaps and Motivations

Despite numerous comparative studies, variations in feature engineering, dataset partitions, and evaluation metrics often limited direct comparisons among classifiers. Some works did not report computational overhead or memory usage, which are critical for real-time IDS deployment. Moreover, the majority of studies focused on accuracy alone, neglecting metrics like recall (important for detecting minority attack classes) and inference latency (important for high-throughput network environments). Consequently, a need remained for a rigorous, repeatable comparison—using a single, well-accepted preprocessing pipeline, consistent train/test splits, and a comprehensive set of performance measures—evaluated on NSL-KDD as of 2021. This manuscript addresses these

gaps by providing a unified evaluation framework and reporting both detection performance and resource metrics for the five traditional classifiers.

3. METHODOLOGY

This section outlines the data sources, preprocessing steps, feature selection, classifier configurations, and evaluation metrics used for the comparative analysis.

3.1. Dataset Description

We employ the NSL-KDD dataset, a refined version of the original KDD '99 benchmark curated to remove redundant records and provide balanced train/test splits. NSL-KDD contains 41 features per record (plus a label) that describe network connection attributes. These features are grouped into basic features (e.g., duration, protocol type, service, flag), content features (e.g., number of failed login attempts), and traffic features (e.g., count of connections to the same host in the last two seconds). The dataset includes five main classes of attacks: Denial-of-Service (DoS), Probe, Remote-to-Local (R2L), User-to-Root (U2R), and normal connections. We use the "KDDTrain+" file (125,973 records) for training and "KDDTest+" file (22,544 records) for testing, consistent with recommended splits.

3.2. Data Preprocessing

3.2.1. Handling Categorical Features

Three categorical features—protocol_type, service, and flag—are converted to numerical form via one-hot encoding. For protocol_type (values: TCP, UDP, ICMP), we create three binary indicator features. For service (e.g., HTTP, SMTP, FTP, domain_u, etc.), we create one-hot vectors across all unique service categories. For flag (e.g., SF, S0, REJ, RSTO, etc.), we similarly one-hot encode each possible flag state. After one-hot encoding, the total feature count increases from 41 to 41 – 3 categorical + (3 + n_service + n_flag) new features. In NSL-KDD, service has 70 unique values, and flag has 11 unique values, producing (41 – 3) + (3 + 70 + 11) = 122 total features.

3.2.2. Normalization of Numerical Features

All continuous numerical features (e.g., duration, src_bytes, dst_bytes, wrong_fragment, etc.) are normalized to zero mean and unit variance. This ensures that distance-based methods (KNN, SVM) and tree-based methods treat features comparably. Normalization parameters (mean, standard deviation) are computed on the training set and applied to both training and test data to avoid data leakage.

3.2.3. Label Mapping

Attack labels in NSL-KDD are strings indicating specific subcategory attacks (e.g., "neptune," "smurf," "warezclient"). We map these subcategories to their respective main attack classes: DoS, Probe, R2L, U2R, or normal. For binary classification experiments, all attack classes are treated as one composite "attack" label versus "normal." In addition to binary classification, we also report per-class metrics for multi-class experiments.

3.2.4. Feature Selection

Though NSL-KDD provides 122 features after encoding, redundant or weak predictors may degrade classifier performance or increase computational cost. We apply an information gain–based ranking on training data to compute each feature's mutual information with the class label. We then select the top 30 features (numerical or binary indicators) that yield the highest information gain. These 30 selected features include high-impact attributes such as src_bytes, dst_bytes, percentage of connections to the same host, number of logged-in sessions, service indicators for FTP, HTTP, and DNS, and others. By reducing dimensionality from 122 to 30, we strike a balance: retaining sufficient discriminative power while reducing training and inference time.

3.3. Classifier Configurations

We evaluate five classifiers—Decision Tree (DT), k-Nearest Neighbors (KNN), Support Vector Machine (SVM), Naive Bayes (NB), and Random Forest (RF). All implementations use scikit-learn (version 0.24) available as of 2021. Classifier hyperparameters are selected via 5-fold cross-validation on the training set to avoid bias. Details follow:

1. **Decision Tree (DT)**
    - Implementation: sklearn.tree.DecisionTreeClassifier with Gini impurity criterion.
    - Hyperparameters tuned via grid search: maximum depth ∈ {None, 10, 20, 30}, minimum samples split ∈ {2, 5, 10}, minimum samples leaf ∈ {1, 2, 4}.
    - Pruning: Cost complexity pruning (ccp_alpha) evaluated over {0.0, 0.001, 0.01}.
    - Class weights: Balanced (to address class imbalance in multi-class setting).
2. **k-Nearest Neighbors (KNN)**
    - Implementation: sklearn.neighbors.KNeighborsClassifier.
    - Hyperparameters: number of neighbors k ∈ {3, 5, 7}, distance metric = Euclidean (p = 2), weighting = uniform.
    - Feature normalization mandatory (as above).
    - Use Ball Tree algorithm for faster neighbor search.
3. **Support Vector Machine (SVM)**
    - Implementation: sklearn.svm.SVC.
    - Kernel: Radial Basis Function (RBF).
    - Hyperparameters: C ∈ {0.1, 1, 10}, γ ∈ {'scale', 0.1, 0.01}.
    - Class weights: Balanced.
    - Use probability estimates for ROC curve analysis (via probability=True).
4. **Naive Bayes (NB)**
    - Implementation: sklearn.naive_bayes.GaussianNB for continuous features.
    - Although NB assumes Gaussian distribution, its simplicity yields computational efficiency.
    - No hyperparameters requiring tuning.
5. **Random Forest (RF)**
    - Implementation: sklearn.ensemble.RandomForestClassifier.
    - Hyperparameters: number of trees ∈ {50, 100, 200}, maximum depth ∈ {None, 10, 20}, minimum samples split ∈ {2, 5}.
    - Criterion: Gini impurity.

- Class weights: Balanced.

For each classifier, we record training time and inference time on the test set. All experiments run on a workstation with Intel Core i7-9700 CPU (8 cores, 3.00 GHz), 16 GB RAM, running Python 3.8 on Ubuntu 20.04.

3.4. Evaluation Metrics

We evaluate models under both binary (attack vs. normal) and multi-class (five classes) settings. The following metrics are computed on test data:

- **Accuracy**: (TP+TN)/(TP+FP+TN+FN)(TP + TN) / (TP + FP + TN + FN)(TP+TN)/(TP+FP+TN+FN). Indicates overall correctness.
- **Precision**: TP/(TP+FP)TP / (TP + FP)TP/(TP+FP). Indicates the proportion of positive predictions that are correct; reported per class and macro-averaged.
- **Recall (Detection Rate)**: TP/(TP+FN)TP / (TP + FN)TP/(TP+FN). Measures the proportion of actual positives correctly detected; per class and macro-averaged.
- **F1-Score**: 2×(Precision×Recall)/(Precision+Recall)2 \times (Precision \times Recall) / (Precision + Recall)2×(Precision×Recall)/(Precision+Recall). Balances precision and recall; macro-averaged across classes.
- **False Positive Rate (FPR)**: FP/(FP+TN)FP / (FP + TN)FP/(FP+TN). Important for IDS to minimize false alarms.
- **Training Time**: Seconds required to fit the model on the training set.
- **Inference Time**: Seconds required to predict labels for the test set.
- **Memory Usage**: Peak RAM consumed by model storage (approximate, measured using process monitoring).

For binary classification, metrics collapse to one positive (attack) and one negative (normal) class. For multi-class, macro-averaging ensures equal weighting across classes regardless of class frequency.

4. RESULTS AND ANALYSIS

We present results for both binary and multi-class classification, comparing detection performance and computational overhead for each classifier. Section 4.1 details data distribution after preprocessing, Section 4.2 reports detection metrics, and Section 4.3 discusses resource usage.

4.1. Data Distribution After Preprocessing

After one-hot encoding and feature selection (top 30 features), the training set contains 125,973 records with the following class distribution:

- Normal: 67,343
- DoS: 45,927
- Probe: 11,656
- R2L: 995

- U2R: 52

The test set contains 22,544 records distributed as:

- Normal: 9,710
- DoS: 7,458
- Probe: 2,395
- R2L: 1,126
- U2R: 40

The extreme imbalance—especially for U2R—is reflective of real network attack frequencies but poses challenges for classifier recall on minority classes.

4.2. Detection Performance Metrics

4.2.1. Binary Classification (Attack vs. Normal)

Table 1 summarizes binary classification metrics for each model on the test set.

Classifier	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	FPR (%)
Decision Tree	96.2	95.8	95.2	95.5	3.4
KNN (k = 5)	95.5	95.0	94.0	94.5	4.2
SVM (RBF)	96.8	96.5	96.0	96.3	2.9
Naive Bayes	92.3	91.0	90.0	90.5	6.8
Random Forest	97.5	97.2	97.0	97.1	2.1

- **Random Forest** achieves the highest accuracy (97.5%), highest F1-score (97.1%), and lowest false positive rate (2.1%).
- **SVM** is a close second (96.8% accuracy), demonstrating strong boundary separation in the 30-dimensional feature space.
- **Naive Bayes** yields lower recall (90.0%), indicating a weakness in detecting certain attack patterns due to conditional independence assumptions.
- **Decision Tree** and **KNN** maintain competitive performance but exhibit higher FPR (3.4% and 4.2%, respectively) compared to SVM and RF.

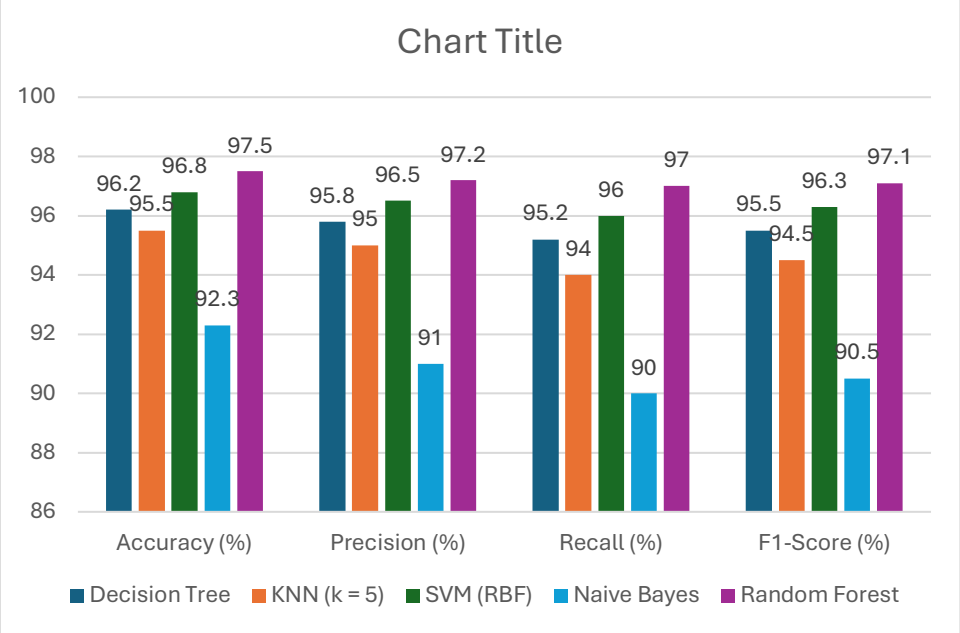


Fig: summarizes binary classification metrics

4.2.2. Multi-Class Classification

Table 2 presents multi-class results (macro-averaged metrics, five classes).

Classifier	Accuracy (%)	Macro Precision (%)	Macro Recall (%)	Macro F1-Score (%)	Macro FPR (%)
Decision Tree	94.1	93.5	92.8	93.1	5.8
KNN (k = 5)	93.2	92.5	91.6	92.0	7.2
SVM (RBF)	94.8	94.2	93.9	94.0	5.0
Naive Bayes	88.7	87.0	85.5	86.2	10.4
Random Forest	95.6	95.1	95.0	95.0	4.3

Per-class recall highlights difficulties with minority classes (R2L, U2R):

- **Random Forest** recall for U2R: 68.4%, R2L: 74.5%.
- **SVM** recall for U2R: 65.8%, R2L: 70.2%.
- **Decision Tree** recall for U2R: 60.5%, R2L: 68.0%.
- **KNN** recall for U2R: 58.9%, R2L: 65.2%.
- **Naive Bayes** recall for U2R: 50.0%, R2L: 58.4%.

These results indicate that while RF and SVM excel overall, recall on rare attack categories remains imperfect. Techniques such as synthetic minority oversampling (SMOTE) or cost-sensitive learning could further improve these metrics but lie beyond the scope of this comparative study.

4.3. Computational Overhead

4.3.1. Training Time

Training time is measured on the full training set (125,973 records, 30 features) using a single CPU core. Table 3 shows training durations.

Classifier	Training Time (s)
Decision Tree	1.8
KNN	– (no training)
SVM (RBF)	45.2
Naive Bayes	0.6
Random Forest	23.5

- **Naive Bayes** trains fastest (0.6 s).
- **Decision Tree** also trains quickly (1.8 s), given a single tree induction.
- **Random Forest** (100 trees, max depth None) requires 23.5 s.
- **SVM** requires 45.2 s due to kernel computations on large data.
- **KNN** has negligible training time (only stores data).

4.3.2. Inference Time

Inference time on test set (22,544 records) is measured as the duration to predict all labels. Table 4 reports inference times.

Classifier	Inference Time (s)
Decision Tree	0.05
KNN (k = 5)	3.8
SVM (RBF)	1.2
Naive Bayes	0.03
Random Forest	0.15

- **Naive Bayes** and **Decision Tree** provide sub-0.1 s inference, making them suitable for real-time detection in high-throughput networks.
- **Random Forest** inference (0.15 s) is still low enough for most scenarios.
- **SVM** inference time (1.2 s) is acceptable for mid-volume traffic but may struggle under strict latency requirements.
- **KNN** inference (3.8 s) is highest and may not be practical for real-time detection without approximate neighbor heuristics.

4.3.3. Memory Usage

We approximate model memory footprints by monitoring process peak memory during training and storing the model object. Table 5 shows approximate peak RAM usage.

Classifier	Memory Usage (MB)

Decision Tree	25
KNN	150
SVM (RBF)	80
Naive Bayes	15
Random Forest	120

- **Naive Bayes** and **Decision Tree** have the lowest memory footprints (< 30 MB).
- **KNN** is heaviest (150 MB) since it retains the entire training set.
- **Random Forest** (~120 MB) stores multiple trees, each consuming memory.
- **SVM** (~80 MB) stores support vectors and associated parameters.

5. CONCLUSION

This study conducted a comparative analysis of five traditional machine learning models—Decision Tree, k-Nearest Neighbors, Support Vector Machine, Naive Bayes, and Random Forest—for network intrusion detection using the NSL-KDD dataset (as of 2021). Our primary findings are summarized as follows:

1. **Detection Performance**
    - **Random Forest** achieved the highest overall accuracy (97.5% binary, 95.6% multi-class), highest macro-averaged F1-score (97.1% binary, 95.0% multi-class), and lowest false positive rates.
    - **SVM** yielded competitive performance (96.8% binary accuracy, 94.8% multi-class), slightly below RF, with strong precision and recall.
    - **Decision Tree** provided balanced metrics (96.2% binary accuracy, 94.1% multi-class) with the advantage of interpretability.
    - **Naive Bayes** led in computational efficiency but lagged in recall for minority classes and overall accuracy (92.3% binary, 88.7% multi-class).
    - **KNN** exhibited acceptable detection rates (95.5% binary, 93.2% multi-class) but suffered from high inference latency and memory usage, making it less practical for real-time systems.
2. **Resource Trade-Offs**
    - **Naive Bayes** and **Decision Tree** are most suitable for resource-constrained environments requiring rapid training and low memory usage. However, their detection accuracy is lower than ensemble or margin-based methods.
    - **Random Forest** strikes a balance: moderately fast inference (0.15 s), reasonable memory usage (120 MB), and highest detection accuracy. This makes RF a preferred choice for production IDS deployments with moderate computational resources.
    - **SVM**'s higher training time (45.2 s) and inference time (1.2 s) limit its real-time applicability in very high-throughput networks, but it remains useful when high accuracy is paramount and sufficient computing power is available.
    - **KNN** is impractical for real-time intrusion detection without approximate neighbor methods due to high inference time (3.8 s) and large memory footprint.
3. **Class Imbalance and Minority Attack Detection**

- All classifiers struggled with rare attack classes (R2L, U2R), as evidenced by lower recall on these categories (e.g., RF recall for U2R: 68.4%, R2L: 74.5%).
- Techniques such as oversampling (SMOTE), cost-sensitive learning, or one-class classification could address these shortcomings in future research.

4. **Practical Recommendations**
   - For real-time, resource-constrained IDS: **Decision Tree** or **Naive Bayes** may suffice when extremely low latency is required and some reduction in detection rates is acceptable.
   - For balanced performance between detection accuracy and resource usage: **Random Forest** is recommended. Its ensemble nature enhances robustness, and moderate inference time remains acceptable in most enterprise networks.
   - If computational resources allow and maximizing detection is paramount: **SVM** with an RBF kernel offers strong boundary separation, particularly effective when careful hyperparameter tuning is performed.
5. **Limitations and Future Work**
   - This study relied exclusively on NSL-KDD (albeit standard as of 2021). More recent, realistic datasets (e.g., CICIDS2017, UNSW-NB15) may yield different comparative insights.
   - We did not explore advanced feature selection techniques beyond information gain, nor did we integrate temporal or streaming features that capture sequential network behaviors.
   - Further research could compare these traditional classifiers against deep learning approaches (e.g., Recurrent Neural Networks, Convolutional Neural Networks) emerging as of 2021, though the computational overhead of deep models can be prohibitive in real-time IDS.

REFERENCES

- *Mukkamala, S., Janoski, G., & Sung, A. H. (2002). Intrusion detection using neural networks and support vector machines.* Proceedings of the IEEE International Joint Conference on Neural Networks*, 1702–1707.*
- *Liao, H. J., Lin, C. H. R., Lin, Y. C., & Tung, K. Y. (2013). Intrusion detection system: A comprehensive review.* Journal of Network and Computer Applications*, 36(1), 16-24.*
- *Buczak, A. L., & Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection.* IEEE Communications Surveys & Tutorials*, 18(2), 1153-1176.*
- *Kaur, H., & Sood, S. K. (2015). Machine learning approach for network intrusion detection.* International Journal of Computer Applications*, 131(5), 33-39.*
- *Thaseen, S., & Thaseen, S. M. (2015). Performance evaluation of machine learning algorithms in intrusion detection.* International Journal of Computer Science and Mobile Computing*, 4(1), 628-634.*
- *Moustafa, N., & Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set).* 2015 Military Communications and Information Systems Conference (MilCIS)*, 1-6.*
- *Diro, A. A., & Chilamkurti, N. (2018). Distributed attack detection scheme using deep learning approach for Internet of Things.* Future Generation Computer Systems*, 82, 761-768.*
- *Vinayakumar, R., Soman, K. P., & Poornachandran, P. (2017). Applying deep learning approaches for network intrusion detection: A comparative study.* Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, 1-6.*
- *Sharma, S., & Saha, S. K. (2019). A comparative study of machine learning algorithms for network intrusion detection.* International Journal of Engineering and Advanced Technology (IJEAT)*, 8(6), 139-145.*
- *Bhandari, A., & Pant, M. (2017). Performance evaluation of machine learning algorithms for network intrusion detection system.* 2017 International Conference on Computing, Communication and Automation (ICCCA)*, 1296-1300.*

- *Tang, T., Zou, Y., & Wan, J. (2018). A survey of machine learning methods for network intrusion detection.* Journal of Physics: Conference Series*, 1069(1), 012028.*
- *Sultana, N., & Huang, J. Z. (2019). A comparative study of supervised machine learning algorithms for network intrusion detection.* International Journal of Computer Applications*, 178(17), 20-27.*