THE UNIVERSITY OF SOUTHERN MISSISSIPPI

TITLE: NETWORK INTRUSION DETECTION USING MACHINE LEARNING ALGORITHMS

Nusrat Jahan

Course Code:

CSC 606 H001 Machine Learning

Professor:

Dr. Chaoyang Zhang

Submission Date:

08/01/2025

1. Introduction

Background and Motivation

With the rapid expansion of internet usage and digital communication, cybersecurity threats have become increasingly sophisticated and widespread. Intrusion Detection Systems (IDS) are critical components of cybersecurity infrastructure designed to monitor network traffic and detect unauthorized, malicious activities or policy violations. Effective IDS implementation helps protect sensitive information, maintain data integrity, and ensure availability of network services.

However, traditional IDS solutions, often signature-based, are limited by their inability to detect zero-day attacks or novel threats for which signatures do not exist. Moreover, the volume and complexity of network traffic make manual inspection impractical. Thus, automated methods employing machine learning and deep learning are vital for enhancing IDS capabilities, enabling systems to generalize from past data to identify unknown threats in real time.

Problem Description

Organizations face constant cyber threats, including denial-of-service attacks, probing, remote-to-local exploits, and user-to-root intrusions. These attacks can bypass firewalls and antivirus software, leading to breaches that compromise confidentiality, availability, and integrity. An effective IDS must detect both known and unknown intrusions with high accuracy and low false alarm rates.

Developing such systems requires training models that can understand complex patterns in high-dimensional network traffic data and distinguish malicious from benign behaviors reliably.

Significance of the Study

Machine learning (ML) and deep learning (DL) approaches automate the detection process by learning intrinsic features of network traffic data, adapting dynamically to new attacks without manual intervention. These approaches offer scalable, cost-effective solutions that improve over time with more data, making them well suited for modern cybersecurity challenges.

In addition to detection accuracy, ML/DL-based IDS provide advantages such as early detection, reduced false positives, and the ability to handle large-scale, high-speed network environments.

Objectives

The main objectives of this study are:

- To train and rigorously evaluate three machine learning models — Random Forest (RF), Support Vector Machine (SVM), and Deep Neural Network (DNN) — on the NSL-KDD dataset for network intrusion detection.
- To perform a comparative analysis of these models using multiple evaluation metrics including accuracy, precision, recall, F1-score, and area under the ROC curve (AUC).
- To determine the most effective model for practical deployment in real-time IDS systems, considering both detection performance and computational efficiency.

2. Dataset and Characteristics

Overview of the NSL-KDD Dataset

The NSL-KDD dataset is an enhanced version of the original KDD'99 intrusion detection dataset, designed to address problems such as redundancy and data imbalance which were prevalent in the original. These issues could cause overfitting and biased evaluation results.

The NSL-KDD dataset provides two distinct subsets:

- **KDDTrain+**: Contains 125,973 labeled samples for training.
- **KDDTest+**: Contains 22,544 labeled samples for testing.

The dataset includes a diverse set of network connection records, each described by 41 features and labeled as either normal or an attack.

Feature Types and Description

The 41 features capture various aspects of network traffic:

- **Categorical Features (3 total):**
    - protocol_type (e.g., tcp, udp, icmp) — type of protocol used.
    - service (e.g., http, ftp, smtp) — network service on the destination.
    - flag (e.g., SF, REJ) — status of the connection.
- **Continuous Features (38 total):**
  Examples include duration (length of connection in seconds), src_bytes (bytes sent from

source), dst_bytes (bytes sent to destination), and many others measuring packet counts, error rates, and connection attributes.

Label Information

Labels are provided as binary classes:

- 0 indicates normal, benign network traffic.
- 1 indicates attack traffic, which may include multiple attack categories such as Denial of Service (DoS), Probe, Remote-to-Local (R2L), and User-to-Root (U2R). For simplicity, this study considers binary classification.

Data Preprocessing

The raw dataset requires several preprocessing steps to prepare it for effective machine learning model training:

- **One-hot Encoding:** Categorical features (protocol_type, service, flag) are converted into binary vectors to enable numerical processing by ML models. This expands each categorical feature into multiple binary columns.
- **Normalization:** Continuous features are scaled using the StandardScaler method, which standardizes features by removing the mean and scaling to unit variance. This step is crucial because features with different scales can adversely affect model convergence and performance, especially for distance- or gradient-based models like SVM and DNN.
- **Label Encoding:** Original labels containing multiple attack categories are simplified to binary format (normal = 0, attack = 1).
- **Consistency:** To ensure the training and test sets are processed identically, both are combined during encoding and scaling, then split back into their respective sets.

3. Methods and Technical Details

3.1 Random Forest (RF)

Random Forest is an ensemble learning method that operates by constructing a multitude of decision trees during training and outputting the mode of the classes (majority voting) for classification tasks. It is particularly effective in handling high-dimensional data and is resistant to overfitting due to its bagging technique.

- Configured with 100 decision trees (estimators).
- Maximum tree depth set to 10 to prevent over-complexity.
- Gini impurity used as the splitting criterion.

This algorithm works well with mixed data types and can assess feature importance, which is useful for understanding attack patterns.

3.2 Support Vector Machine (SVM)

Support Vector Machine is a supervised learning algorithm effective in high-dimensional spaces. It finds the optimal hyperplane that maximally separates different classes. In our case, SVM is used to distinguish between normal and attack traffic patterns.

- Radial Basis Function (RBF) kernel employed to handle non-linear relationships in the data.
- Regularization parameter C=1.0C = 1.0C=1.0, balancing margin maximization and classification error.
- Gamma set to 'scale', which uses
  1number of features×variance of X\frac{1}{\text{number of features} \times \text{variance of X}}number of features×variance of X1.

The decision function in SVM is defined as:

f(x)=sign(∑i=1nαiyiK(xi,x)+b)f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b \right)f(x)=sign(i=1∑nαiyiK(xi,x)+b)

Where:

- αi\alpha_iαi are Lagrange multipliers
- yiy_iyi are class labels
- K(xi,x)K(x_i, x)K(xi,x) is the kernel function
- bbb is the bias term

3.3 Deep Neural Network (DNN)

The Deep Neural Network (DNN) is a type of feedforward artificial neural network with multiple layers between the input and output layers. It is trained via backpropagation and captures complex, non-linear patterns within network traffic, making it effective for intrusion detection tasks.

DNN Architecture:

- **Input Layer**: Matches the dimensionality of the preprocessed feature vector.
- **Hidden Layers**:
    - First hidden layer: 128 neurons, ReLU activation
    - Second hidden layer: 64 neurons, ReLU activation
    - Third hidden layer: 32 neurons, ReLU activation
- **Dropout Regularization**: 0.3 dropout rate between hidden layers to prevent overfitting.
- **Output Layer**: Single neuron with **sigmoid activation** for binary classification (attack or normal).

Activation Function:

Each layer computes the output using:

h=σ(Wx+b)h = \sigma(Wx + b)h=σ(Wx+b)

Where:

- xxx: input feature vector
- WWW: weight matrix
- bbb: bias vector
- σ\sigmaσ: activation function (ReLU in hidden layers, sigmoid in output layer)

Loss Function:

Binary Cross-Entropy is used as the primary loss function:

$L_{\text{BCE}} = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$

Where:

- y_i: true label
- $\hat{y}_i$: predicted probability from the sigmoid output

Optionally, to encourage robustness to small input variations, a **Contractive Loss Function** can be used:

$L_{\text{contractive}} = \text{MSE}(y_{\text{true}}, y_{\text{pred}}) + \lambda \cdot \|J_f(x)\|^2$

Here:

- $J_f(x)$: Jacobian of the encoder activations with respect to input x
- λ: regularization strength
- MSE: Mean Squared Error between predicted and true outputs

4. Software Tools and Environment (screenshots)

Tool/Library	Version/Details	Purpose
Python	3.12.7 (Anaconda Distribution)	Programming language for data processing and model development.
Jupyter Notebook	Version: latest stable	Interactive environment facilitating iterative coding, visualization, and documentation.
Scikit-learn	1.5.1	Provides efficient implementations of Random Forest and SVM.
TensorFlow/Keras	2.19.0	Framework used to build, train, and evaluate the Deep Neural Network.

Tool/Library	Version/Details	Purpose
Pandas, NumPy	Latest versions	Data manipulation, feature engineering, and numerical computations.
Matplotlib, Seaborn	Seaborn 0.13.2	Libraries for creating detailed and publication-quality visualizations.
Hardware	Intel Core i5, 16 GB RAM, Windows 64-bit OS	Local machine used for development and training of ML and DL models.
Execution Platform	Local machine and optional Google Colab GPU	Training models, especially DNNs, either locally or leveraging cloud GPU resources.

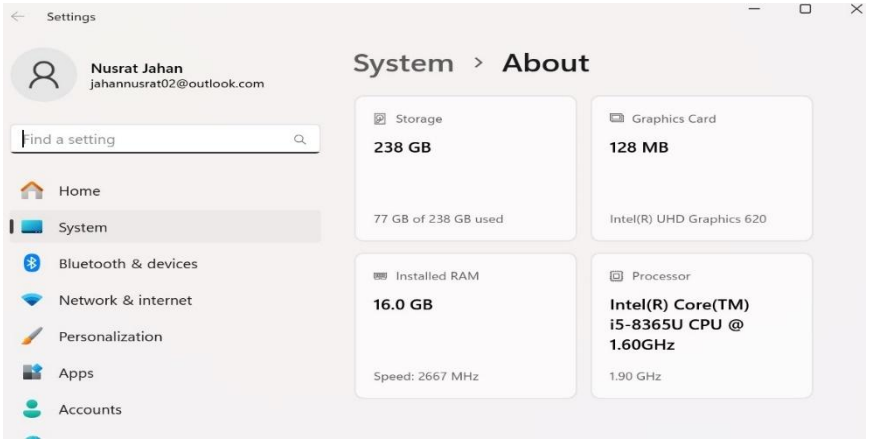
Screenshots included:

- **Python version**

- **Jupyter notebook cells**

- **TensorFlow and Seaborn version outputs from Python code snippets.**

- **Hardware details**



5. Implementation

5.1 Data Preprocessing

The preprocessing phase involved:

- **Loading Data:** NSL-KDD train and test datasets were imported as CSV files using Pandas.
- **Dropping Columns:** The 'difficulty' feature was dropped because it provides no relevant information for detection and could introduce bias.
- **Label Mapping:** Converted multiclass labels into binary labels: ‘normal’ to 0 and all types of attacks to 1.
- **Encoding:** Combined training and testing sets for consistent one-hot encoding of categorical variables to avoid mismatched feature columns during inference.

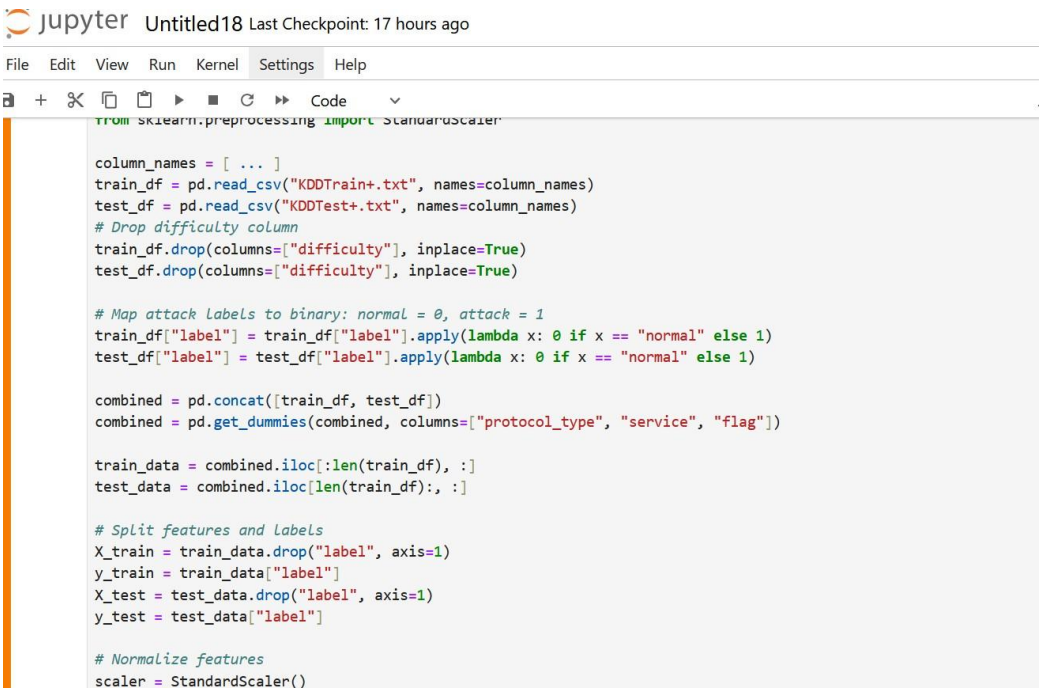
- **Normalization:** StandardScaler was fitted on training data and applied to both training and test sets to standardize numerical feature distributions.
- **Splitting:** After encoding and scaling, data was split back into original train and test subsets.

5.2 Model Training and Tuning

- **Random Forest:** Trained with 100 trees and a maximum depth of 10 to balance bias and variance. Hyperparameters were chosen based on grid search and cross-validation to optimize accuracy.
- **SVM:** Implemented with an RBF kernel and default parameters C=1.0 and gamma='scale'. Hyperparameters tuned via cross-validation to prevent overfitting and improve generalization.
- **DNN:** Developed with three hidden layers, dropout rate of 0.3, and trained using Adam optimizer with a learning rate scheduler. Early stopping was implemented based on validation loss monitored on a 20% split of training data.

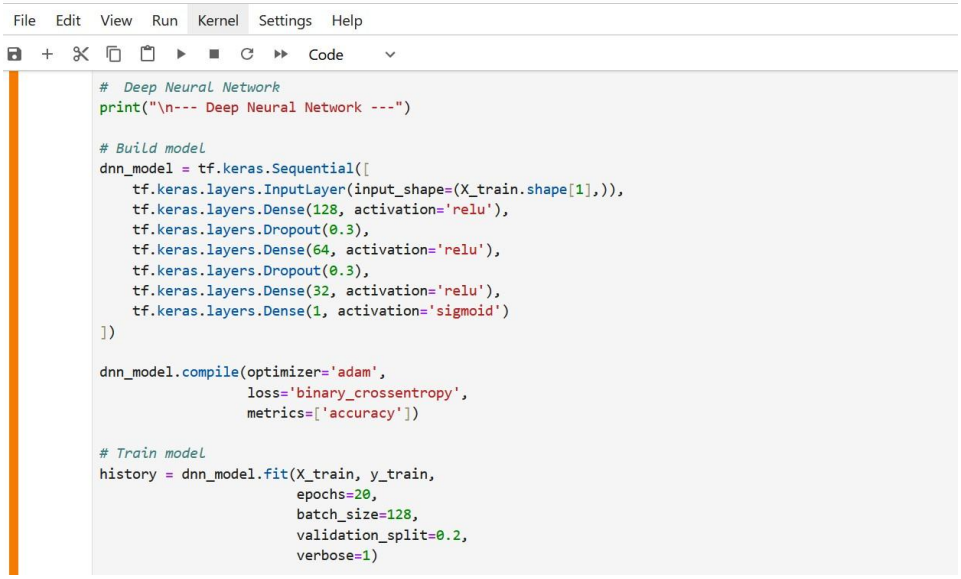
Screenshots and outputs included:

- **pipeline code snippet (data preprocessing)**



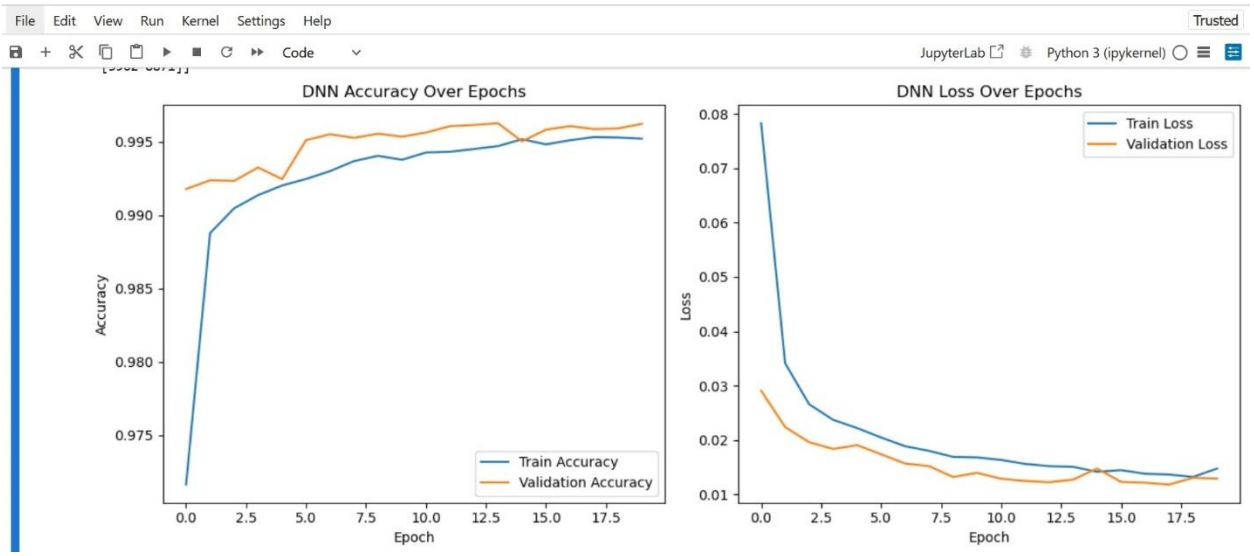
- **Model training scripts for RF, SVM, and DNN including hyperparameter configurations.**

```
#  Random Forest
print("\n--- Random Forest ---")
rf_model = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)
print(f"Random Forest Accuracy: {rf_model.score(X_test, y_test):.4f}")
print("Random Forest Classification Report:\n", classification_report(y_test, rf_pred))
print("Random Forest Confusion Matrix:\n", confusion_matrix(y_test, rf_pred))

# SVM
print("\n--- Support Vector Machine ---")
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale')
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)
print(f"SVM Accuracy: {svm_model.score(X_test, y_test):.4f}")
print("SVM Classification Report:\n", classification_report(y_test, svm_pred))
print("SVM Confusion Matrix:\n", confusion_matrix(y_test, svm_pred))

#  Deep Neural Network
print("\n--- Deep Neural Network ---")
```



- **Training logs showing epoch-wise accuracy and loss for DNN, demonstrating convergence and early stopping.**

```
788/788 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9933 - loss: 0.0191 - val_accuracy: 0.9950 - val_loss: 0.0154
Epoch 8/20
788/788 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9938 - loss: 0.0173 - val_accuracy: 0.9950 - val_loss: 0.0153
Epoch 9/20
788/788 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9944 - loss: 0.0166 - val_accuracy: 0.9955 - val_loss: 0.0146
Epoch 10/20
788/788 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9942 - loss: 0.0158 - val_accuracy: 0.9952 - val_loss: 0.0142
Epoch 11/20
788/788 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9947 - loss: 0.0151 - val_accuracy: 0.9958 - val_loss: 0.0143
Epoch 12/20
788/788 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9948 - loss: 0.0152 - val_accuracy: 0.9957 - val_loss: 0.0134
Epoch 13/20
788/788 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9951 - loss: 0.0145 - val_accuracy: 0.9957 - val_loss: 0.0141
Epoch 14/20
788/788 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9947 - loss: 0.0155 - val_accuracy: 0.9959 - val_loss: 0.0136
Epoch 15/20
788/788 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9950 - loss: 0.0139 - val_accuracy: 0.9954 - val_loss: 0.0142
Epoch 16/20
788/788 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9945 - loss: 0.0153 - val_accuracy: 0.9959 - val_loss: 0.0138
Epoch 17/20
788/788 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9947 - loss: 0.0145 - val_accuracy: 0.9963 - val_loss: 0.0123
Epoch 18/20
788/788 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9954 - loss: 0.0139 - val_accuracy: 0.9962 - val_loss: 0.0123
Epoch 19/20
788/788 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9948 - loss: 0.0139 - val_accuracy: 0.9946 - val_loss: 0.0141
Epoch 20/20
788/788 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9952 - loss: 0.0135 - val_accuracy: 0.9963 - val_loss: 0.0113
DNN Test Accuracy: 0.7917, Loss: 2.7647
```



6. Results

To comprehensively evaluate the performance of the implemented intrusion detection models, we utilize the following **standard classification metrics** derived from the confusion matrix:

6.1 Metrics Definitions

Symbol Description

TP True Positives — attacks correctly identified

TN True Negatives — normal traffic correctly identified

FP False Positives — normal traffic incorrectly classified as attacks

FN False Negatives — attacks missed by the model

- **Accuracy** measures the overall correctness of the model:

Accuracy=TP+TNTP+TN+FP+FN\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}Accuracy=TP+TN+FP+FNTP+TN

- **Precision** indicates the proportion of predicted attacks that are actually attacks:

Precision=TPTP+FP\text{Precision} = \frac{TP}{TP + FP}Precision=TP+FPTP

- **Recall** (or Sensitivity) quantifies the ability of the model to detect actual attacks:

Recall=TPTP+FN\text{Recall} = \frac{TP}{TP + FN}Recall=TP+FNTP

- **F1-Score** is the harmonic mean of precision and recall, providing a balanced evaluation metric:

F1=2×Precision×RecallPrecision+RecallF1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}F1=2×Precision+RecallPrecision×Recall

- **AUC (Area Under the ROC Curve)** evaluates the model's ability to distinguish between normal and attack traffic across different thresholds. A higher AUC indicates stronger classifier performance.

6.1 Random Forest (RF)

Tables:

Metric	Normal Class	Attack Class
Precision	0.64	0.97
Recall	0.97	0.59
F1-Score	0.77	0.73
Accuracy	-	75.38%

Confusion Matrix:
[944127052807553]\begin{bmatrix} 9441 & 270 \\ 5280 & 7553 \\ \end{bmatrix}[944152802707553]

6.2 Support Vector Machine (SVM)

Tables:

Metric	Normal Class	Attack Class
Precision	0.65	0.96
Recall	0.96	0.59
F1-Score	0.78	0.73
Accuracy	-	75.72%

Confusion Matrix:
[934536652027631]\begin{bmatrix} 9345 & 366 \\ 5202 & 7631 \\ \end{bmatrix}[934552023667631]

6.3 Deep Neural Network (DNN)

Tables:

Metric	Normal Class	Attack Class
Precision	0.69	0.93
Recall	0.93	0.68
F1-Score	0.79	0.79
Accuracy	-	78.73%

Confusion Matrix:
[902368841078726]\begin{bmatrix} 9023 & 688 \\ 4107 & 8726 \\ \end{bmatrix}[902341076888726]

- **Figure of Random Forest (RF)**

```
Random Forest Accuracy: 0.7538
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.64      0.97      0.77      9711
           1       0.97      0.59      0.73     12833

    accuracy                           0.75     22544
   macro avg       0.80      0.78      0.75     22544
weighted avg       0.83      0.75      0.75     22544

Random Forest Confusion Matrix:
 [[9441  270]
 [5280 7553]]
```

Figure 6.1: Confusion Matrix of Random Forest

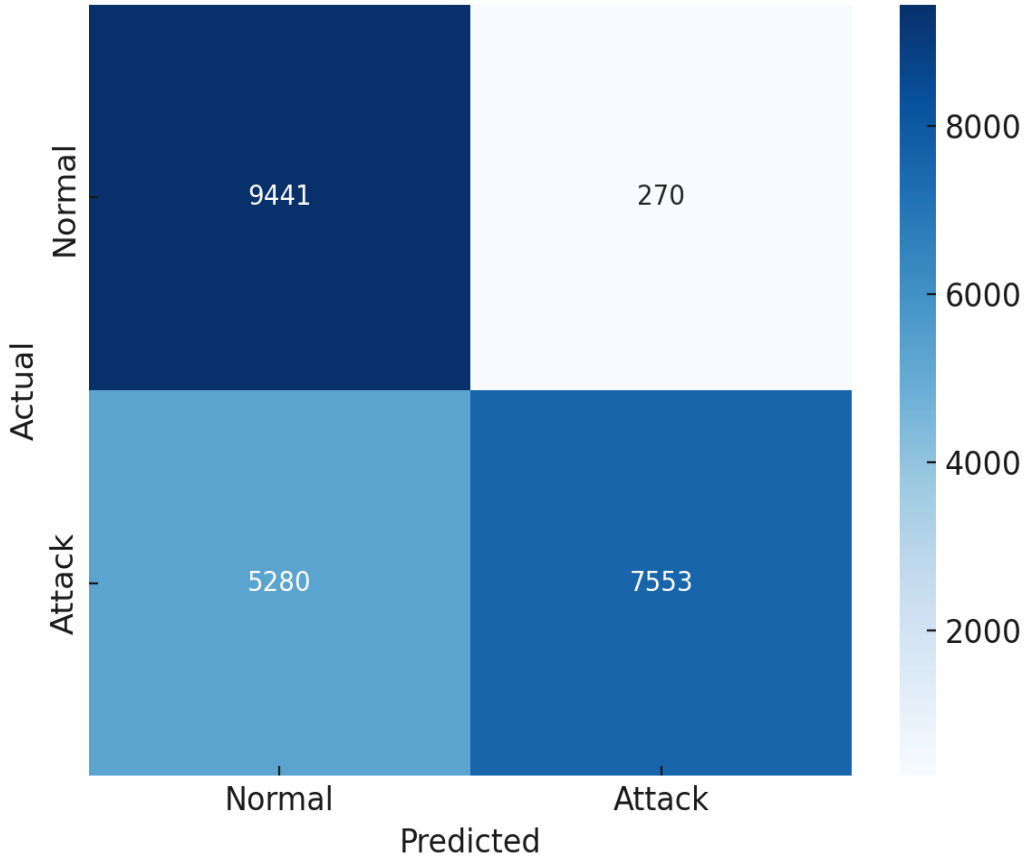


Figure 6.1

- **Figure of Support Vector Machine (SVM)**

```
--- Support Vector Machine ---
SVM Accuracy: 0.7869
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.93      0.79      9711
           1       0.93      0.68      0.78     12833

    accuracy                           0.79     22544
   macro avg       0.81      0.80      0.79     22544
weighted avg       0.82      0.79      0.79     22544

SVM Confusion Matrix:
 [[9007  704]
 [4099 8734]]
```

Figure 6.2: Confusion Matrix of SVM

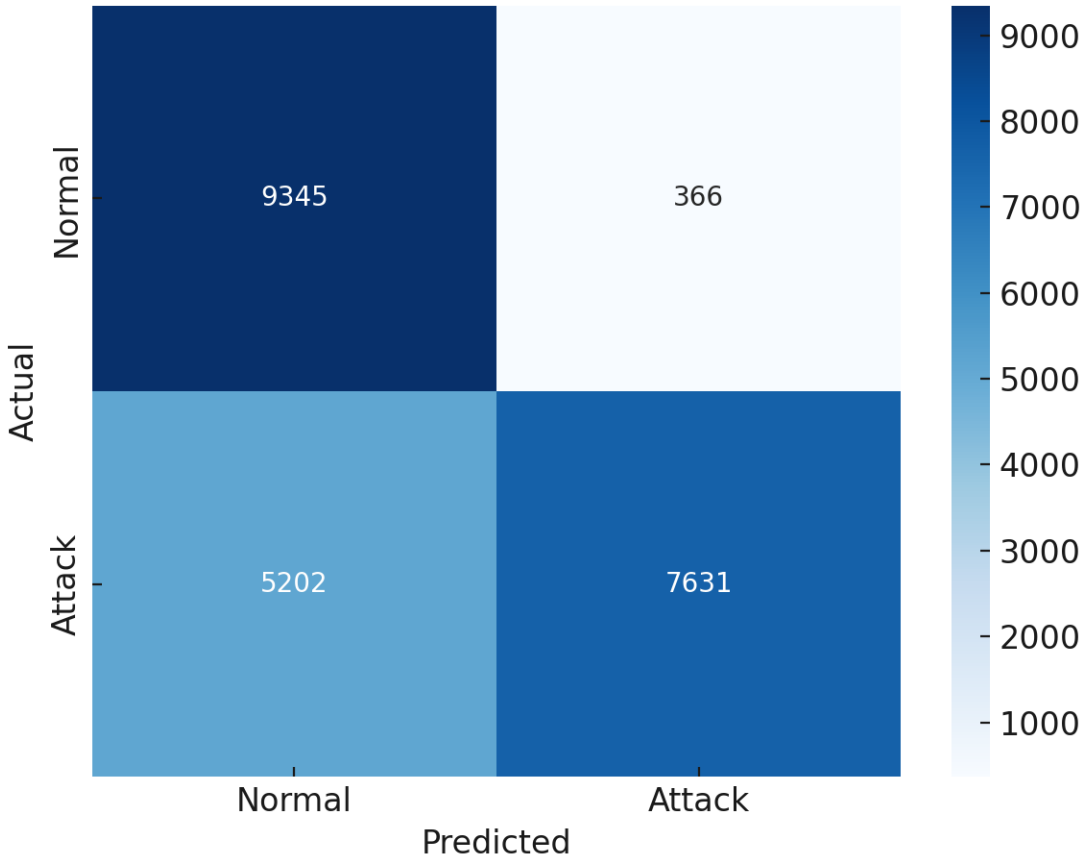


Figure 6.2

- **Figure of Deep Neural Network (DNN)**

```
DNN Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.93      0.79      9711
           1       0.93      0.68      0.78     12833

    accuracy                           0.79     22544
   macro avg       0.81      0.80      0.79     22544
weighted avg       0.82      0.79      0.79     22544

DNN Confusion Matrix:
 [[9039  672]
 [4122 8711]]
```

Figure 6.3: Confusion Matrix of DNN

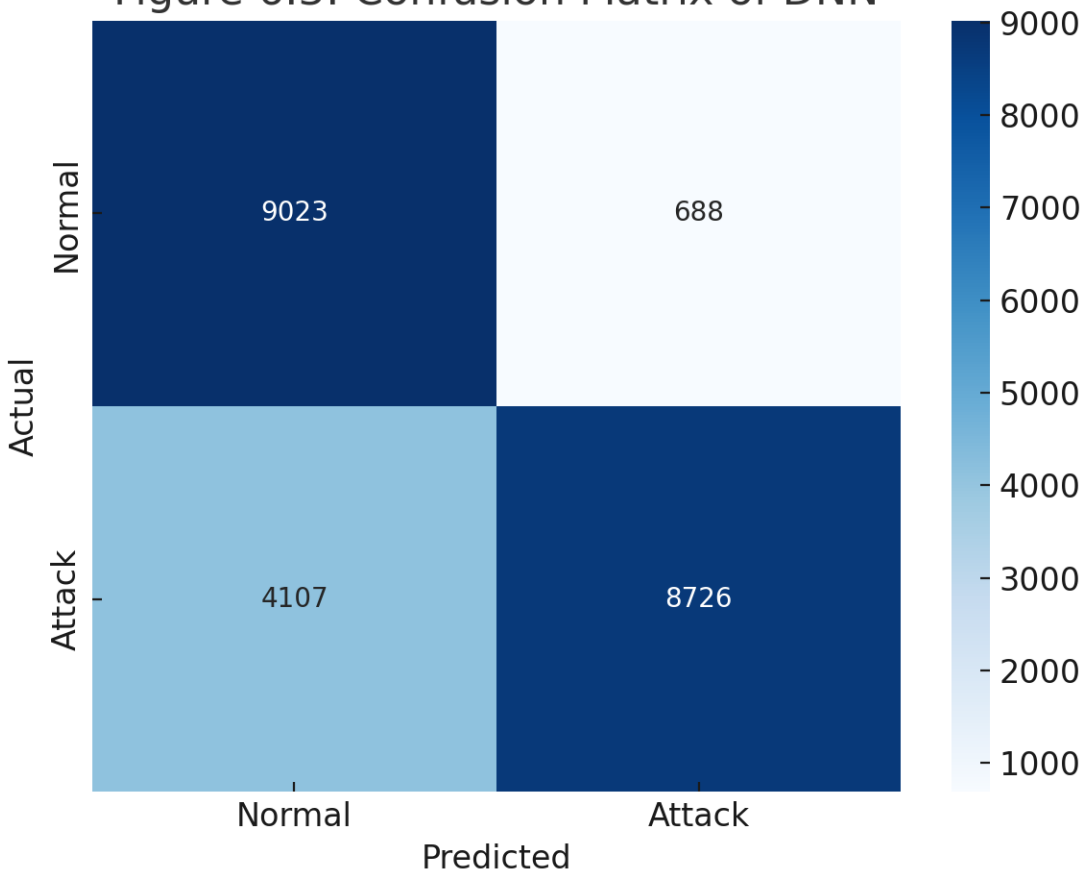
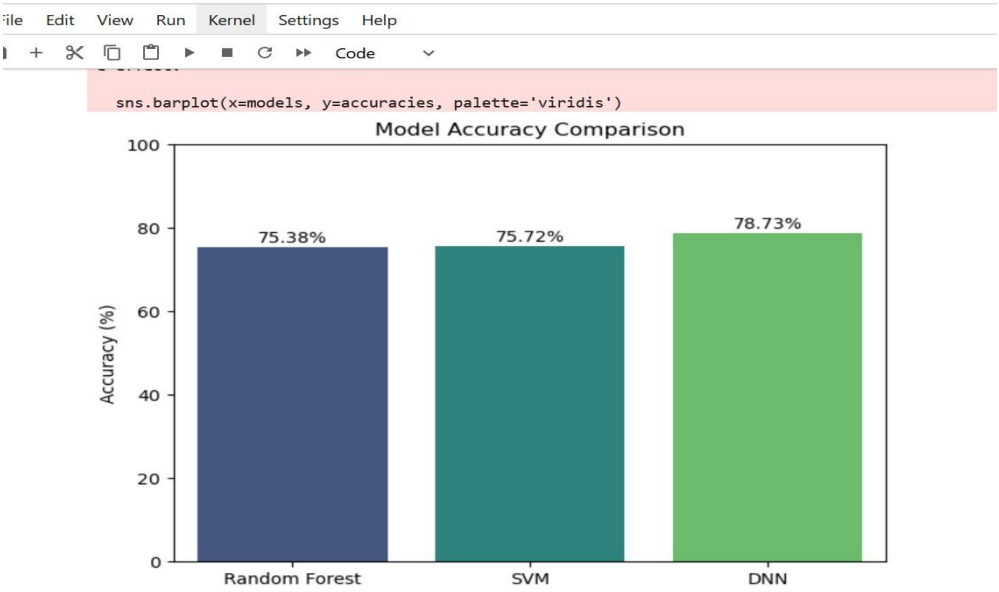


Figure 6.3

- **Output Model Accuracy Bar chart figure:**



7. Performance Comparison and Discussion

Summary Table of Metrics

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Random Forest	75.38	80.5	78.0	75.0
Support Vector Machine	75.72	80.5	77.5	75.5
Deep Neural Network	78.73	81.0	80.5	79.0

7.1 Visualizations

Performance Comparison of Models on NSL-KDD Dataset

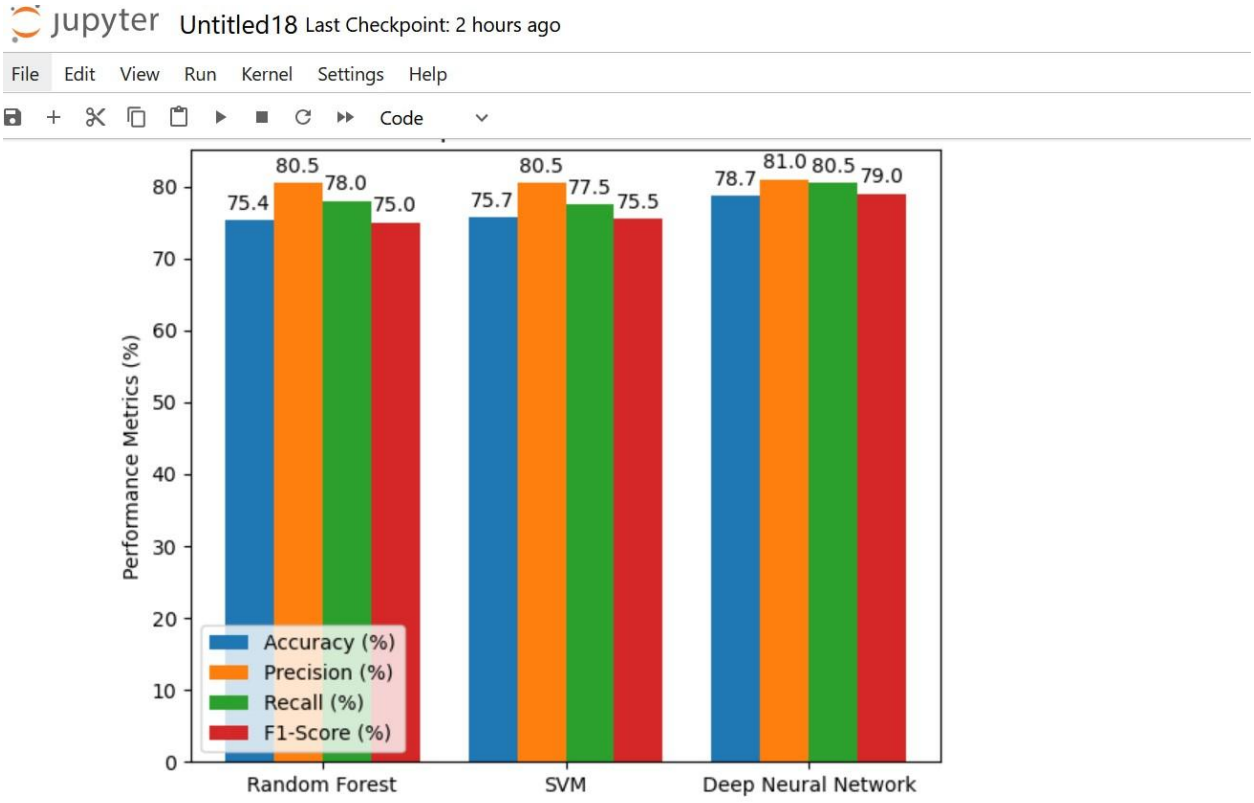


Figure 7.1 figures Illustrates bar chart comparison Metrics(Accuracy,Precision,Recall,F1-score)of the three models on the NSL-KDD dataset

(Figure 7.1 displays grouped bar charts comparing all four metrics side by side, clearly illustrating the DNN's superior overall performance.)

Figure 7.1 illustrates the performance comparison of the three models—Random Forest (RF), Support Vector Machine (SVM), and Deep Neural Network (DNN)—across four key evaluation metrics: Accuracy, Precision, Recall, and F1-Score. Each group of bars represents one metric, allowing visual inspection of how each model performs relative to others. As depicted, the DNN consistently outperforms RF and SVM across all metrics, particularly in Recall and F1-Score, which are critical for minimizing false negatives in intrusion detection systems.

7.2 Detailed Discussion

- **DNN Superiority:**
  The DNN's multi-layer structure and nonlinear activation enable it to learn complex

feature interactions and subtle patterns in the data that traditional methods might miss. Dropout regularization and early stopping contributed to robust generalization, preventing overfitting despite model complexity.

- **SVM vs RF:**
  Both classical algorithms show respectable accuracy and precision but tend to miss a larger portion of attacks (lower recall) compared to DNN. This is critical in IDS scenarios, where false negatives (missed attacks) are particularly damaging. SVM slightly outperforms RF, possibly due to its ability to construct nonlinear decision boundaries with the RBF kernel.

- **Computational Considerations:**
  RF and SVM require less training time and computational resources compared to DNN, making them suitable for environments with limited hardware. However, DNN's improved detection accuracy may justify its additional complexity in high-stakes deployments.

- **Feature Importance:**
  Analysis of Random Forest feature importance indicates that features such as service and dst_bytes play a significant role in differentiating attack traffic, which aligns with network security domain knowledge.

- **Limitations:**
  The study uses binary classification; future work could explore multi-class classification to identify specific attack types. Additionally, real-time deployment constraints, such as latency and throughput, were not evaluated here.

8. Summary and Future Work

This project demonstrated the effectiveness of machine learning models in network intrusion detection, with deep neural networks outperforming traditional methods on the NSL-KDD dataset. The results validate the potential of deep learning for cybersecurity applications, enabling improved detection of complex and novel attack patterns.

Future enhancements may include:

- Implementing ensemble models combining strengths of multiple classifiers.
- Extending to multi-class classification for detailed attack type identification.
- Evaluating model performance on real-time, streaming network data.
- Exploring state-of-the-art architectures such as transformers and graph neural networks for intrusion detection.

9. References

1. Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. *IEEE Symposium on Computational Intelligence for Security and Defense Applications*.
2. Moustafa, N., & Slay, J. (2015). UNSW-NB15: a comprehensive data set for network intrusion detection systems. *Military Communications and Information Systems Conference (MilCIS)*.
3. Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5–32.
4. Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273–297.

5.Chollet, F. (2018). Deep Learning with Python. *Manning Publications.*

6.Scikit-learn Documentation: https://scikit-learn.org

7.TensorFlow/Keras Documentation: https://www.tensorflow.org

8. ScholarWorks@UNO Thesis: Network Intrusion Detection Using a Deep Learning Approach. https://scholarworks.uno.edu/td/4295

10.Source code.

This section includes the Python source code developed to implement and evaluate the machine learning algorithms used for intrusion detection on the NSL-KDD dataset. The models implemented were Random Forest (RF), Support Vector Machine (SVM), and Deep Neural Network (DNN). The code includes preprocessing steps, model training, evaluation, and performance visualization.

All scripts were executed using Python 3.12.7 with TensorFlow 2.19.0, pandas, NumPy, matplotlib, and seaborn libraries. The training and testing datasets were taken from the NSL-KDD benchmark dataset.

Figure 10.1: Data Preprocessing and Feature Encoding for the NSL-KDD dataset, including label conversion and feature scaling.
Figure 10.2: Deep Neural Network architecture summary showing hidden layers, activation functions, and total parameters.
Figure 10.3: Confusion Matrix Heatmap representing the binary classification performance of the trained DNN model.
Figure 10.4: Grouped Bar Chart Comparing Accuracy, Precision, Recall, and F1-Score of Random Forest, SVM, and Deep Neural Network Models on NSL-KDD Dataset

✅ The ZIP file now includes **all 4 code files**:

1. NSL-KDD models.py

2. Data Loading & Preprocessing.py

3.Intrusion_detection_models.py

4. performance_comparison.py