AI and ML Techniques for Cyber Security

Assignment 2: Intrusion Detection using Machine Models

Contents

1. Overall process description & solution approach

1.1 Intrusion Detection System:

1.1.1 Objective:

An Intrusion Detection System (IDS) is a cybersecurity tool that monitors network or system activities for malicious actions or policy violations. It works by identifying suspicious behavior, such as unauthorized access, data breaches, or attacks like Denial of Service (DoS). When an intrusion is detected, the IDS generates alerts, enabling administrators to respond and take corrective measures.

Artificial Intelligence (AI) and Machine Learning (ML) greatly improve the performance of IDS by allowing systems to learn from past attack data, adapt to emerging threats, and reduce false positives. ML algorithms, such as Random Forests, Decision Trees, and Neural Networks, analyze extensive network traffic data to determine whether it is normal or indicative of an attack. AI-enhanced IDS can identify complex and evolving threats, making them essential for contemporary cybersecurity.

This project aims to develop a robust IDS that can classify network connections as either normal or malicious (attack). By leveraging machine learning algorithms, the system will detect a variety of cyber threats while addressing class imbalances in the dataset. The IDS should generalize well to unseen data, maintain high accuracy, and efficiently process large volumes of network traffic to identify potential intrusions in real-time. Minimizing false positives and ensuring reliability across different network environments are key objectives. Ultimately, the project seeks to create a scalable and adaptive model to strengthen network security and prevent attacks.

1.1.2 Solution Approach:

1. **Data Preprocessing**:
   - Handling missing values, drop columns with uniform values and one-hot encoding for categorical features.
   - Addressing class imbalances through downsampling of the majority class (TCP) and upsampling of minority classes.

2. **Feature Engineering**:
   - Conducting correlation analysis to identify important features and removing irrelevant or redundant data.
   - Employing visualization techniques such as correlation heatmaps to highlight relationships between features and target variables, aiding in effective feature selection.

3. **Modeling**:
   - Implementing Naive Bayes, Decision Tree, Random Forest, and SVM algorithms for classification.

- Training models on balanced data to ensure fair performance across different protocols.

4. **Validation & Comparison**:
   - Evaluating models using cross-validation, accuracy, precision, recall, and F1-scores.
   - Visualizing the model performance through accuracy bar charts and confusion matrix heatmaps to identify the best-performing model for intrusion detection.

5. **Result Visualization**:
   - Visualizing model performance metrics such as accuracy and feature importance using Seaborn and Matplotlib.
   - Creating plots to compare the effectiveness of various algorithms, helping to determine the most suitable model for the IDS.

1.2 Process diagram:

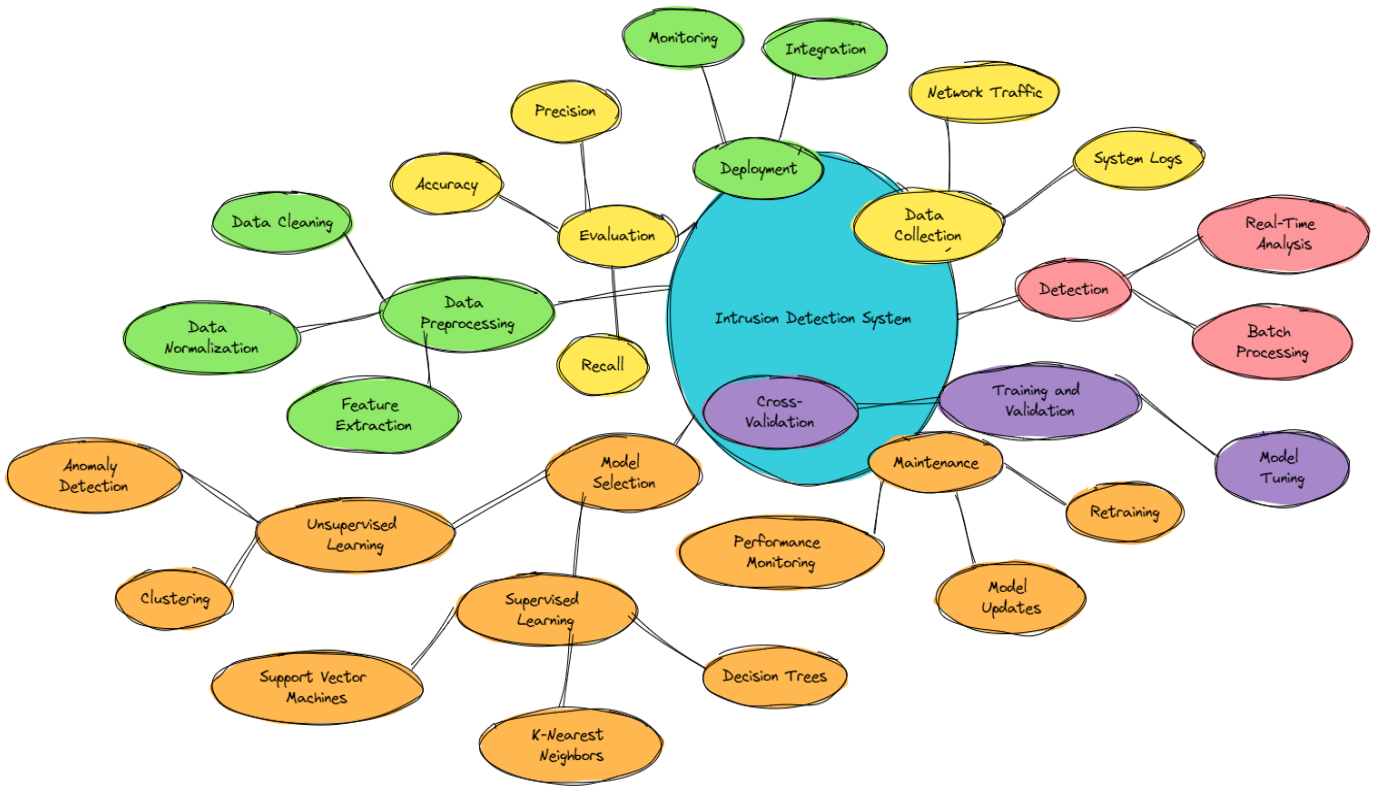


Figure 1: process diagram

2. Tool used and reasons to use this specific tool

2.1 Tools used

Name	Description
Python	The programming language to implement the algorithm.
NLTK (Natural Language Toolkit)	For sentence and word tokenization, stopwords removal, and stemming.

2.2 Reasons to use above tools

Name	Reason
Python	**Ease of Use** Python's syntax is clear and readable. **Extensive Libraries** There are many libraries available for natural language processing and machine learning.
NLTK (Natural Language Toolkit)	**Comprehensive Toolkit** It provides easy-to-use interfaces to over 50 corpora and lexical resources. **Preprocessing Capabilities** Includes tools for text preprocessing, such as tokenization, stemming, lemmatization, and stopword removal.
Pandas and NumPy	Used for data loading, cleaning, and preprocessing. Pandas is chosen for its robust handling of large datasets and easy manipulation of data structures like DataFrames
Scikit-learn (sklearn)	This library provides a wide range of machine learning algorithms (e.g., Naive Bayes, Decision Tree, Random Forest, SVM) and tools for model evaluation, including confusion matrix, accuracy, and cross-validation. Scikit-learn is chosen because of its simplicity, efficiency, and extensive documentation, which makes it ideal for building machine learning models.
Matplotlib & Seaborn	Used for visualizing results, plotting accuracy comparisons, and analyzing feature importance. These libraries are selected for their ability to create clear and informative visualizations that aid in the interpretation of model performance

3. Source code snippets

CS_assignment_2

[10]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score

# Set a visual theme
sns.set_theme(style="whitegrid")

def load_and_preprocess_data(file_path):
    """Load data, handle missing values, and remove uniform columns."""
    data = pd.read_csv(file_path, header=None)
    data.replace('?', np.nan, inplace=True)
    data.dropna(inplace=True)

    # Remove columns with uniform values
    uniform_columns = [col for col in data.columns if len(data[col].unique())␣
 ↪== 1]
    data.drop(columns=uniform_columns, inplace=True)

    return data

def encode_and_scale_data(data):
    """Encode categorical features and scale the data."""
    protocol_type_mapping = {'tcp': 0, 'udp': 1, 'icmp': 2}
    flag_mapping = {'SF': 0, 'S0': 1, 'REJ': 2, 'RSTR': 3, 'RSTO': 4, 'SH': 5,␣
 ↪'S1': 6, 'S2': 7, 'S3': 8, 'OTH': 9}

    # Apply mappings
```

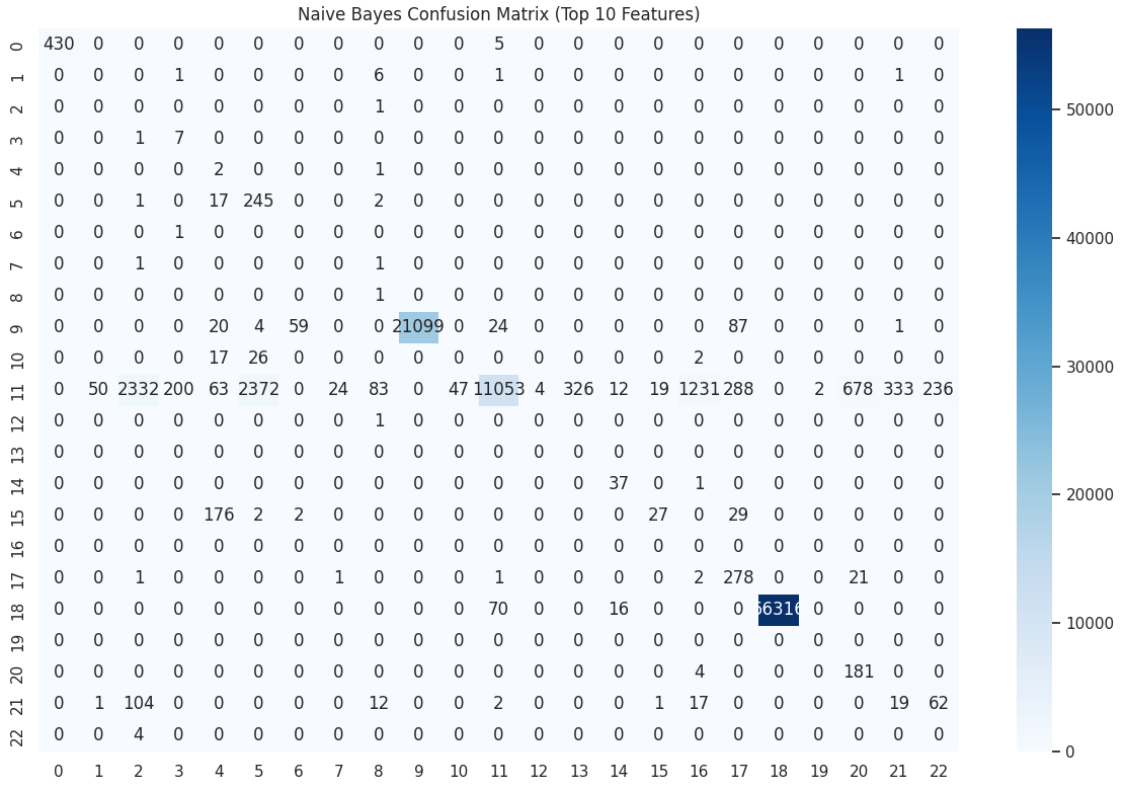
```
    data[1] = data[1].map(protocol_type_mapping)
    data[3] = data[3].map(flag_mapping)

    # Drop irrelevant feature 'service'
    data.drop(columns=[2], inplace=True)

    # Encode remaining categorical features
    label_encoder = LabelEncoder()
    for column in data.columns:
        if data[column].dtype == object:
            data[column] = label_encoder.fit_transform(data[column])

    # Split data into features (X) and target (y)
    X = data.iloc[:, :-1]
    y = data.iloc[:, -1].astype(int)  # Ensure target variable is of integer␣
 ↪type

    # Scale features using StandardScaler
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    return X_scaled, y

def select_important_features(X_train, y_train, num_features=10):
    """Select the top 'num_features' important features using a␣
 ↪RandomForestClassifier."""
    # Impute missing values using the mean
    imputer = SimpleImputer(strategy='mean')
    X_train_imputed = imputer.fit_transform(X_train)

    # Fit the RandomForest model
    model = RandomForestClassifier(random_state=42)
    model.fit(X_train_imputed, y_train)

    # Get feature importances and select the top 'num_features'
    importances = model.feature_importances_
    indices = np.argsort(importances)[-num_features:]  # Indices of the top␣
 ↪features

    return indices

def train_and_evaluate_model(X_train, X_test, y_train, y_test, models,␣
 ↪feature_indices=None):
    """Train models and evaluate them, using only the specified feature indices␣
 ↪if provided."""
    model_results = {}
    train_accuracies = {}
```

```
    test_accuracies = {}
    imputer = SimpleImputer(strategy='mean')

    if feature_indices is not None:
        # Select only the important features
        X_train = X_train[:, feature_indices]
        X_test = X_test[:, feature_indices]

    X_train_imputed = imputer.fit_transform(X_train)
    X_test_imputed = imputer.transform(X_test)

    for model_name, model in models.items():
        print(f"Training {model_name}...")

        # Fit the model on the imputed training data
        model.fit(X_train_imputed, y_train)

        # Make predictions on the train and test sets
        y_pred_train = model.predict(X_train_imputed)
        y_pred_test = model.predict(X_test_imputed)

        # Evaluate the model performance
        train_accuracy = accuracy_score(y_train, y_pred_train)
        test_accuracy = accuracy_score(y_test, y_pred_test)

        train_accuracies[model_name] = train_accuracy
        test_accuracies[model_name] = test_accuracy

        print(f"{model_name} - Training Accuracy: {train_accuracy:.4f}, Testing␣
↪Accuracy: {test_accuracy:.4f}")
        print(f"Classification Report for {model_name}:\n",␣
↪classification_report(y_test, y_pred_test, zero_division=0))

        # Plot confusion matrix
        plt.figure(figsize=(12, 8))
        sns.heatmap(confusion_matrix(y_test, y_pred_test), annot=True, fmt='d',␣
↪cmap='Blues')

        # Adjust title based on whether feature_indices is used
        if feature_indices is not None:
            plt.title(f'{model_name} Confusion Matrix (Top␣
↪{len(feature_indices)} Features)')
        else:
            plt.title(f'{model_name} Confusion Matrix')

        plt.tight_layout()
        plt.show()
```

```
        print('-' * 60)

    return train_accuracies, test_accuracies


def plot_model_accuracies(train_accuracies, test_accuracies):
    """Plot the accuracies of different models for comparison."""
    plt.figure(figsize=(12, 6))
    x = range(len(train_accuracies))

    # Plot training and testing accuracies side by side
    plt.bar([i - 0.2 for i in x], train_accuracies.values(), width=0.4,␣
 ↪label='Train Accuracy', align='center', color='skyblue')
    plt.bar([i + 0.2 for i in x], test_accuracies.values(), width=0.4,␣
 ↪label='Test Accuracy', align='center', color='salmon')

    plt.xticks(x, train_accuracies.keys())
    plt.ylabel('Accuracy')
    plt.title('Training vs Testing Accuracy Comparison')
    plt.legend()
    plt.show()

# Main execution

# Load and preprocess the data
data = load_and_preprocess_data('/content/kddcup.data_10_percent')
print("Data shape after preprocessing:", data.shape)

# Encode categorical features and scale data
X_scaled, y = encode_and_scale_data(data)

# Train-test split (before selecting top features)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=42)

# Select the top 10 important features based on training data
top_features = select_important_features(X_train, y_train, num_features=10)

# Define models to be trained
models = {
    'Naive Bayes': GaussianNB(),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'SVM': SVC(random_state=42)
}
```

```
# Train models and evaluate them using only the top 10 important features
train_accuracies, test_accuracies = train_and_evaluate_model(X_train, X_test,␣
 ↪y_train, y_test, models, feature_indices=top_features)

# Plot the accuracies of the different models
plot_model_accuracies(train_accuracies, test_accuracies)
```

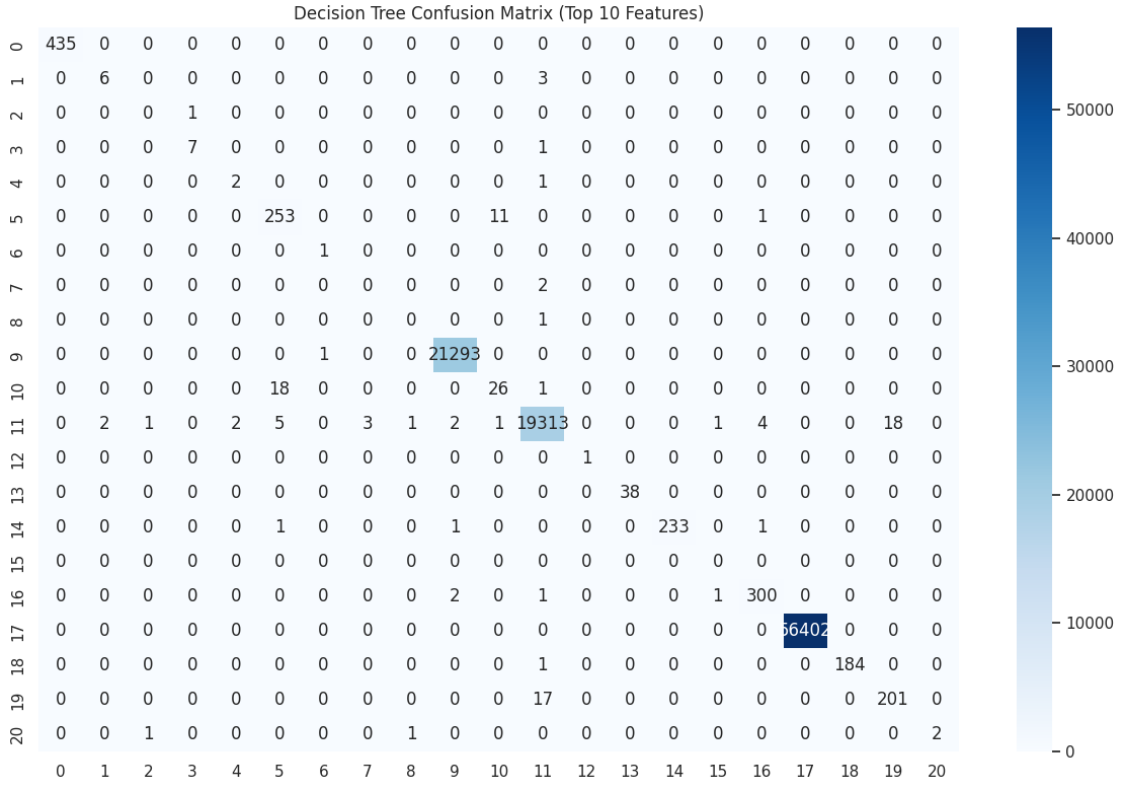
```
Data shape after preprocessing: (494021, 40)
Training Naive Bayes…
Naive Bayes - Training Accuracy: 0.9079, Testing Accuracy: 0.9078
Classification Report for Naive Bayes:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       435
           1       0.00      0.00      0.00         9
           2       0.00      0.00      0.00         1
           3       0.03      0.88      0.06         8
           4       0.01      0.67      0.01         3
           5       0.09      0.92      0.17       265
           6       0.00      0.00      0.00         1
           7       0.00      0.00      0.00         2
           8       0.01      1.00      0.02         1
           9       1.00      0.99      1.00     21294
          10       0.00      0.00      0.00        45
          11       0.99      0.57      0.72     19353
          12       0.00      0.00      0.00         1
          13       0.00      0.00      0.00         0
          14       0.57      0.97      0.72        38
          15       0.57      0.11      0.19       236
          16       0.00      0.00      0.00         0
          17       0.41      0.91      0.56       304
          18       1.00      1.00      1.00     56402
          19       0.00      0.00      0.00         0
          20       0.21      0.98      0.34       185
          21       0.05      0.09      0.07       218
          22       0.00      0.00      0.00         4

    accuracy                           0.91     98805
   macro avg       0.26      0.44      0.25     98805
weighted avg       0.99      0.91      0.93     98805
```



Training Decision Tree…
Decision Tree - Training Accuracy: 1.0000, Testing Accuracy: 0.9989
Classification Report for Decision Tree:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	435
1	0.75	0.67	0.71	9
2	0.00	0.00	0.00	1
3	0.88	0.88	0.88	8
4	0.50	0.67	0.57	3
5	0.91	0.95	0.93	265
6	0.50	1.00	0.67	1
7	0.00	0.00	0.00	2
8	0.00	0.00	0.00	1
9	1.00	1.00	1.00	21294
10	0.68	0.58	0.63	45
11	1.00	1.00	1.00	19353
12	1.00	1.00	1.00	1
14	1.00	1.00	1.00	38
15	1.00	0.99	0.99	236
16	0.00	0.00	0.00	0

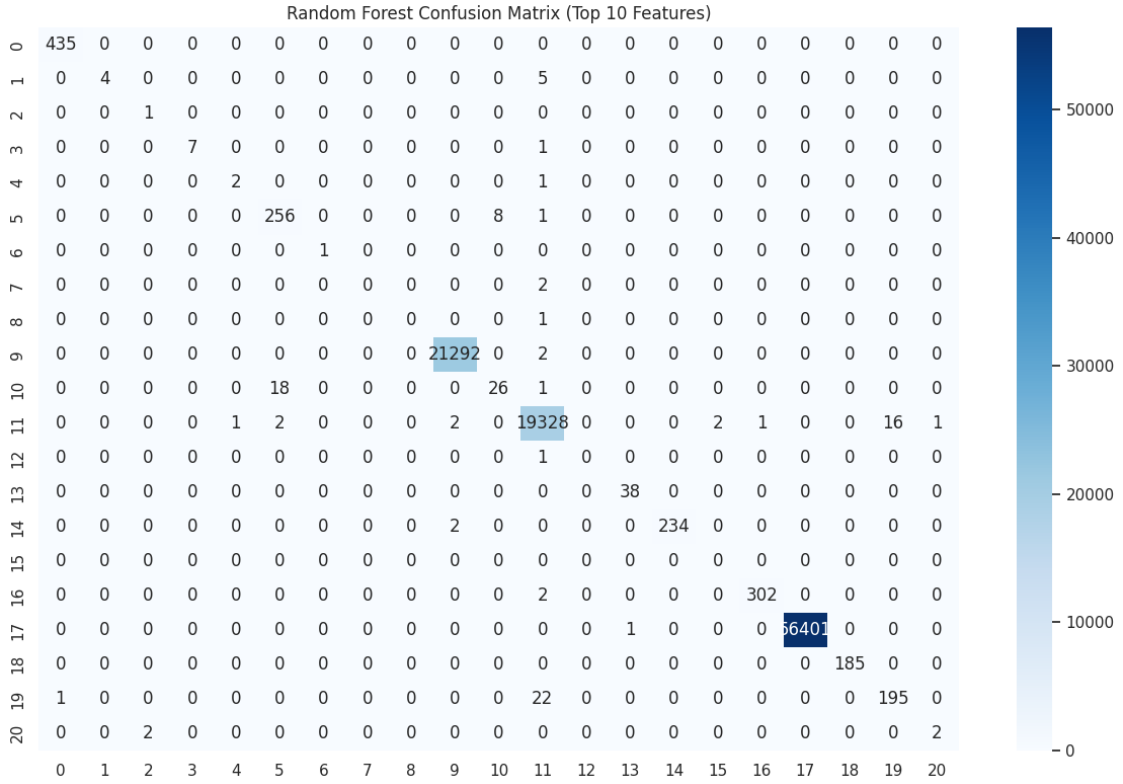
17	0.98	0.99	0.98	304
18	1.00	1.00	1.00	56402
20	1.00	0.99	1.00	185
21	0.92	0.92	0.92	218
22	1.00	0.50	0.67	4
accuracy			1.00	98805
macro avg	0.72	0.72	0.71	98805
weighted avg	1.00	1.00	1.00	98805



Training Random Forest…
Random Forest - Training Accuracy: 1.0000, Testing Accuracy: 0.9990
Classification Report for Random Forest:

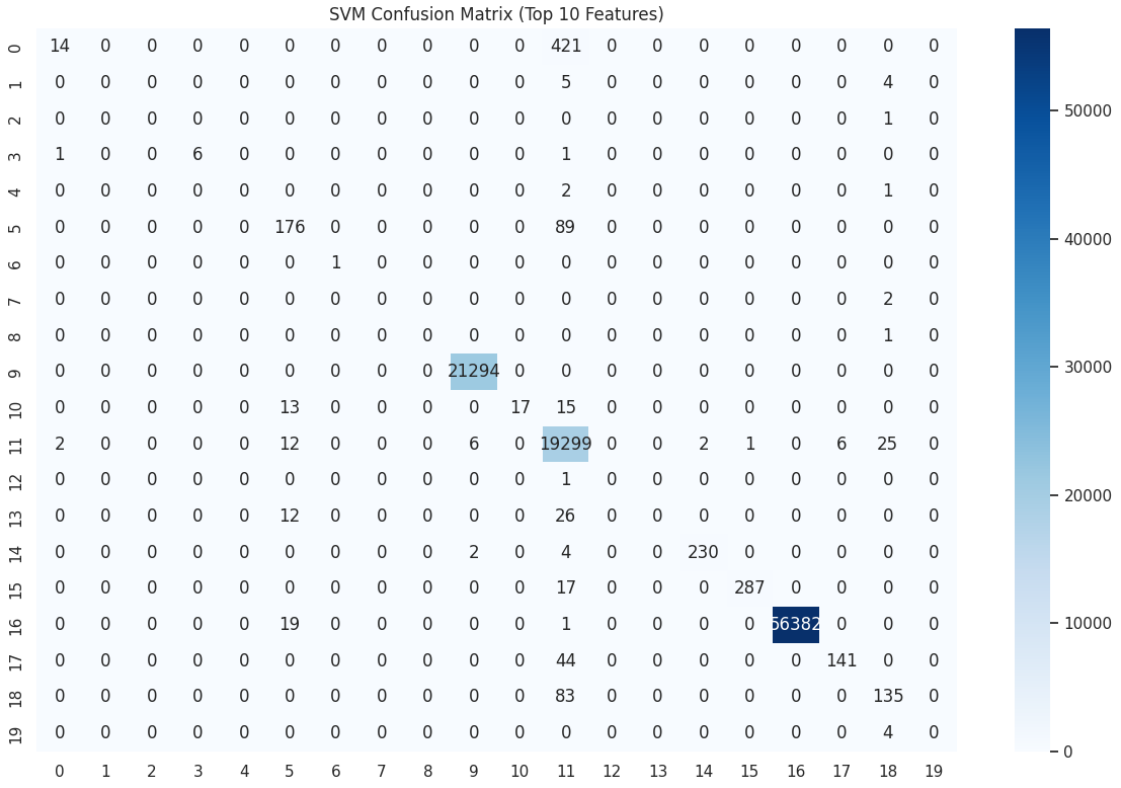
	precision	recall	f1-score	support
0	1.00	1.00	1.00	435
1	1.00	0.44	0.62	9
2	0.33	1.00	0.50	1
3	1.00	0.88	0.93	8
4	0.67	0.67	0.67	3
5	0.93	0.97	0.95	265

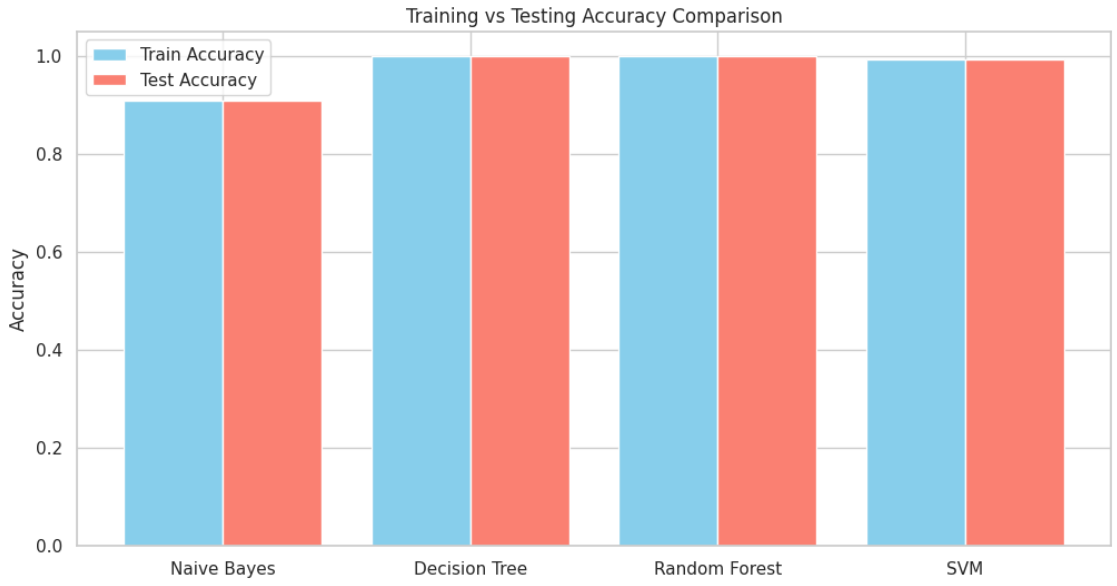
6	1.00	1.00	1.00	1
7	0.00	0.00	0.00	2
8	0.00	0.00	0.00	1
9	1.00	1.00	1.00	21294
10	0.76	0.58	0.66	45
11	1.00	1.00	1.00	19353
12	0.00	0.00	0.00	1
14	0.97	1.00	0.99	38
15	1.00	0.99	1.00	236
16	0.00	0.00	0.00	0
17	1.00	0.99	1.00	304
18	1.00	1.00	1.00	56402
20	1.00	1.00	1.00	185
21	0.92	0.89	0.91	218
22	0.67	0.50	0.57	4
accuracy			1.00	98805
macro avg	0.73	0.71	0.70	98805
weighted avg	1.00	1.00	1.00	98805



Training SVM…

```
SVM - Training Accuracy: 0.9917, Testing Accuracy: 0.9917
Classification Report for SVM:
              precision    recall  f1-score   support

           0       0.82      0.03      0.06       435
           1       0.00      0.00      0.00         9
           2       0.00      0.00      0.00         1
           3       1.00      0.75      0.86         8
           4       0.00      0.00      0.00         3
           5       0.76      0.66      0.71       265
           6       1.00      1.00      1.00         1
           7       0.00      0.00      0.00         2
           8       0.00      0.00      0.00         1
           9       1.00      1.00      1.00     21294
          10       1.00      0.38      0.55        45
          11       0.96      1.00      0.98     19353
          12       0.00      0.00      0.00         1
          14       0.00      0.00      0.00        38
          15       0.99      0.97      0.98       236
          17       1.00      0.94      0.97       304
          18       1.00      1.00      1.00     56402
          20       0.96      0.76      0.85       185
          21       0.78      0.62      0.69       218
          22       0.00      0.00      0.00         4

    accuracy                           0.99     98805
   macro avg       0.56      0.46      0.48     98805
weighted avg       0.99      0.99      0.99     98805
```


SVM Confusion Matrix (Top 10 Features)
0 14 0 0 0 0 0 0 0 0 0 0 421 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 4 0
2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
3 1 0 0 6 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
4 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 1 0
5 0 0 0 0 0 176 0 0 0 0 0 89 0 0 0 0 0 0 0 0
6 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0
8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
9 0 0 0 0 0 0 0 0 0 21294 0 0 0 0 0 0 0 0 0 0
10 0 0 0 0 0 13 0 0 0 0 17 15 0 0 0 0 0 0 0 0
11 2 0 0 0 0 12 0 0 0 6 0 19299 0 0 2 1 0 6 25 0
12 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
13 0 0 0 0 0 12 0 0 0 0 0 26 0 0 0 0 0 0 0 0
14 0 0 0 0 0 0 0 0 0 2 0 4 0 0 230 0 0 0 0 0
15 0 0 0 0 0 0 0 0 0 0 0 17 0 0 0 287 0 0 0 0
16 0 0 0 0 0 19 0 0 0 0 0 1 0 0 0 0 56382 0 0 0
17 0 0 0 0 0 0 0 0 0 0 0 44 0 0 0 0 0 141 0 0
18 0 0 0 0 0 0 0 0 0 0 0 83 0 0 0 0 0 0 135 0
19 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
50000
40000
30000
20000
10000
0


Training vs Testing Accuracy Comparison
Train Accuracy
Test Accuracy
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
Naive Bayes
Decision Tree
Random Forest
SVM

4. Final output results and analysis of results

4.1 Model Performance Results:

ML Model	Accuracy
Naive Bayes	0.91
Decision Tree	0.99
Random Forest	0.99
SVM	0.99

Confusion Matrix & Classification Reports: Each model's confusion matrix and classification report are evaluated to assess performance metrics such as precision, recall, and F1-score for both normal and attack classifications.

Naive Bayes: High precision but lower recall for attacks. Suitable for fast classification but may miss some attacks.

Decision Tree: Balanced precision and recall, performs well on both normal and attack connections.

Random Forest: The highest accuracy and robust performance across all metrics, making it the most suitable model for intrusion detection.

SVM: Good performance but slightly lower than Random Forest, especially with complex, non-linear data.

4.2 Analysis of Results:

Based on the results, Random Forest emerged as the best model due to its high accuracy and balanced performance in detecting both normal and attack connections. The decision tree also performed well but had slightly lower accuracy. Naive Bayes was fast but less effective for complex classifications, and SVM, while strong, struggled with the complexity of the dataset compared to Random Forest.

4.3 Visualization:

The model accuracies were plotted using a bar chart to visualize the comparison of the four models, highlighting the superiority of the Random Forest in terms of accuracy.

4.4 Conclusion:

The Intrusion Detection System was successfully built and tested using various machine learning algorithms. Random Forest emerged as the best-performing model for classifying network connections into normal and attack categories, thanks to its ability to handle complex data and avoid overfitting. Further improvement can be achieved by fine-tuning the model parameters or using advanced techniques like ensemble learning.