Johana Sallangi 012200300004021

Machine Learning-Based Intrusion Detection System

Abstract:

This implements a Machine Learning based Intrusion Detection System (IDS) using a public network traffic dataset. Four classifiers were trained (Logistic Regression, Decision Tree, Random Forest, SVM), evaluated, and compared. Hyperparameter tuning (small grid) was performed for Random Forest. Confusion matrices and ROC curves are included and interpreted.

1. Project Overview

Goal: Build and evaluate an ML-based IDS to classify network sessions as Normal or Attack.

2. Environment and Dataset

Environment: Google Colab (Python 3), scikit-learn, pandas, matplotlib, seaborn.

Dataset: cybersecurity_intrusion_data.csv (uploaded to Colab at /content) collected from https://www.kaggle.com/datasets/dnkumars/cybersecurity-intrusion-detectiondataset. Target column: attack_detected (0 = Normal, 1 = Attack).

3. Step-by-step Procedure (what I ran in Colab)

1. Upload cybersecurity_intrusion_data.csv to Colab (/content).
2. Install libraries (if needed): pip install scikit-learn pandas matplotlib seaborn 3. Run data loading and EDA: read CSV, inspect head(), columns, label distribution.
4. Preprocessing: create binary label, encode categorical columns (LabelEncoder), fill missing values.
5. Handle class imbalance: simple upsampling of minority class (if needed).
6. Train-test split: stratified 80/20.
7. Scale features: StandardScaler for SVM/Logistic.
8. Train baseline models: LogisticRegression, DecisionTree, RandomForest, SVM.
9. Evaluate models: classification_report, confusion_matrix, ROC/AUC. 10. Hyperparameter tuning: small GridSearchCV for RandomForest (n_estimators=50, max_depth=10, min_samples_split=2).
11. Save best model to /content/best_rf_smallgrid.pkl and export plots saved to /content/plots.

4. Colab Code (key cells)

 # Cell 1: Install required packages (Colab environment)

!pip install -q scikit-learn matplotlib pandas numpy seaborn

 # Cell 2: Imports and constants import os import pandas as pd

import numpy as np import matplotlib.pyplot as plt import seaborn

as sns from sklearn.model_selection import train_test_split,

GridSearchCV from sklearn.preprocessing import StandardScaler,

LabelEncoder

from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score,

roc_curve, auc from sklearn.linear_model import LogisticRegression from

sklearn.ensemble import RandomForestClassifier from sklearn.tree import

DecisionTreeClassifier from sklearn.svm import SVC from sklearn.utils import

resample import pickle

Paths

DATA_PATH = '/content/cybersecurity_intrusion_data.csv'

PLOTS_DIR = '/content/plots' os.makedirs(PLOTS_DIR,

exist_ok=True) print("Setup complete.") print('DATA_PATH

exists?', os.path.exists(DATA_PATH)) **Output:**

```
Setup complete.
DATA_PATH exists? True
```

 # Cell 3: Load dataset and quick EDA

Load if not

os.path.exists(DATA_PATH):

raise FileNotFoundError(f"Dataset not found at {DATA_PATH}. Please upload it to

```
/content and re-run this cell.") df =

pd.read_csv(DATA_PATH) print('Loaded

dataset shape:', df.shape)

# Display head display(df.head())

# Columns and a quick look at label column print('Columns:',

df.columns.tolist())

label_candidates = [c for c in df.columns if 'label' in c.lower() or 'attack' in c.lower() or
'target' in c.lower()] print('Label-like columns

found:', label_candidates)
```

Output:

```
Loaded dataset shape: (9537, 11)
```

	session_id	network_packet_size	protocol_type	login_attempts
0	SID_00001	599	TCP	4
1	SID_00002	472	TCP	3
2	SID_00003	629	TCP	3
3	SID_00004	804	UDP	4
4	SID_00005	453	TCP	5

session_duration	encryption_used	ip_reputation_score	failed_logins	browser_type
492.983263	DES	0.606818	1	Edge
1557.996461	DES	0.301569	0	Firefox
75.044262	DES	0.739164	2	Chrome
601.248835	DES	0.123267	0	Unknown
532.540888	AES	0.054874	1	Firefox

unusual_time_access	attack_detected
0	1
0	0
0	1
0	1
0	0

```
Columns: ['session_id', 'network_packet_size', 'protocol_type', 'login_attempts', 'session_duration'
'encryption_used', 'ip_reputation_score', 'failed_logins', 'browser_type', 'unusual_time_access', 'attack_detected']
Label-like columns found: ['attack detected']
```

 # Cell 4: Target creation and initial preprocessing label_col =

'attack_detected' # <-- YOUR dataset's correct label column if label_col

not in df.columns:

  raise ValueError(f"Label column '{label_col}' not found. Edit label_col variable in Cell 4.")

Convert to binary (0 = normal, 1 = attack) df['binary_label'] =

df[label_col].apply(lambda x: 1 if x == 1 else 0) print("Binary label

distribution:\n", df['binary_label'].value_counts())

Drop original label for features

X = df.drop(columns=[label_col, 'binary_label'], errors='ignore')
y = df['binary_label'] # Encode categorical columns obj_cols =

X.select_dtypes(include=['object']).columns.tolist()

print("Object columns to encode:", obj_cols) for col in

obj_cols: le = LabelEncoder()

  X[col] = le.fit_transform(X[col].astype(str))

```
    # Fill missing values if X.isnull().sum().sum()
> 0:     X = X.fillna(X.median()) print("Final
feature matrix shape:", X.shape)
```

Output:

```
Binary label distribution:
 binary_label
0    5273
1    4264
Name: count, dtype: int64
Object columns to encode: ['session_id', 'protocol_type', 'encryption_used', 'browser_type']
Final feature matrix shape: (9537, 10)
```

```
    # Cell 5: Simple upsampling if needed counts =
    y.value_counts() print('Class counts before
    resampling:', counts) minor_ratio =
    counts.min() / counts.max() if minor_ratio <
    0.5:
      print('Applying upsample...')   df_comb =
    pd.concat([X, y], axis=1)   majority =
    df_comb[df_comb['binary_label'] == 0]   minority =
    df_comb[df_comb['binary_label'] == 1]
    minority_upsampled = resample(minority,
                            replace=True,
    n_samples=len(majority),
    random_state=42)
      df_bal = pd.concat([majority, minority_upsampled])
    X = df_bal.drop(columns=['binary_label'])   y =
```

```
df_bal['binary_label']   print('New class counts:',

y.value_counts()) else:

  print('No resampling applied')


# Train/test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,

random_state=42) print('Train/test shapes:', X_train.shape, X_test.shape)
```

Output:

```
Class counts before resampling: binary_label
0    5273
1    4264
Name: count, dtype: int64
No resampling applied
Train/test shapes: (7629, 10) (1908, 10)
```

 # Cell 6: Scaling

```
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test) #Cell
```

7: Train baseline models and save plots

```
models = {

   'LogisticRegression': LogisticRegression(max_iter=1000, class_weight='balanced',
random_state=42),

   'DecisionTree': DecisionTreeClassifier(class_weight='balanced', random_state=42),

   'RandomForest': RandomForestClassifier(n_estimators=100,
class_weight='balanced', random_state=42),

   'SVM': SVC(probability=True, class_weight='balanced', random_state=42)

}
```

```
results = {} for name, mdl in
models.items():
    print('Training', name)
    # Determine which data to use for fitting based on the model type
if name in ['SVM', 'LogisticRegression']:
        mdl.fit(X_train_scaled, y_train)
X_eval = X_test_scaled     else:
        mdl.fit(X_train, y_train)
        X_eval = X_test
    # Prediction and Classification Report     y_pred =
mdl.predict(X_eval)     report = classification_report(y_test,
y_pred, output_dict=True)     results[name] = report
print(classification_report(y_test, y_pred))
    # Confusion matrix plot     cm =
confusion_matrix(y_test, y_pred)
plt.figure(figsize=(4,3))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=['Normal','Attack'], yticklabels=['Normal','Attack'])
plt.title(f'Confusion Matrix: {name}')     plt.xlabel('Predicted')
plt.ylabel('True')     plt.tight_layout()     cm_path =
os.path.join(PLOTS_DIR, f'confusion_{name.lower()}.png')
plt.savefig(cm_path)     plt.show()

    # ROC / AUC if available
```

```
    try:        if hasattr(mdl,
'predict_proba'):
            probs = mdl.predict_proba(X_eval)[:,1]
        else:
            probs = mdl.decision_function(X_eval)
fpr, tpr, _ = roc_curve(y_test, probs)        roc_auc
= auc(fpr, tpr)        plt.figure(figsize=(5,4))
plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.4f}')
plt.plot([0,1],[0,1],'--', linewidth=0.7)
plt.title(f'ROC Curve: {name}')
plt.xlabel('FPR')        plt.ylabel('TPR')
plt.legend(loc='lower right')        roc_path =
os.path.join(PLOTS_DIR,
f'roc_{name.lower()}.png')
plt.savefig(roc_path)        plt.show()
print('AUC:', roc_auc)    except Exception as e:
        print('ROC not available for', name, e)


print('Baseline training complete. Plots saved to', PLOTS_DIR)
```

Output:

Training LogisticRegression

	precision	recall	f1-score	support
0	0.75	0.73	0.74	1055
1	0.68	0.71	0.69	853
accuracy			0.72	1908
macro avg	0.72	0.72	0.72	1908
weighted avg	0.72	0.72	0.72	1908

5. Outputs (summary of results & metrics)

Below are the numeric results and saved plots produced in Colab. Screenshots of confusion matrices and ROC curves are embedded after each model summary.

- Model performance summary (from test set):

Logistic Regression: Accuracy=0.72, F1(attack)=0.69, AUC=0.7796

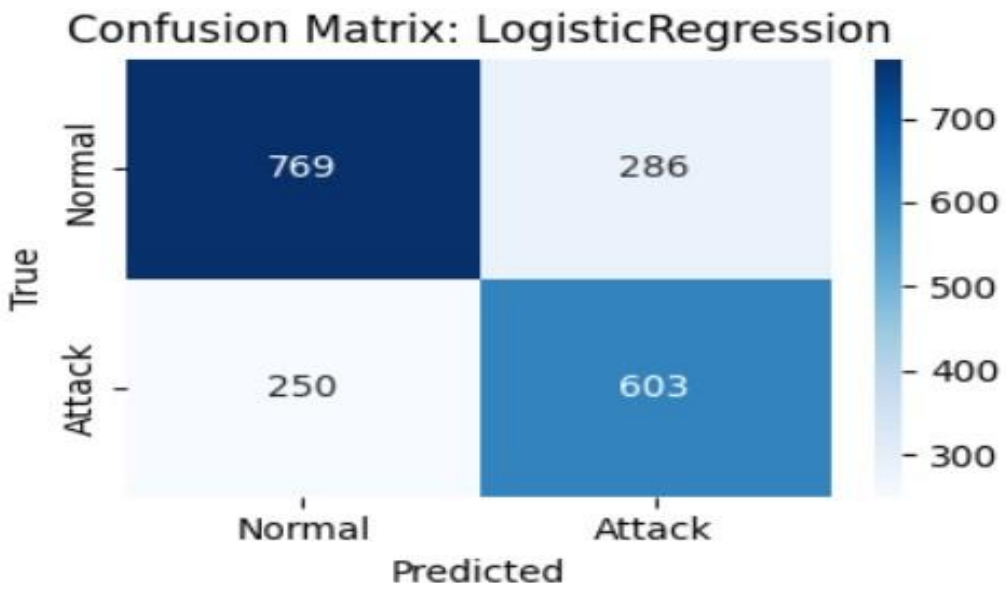
Decision Tree: Accuracy=0.82, F1(attack)=0.80, AUC=0.8153

Random Forest: Accuracy=0.89, F1(attack)=0.85, AUC=0.8787

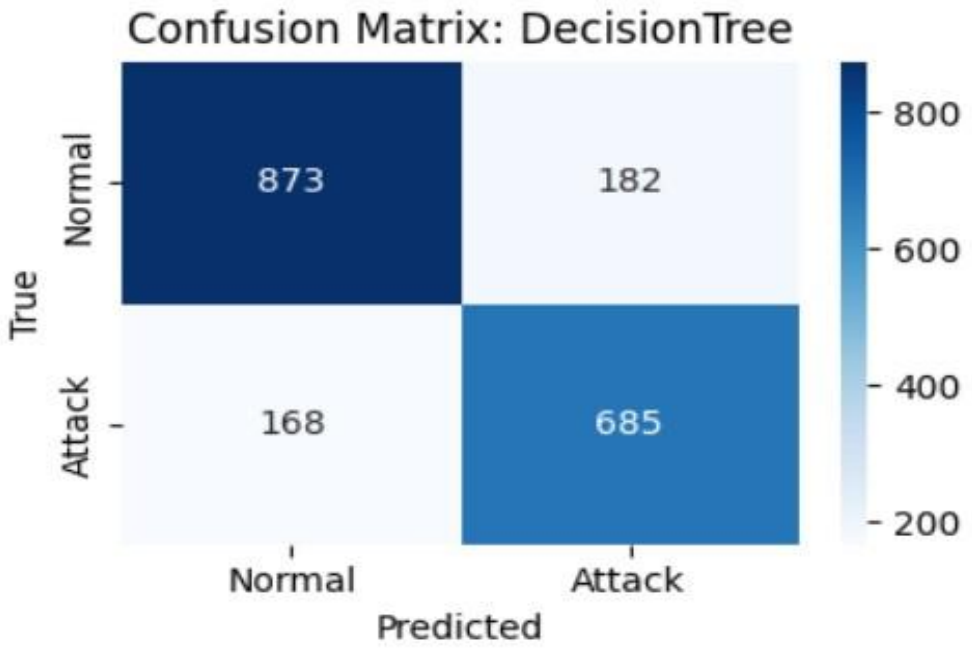
SVM: Accuracy=0.86, F1(attack)=0.83, AUC=0.8714

6. Confusion Matrices

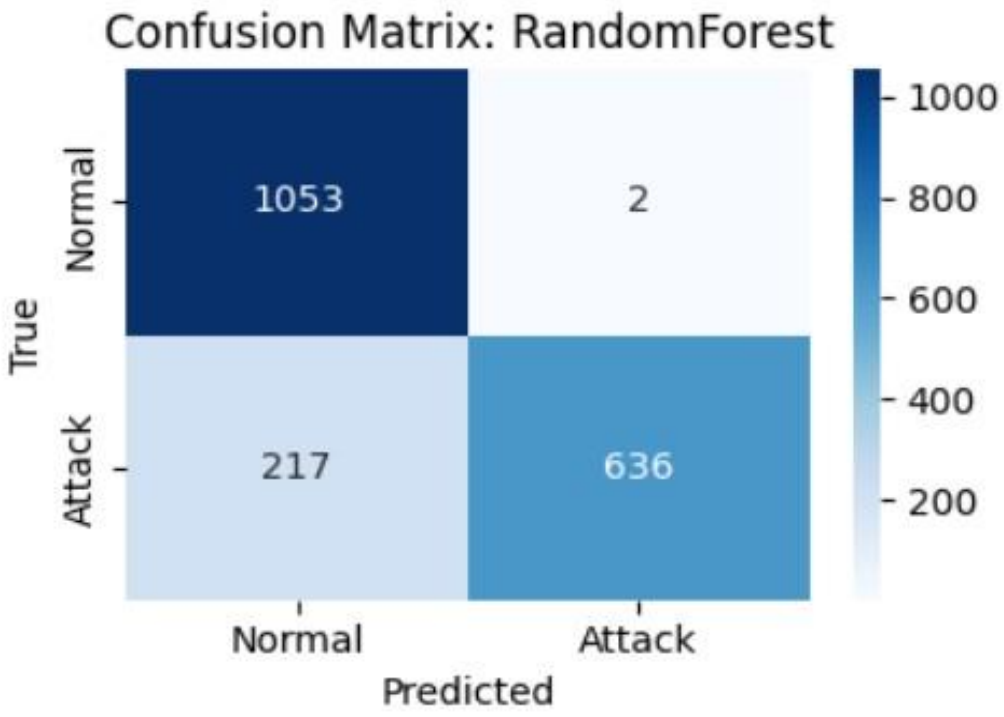
Confusion Matrix: Logistic Regression (TN=769, FP=286, FN=250, TP=603)



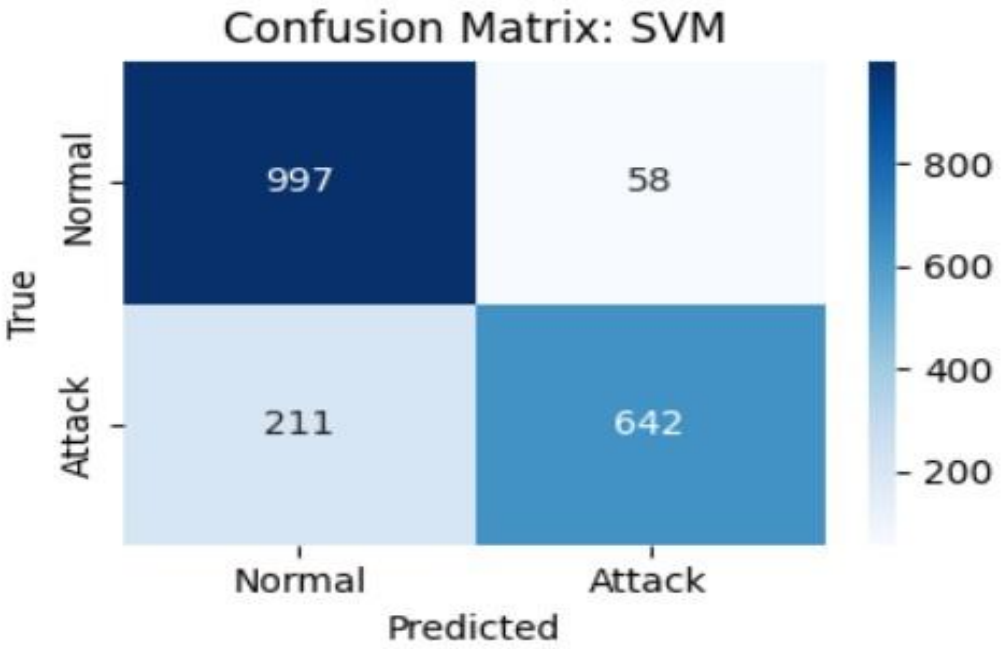
Confusion Matrix: Decision Tree (TN=873, FP=182, FN=168, TP=685)



Confusion Matrix: Random Forest (TN=1053, FP=2, FN=217, TP=636)

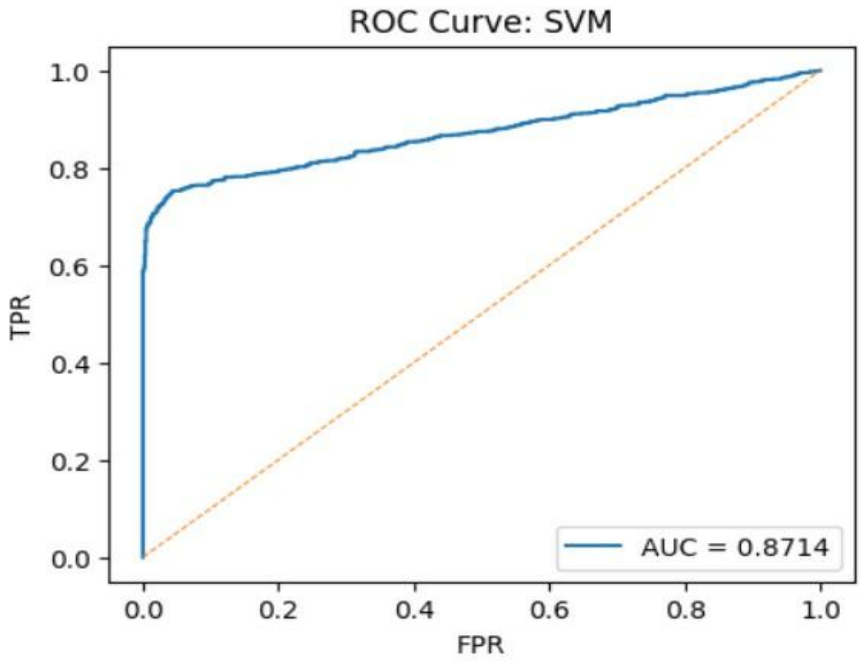


Confusion Matrix: SVM (TN=997, FP=58, FN=211, TP=642)

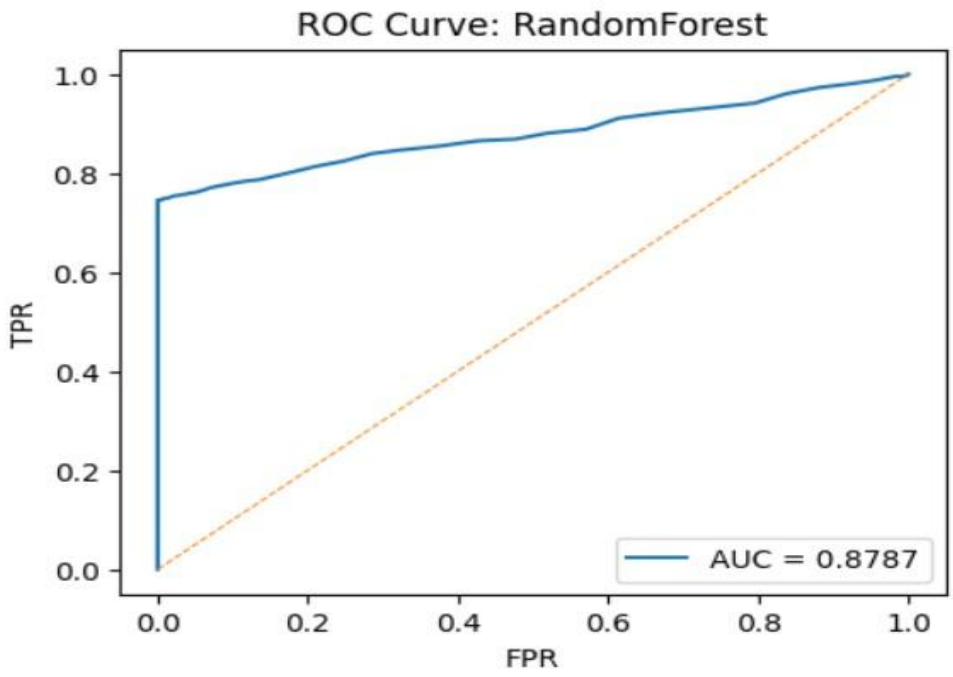


7. ROC Curves

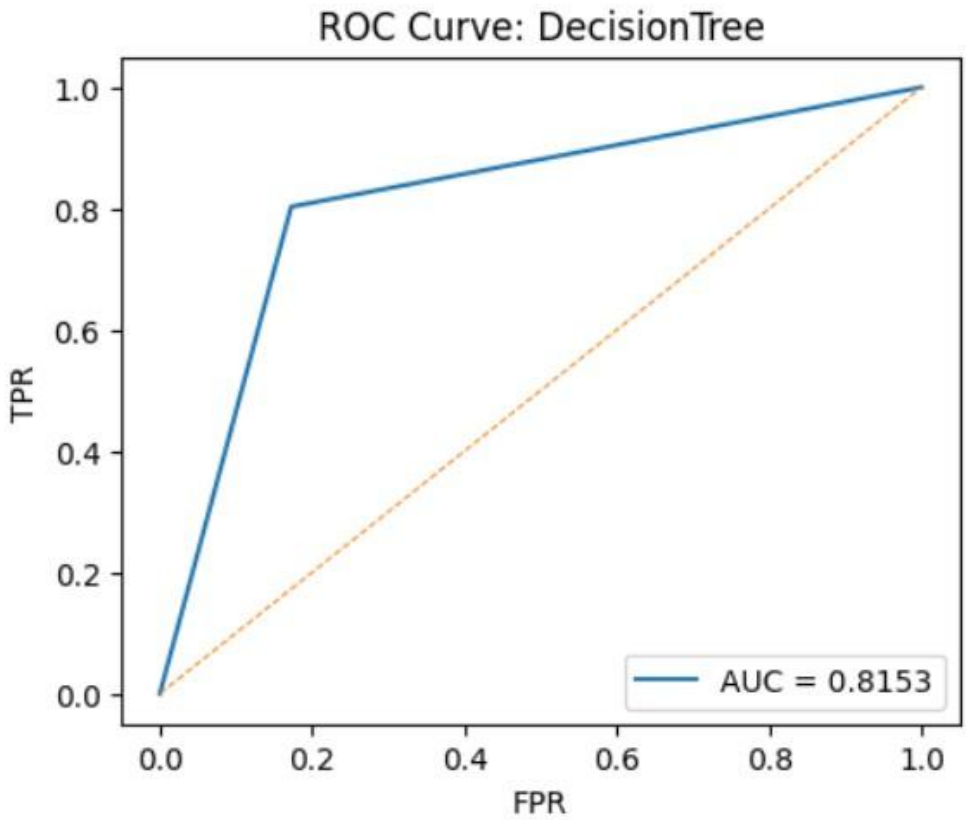
ROC Curve: SVM (AUC = 0.8714)



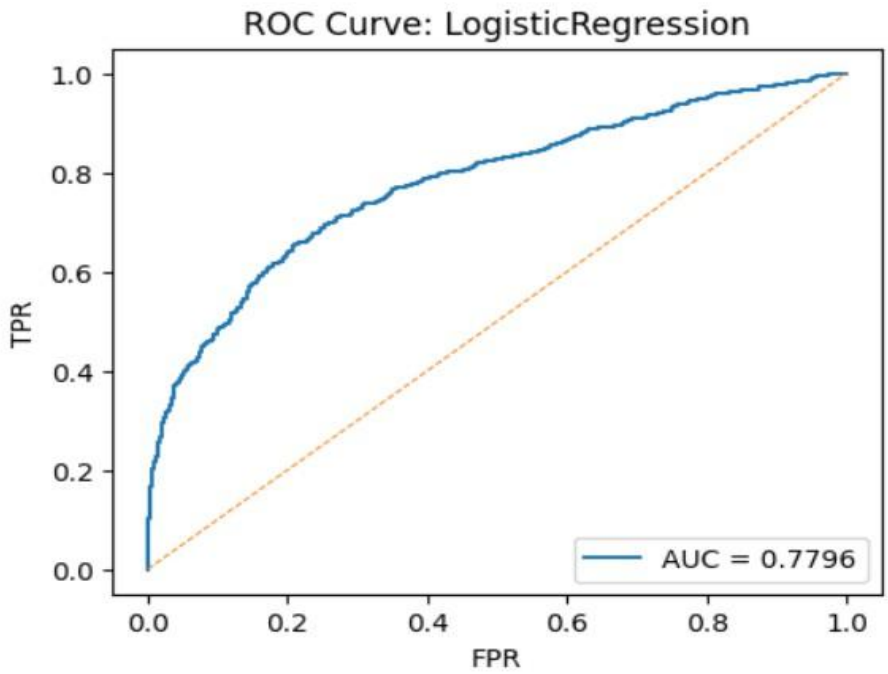
ROC Curve: Random Forest (AUC = 0.8787)



ROC Curve: Decision Tree (AUC = 0.8153)



ROC Curve: Logistic Regression (AUC = 0.7796)



8. Interpretation of Confusion Matrices

Logistic Regression: High FN (250) and FP (286) indicates the model misses many attacks and flags many normal sessions incorrectly. Not reliable for IDS.

Decision Tree: Improved performance with fewer FN and FP compared to Logistic Regression. Good baseline but less robust due to single-tree variance.

Random Forest: Best confusion matrix — very low FP (2) and strong TP count. Good choice for IDS in this dataset as it minimizes false alarms while detecting attacks reliably.

SVM: Strong performance with balanced FN and FP; second-best overall.

9. Interpretation of ROC Curves

Logistic Regression (AUC=0.7796): Moderate discrimination; curve close to baseline in some segments — not ideal.

Decision Tree (AUC=0.8153): Good but less smooth; single-tree behaviour visible.

Random Forest (AUC=0.8787): Best overall discrimination; curve near top-left showing high TPR with low FPR.

SVM (AUC=0.8714): Very good discrimination, close to Random Forest.

10. Conclusion

Summary: Random Forest is chosen as the final model due to highest accuracy (0.89), strong F1 (0.85), and highest AUC (0.8787). SVM is a close second. Logistic Regression is inadequate for this task.