AN INTERNSHIP PROJECT REPORT

AI-BASED INTRUSION DETECTION

submitted by

Preema S - 35

to

Naresh I Technology

NareshIT,

Ameerpet, Hyderabad

May 2025

ACKNOWLEDGEMENT

It gives me great pleasure to present my Internship Project report on “AI-BASED INTRUSION DETECTION”. If words are considered as a symbol of approval and token of knowledge, then these words play heralding role in expressing my gratitude.

First and foremost, I would like express my whole hearted thanks to the invisible and indomitable God for showering his blessings upon me, enabling to complete this project on time. No work, however big or small has ever been done without the contribution of others. So, these words of acknowledgement come as a small gesture of gratitude towards all those people, without whom the successful completion of this project would not have been possible. I would like to express my deep gratitude towards **Mr. Prakash Senapathi, NareshIT** and the entire **NareshIT Management**. I would like to convey my heartfelt and sincere thanks to **Nithyanandam**, Mentor, who gave me their valuable suggestions, motivation, and the direction to proceed at every stage. Last but never the least, I am thankful to my parents and friends for their constant moral, material, and mental support directly and indirectly in my hard times.

ABSTRACT

This project presents a machine learning–based Network Intrusion Detection System (NIDS) utilizing the **XGBoost (Extreme Gradient Boosting)** algorithm to effectively identify malicious network traffic. The system is developed using a publicly available intrusion detection dataset, which undergoes extensive preprocessing, including outlier clipping, feature scaling with StandardScaler, and class balancing using the Synthetic Minority Over-sampling Technique (SMOTE).

The XGBoost model is trained on this processed dataset to classify network connections as either benign or malicious. The model's performance is evaluated using metrics such as accuracy, ROC–AUC, confusion matrix, and SHAP values to ensure both effectiveness and interpretability. The final model and preprocessing components are serialized using joblib, making the system suitable for deployment in real-time security environments, with the capability to detect both known and novel intrusion attempts.

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

1.1 General Background

In today's increasingly connected digital environment, ensuring the security of network systems has become a critical challenge. Traditional Intrusion Detection Systems (IDS), which rely on static rules and known attack signatures, often fail to detect novel or sophisticated threats. As cyberattacks grow more complex, there is a pressing need for intelligent, adaptive detection methods. Machine learning offers a powerful alternative by enabling systems to learn patterns from historical network traffic data and identify anomalies that may indicate intrusions. This project focuses on developing an AI-based IDS using the XGBoost algorithm, supported by data preprocessing techniques such as outlier clipping, feature scaling, and class balancing with SMOTE. The goal is to create a robust and accurate model capable of detecting a wide range of intrusion types in real-time scenarios

1.2 Objectives

This project aims to develop an AI-based Intrusion Detection System (IDS) that can intelligently identify malicious network activities by analysing structured network traffic data. The primary goals of the project are as follows:

1. To preprocess and clean the intrusion dataset
    - Apply outlier clipping to reduce the effect of extreme values.
    - Normalize numerical features using StandardScaler to ensure uniform feature scaling.
2. To address data imbalance using SMOTE
    - Use Synthetic Minority Oversampling Technique (SMOTE) to balance the dataset and improve the model's ability to detect rare intrusion cases.
3. To implement a supervised machine learning pipeline
    - Train an XGBoost classifier using the resampled and scaled dataset for binary classification of normal vs. intrusive traffic.
4. To evaluate the trained model's effectiveness
    - Use performance metrics such as accuracy, ROC-AUC, confusion matrix, and SHAP (SHapley Additive exPlanations) to understand and explain model predictions.
5. To visualize key insights and patterns
    - Generate and save boxplots, feature importance charts, and the ROC curve to understand data distribution and model behavior.

6. To serialize the model and preprocessing pipeline for reuse
    - Save the trained model and the scaler as .pkl files for deployment or future inference tasks.
7. To ensure reproducibility and tracking using MLflow
    - Track experiments, model parameters, metrics, and artifacts using MLflow, supporting future model tuning or comparison.

1.3 Benefits

- The integration of artificial intelligence into intrusion detection systems offers several key benefits. Unlike traditional signature-based methods, AI-based models can learn from data, adapt to new attack patterns, and detect both known and unknown threats with high accuracy.

- These systems also reduce false positives, automate threat detection, and scale effectively to monitor large volumes of network traffic. Furthermore, with tools like SHAP and feature importance analysis, they offer transparency and interpretability, helping security teams understand and trust the model's decisions.

- Overall, AI-driven intrusion detection represents a more intelligent, responsive, and efficient approach to safeguarding digital infrastructure.

2. Modules

2.1 Modules

- Data Preprocessing
- Exploratory Data Analysis (EDA)
- Model Definition and Training
- Evaluation Module
- Visualization & Reporting Module
- Model Serialization

2.1.1 Data Preprocessing

- Handles loading and cleaning of the dataset.
- Load data using pandas.
- Remove or clip outliers (1st and 99th percentile).
- Handle class imbalance using SMOTE.
- Scale features using StandardScaler.

2.1.2 Exploratory Data Analysis (EDA)

- Provides visual insights into the data distribution.
- Boxplots before and after outlier clipping using seaborn.
- Visual inspection of feature distributions and correlations (optional: heatmaps, pair plots).

2.1.3 Model Definition and Training

- Defines and trains the machine learning model.
- Files Involved: src/model.py (contains get_model()).
- Load model configuration (e.g., XGBClassifier, GridSearchCV if used).
- Train the model on training data.
- Save trained model using joblib.

2.1.4 Evaluation Module

- Evaluates model performance and provides visualizations.
- Files Involved: utils/evaluate.py (contains evaluate_model() and shap_explainer()).
- Calculate metrics: Accuracy, ROC-AUC.
- Plot confusion matrix.
- Plot ROC curve.
- Generate SHAP value explanations.
- Plot feature importance (XGBoost only).

2.1.5 Visualization & Reporting Module

- Generates images and plots for model analysis.
- Outputs:
    - Before_Outlier_Clipping.jpg
    - After_Outlier_Clipping.jpg
    - XGBoost_Feature_Importance.jpg
    - Confusion_Matrix.jpg
    - ROC_Curve.jpg.

2.1.6 Model Serialization

- Saves the trained components for deployment or reuse.
- Save trained model (xgboost_intrusion_model.pkl)
- Save scaler (Standard_Scaler.pkl)

3. DOMAIN SPECIFICATION

3.1 Cybersecurity

This project specializes in the domain of Cybersecurity, focusing on Network Intrusion Detection Systems (NIDS). It applies supervised machine learning—specifically the XGBoost algorithm—to classify network traffic as either benign or malicious. Traditional rule-based or signature-based IDS approaches often fail to detect new or evolving threats, so this project uses data-driven methods to overcome those limitations. The model is trained on preprocessed data with outlier clipping, feature scaling, and SMOTE to address class imbalance. Evaluation is performed using metrics such as accuracy, ROC-AUC, confusion matrix, and SHAP values to ensure performance and interpretability. By integrating machine learning into intrusion detection, the system becomes more adaptive, scalable, and capable of identifying complex cyber threats, making it suitable for use in modern network security environments.

3.2 Machine Learning

In this project, machine learning (ML) plays a central role in building an intelligent and adaptive Intrusion Detection System (IDS) capable of identifying malicious network behavior. Unlike traditional rule-based systems, ML enables the model to learn patterns from historical data and make accurate predictions on unseen traffic, including previously unknown threats.

3.2.1 Algorithm Used: XGBoost

The core algorithm used in this project is XGBoost (Extreme Gradient Boosting)—a powerful and efficient ensemble learning method based on decision trees. XGBoost is well-suited for structured/tabular data and offers:

- High performance and scalability
- Robust handling of imbalanced datasets
- Built-in support for feature importance
- Regularization to prevent overfitting

3.2.2 Machine Learning Workflow

1. Data Collection
2. Preprocessing
3. Feature Engineering
4. Model Training
5. Model Evaluation
6. Model Serialization

3.2.3 Evaluation Metrics Used

To evaluate the performance of the model, the following metrics are used:

- Accuracy: Measures overall correctness.
- ROC-AUC (Receiver Operating Characteristic – Area Under Curve): Captures the trade-off between true positive rate and false positive rate.
- Confusion Matrix: Visualizes classification outcomes.
- SHAP Values: Provides model explainability by showing the impact of each feature

3.2.4 Benefits of Using Machine Learning

- Adaptability: Learns from new data, enabling detection of evolving threats.
- Automation: Reduces human effort in maintaining static rules.
- Interpretability: With SHAP and feature importance, analysts can understand why decisions were made.
- Scalability: Can be scaled and deployed in real-time detection environments.

4. SYSTEM ARCHITECTURE

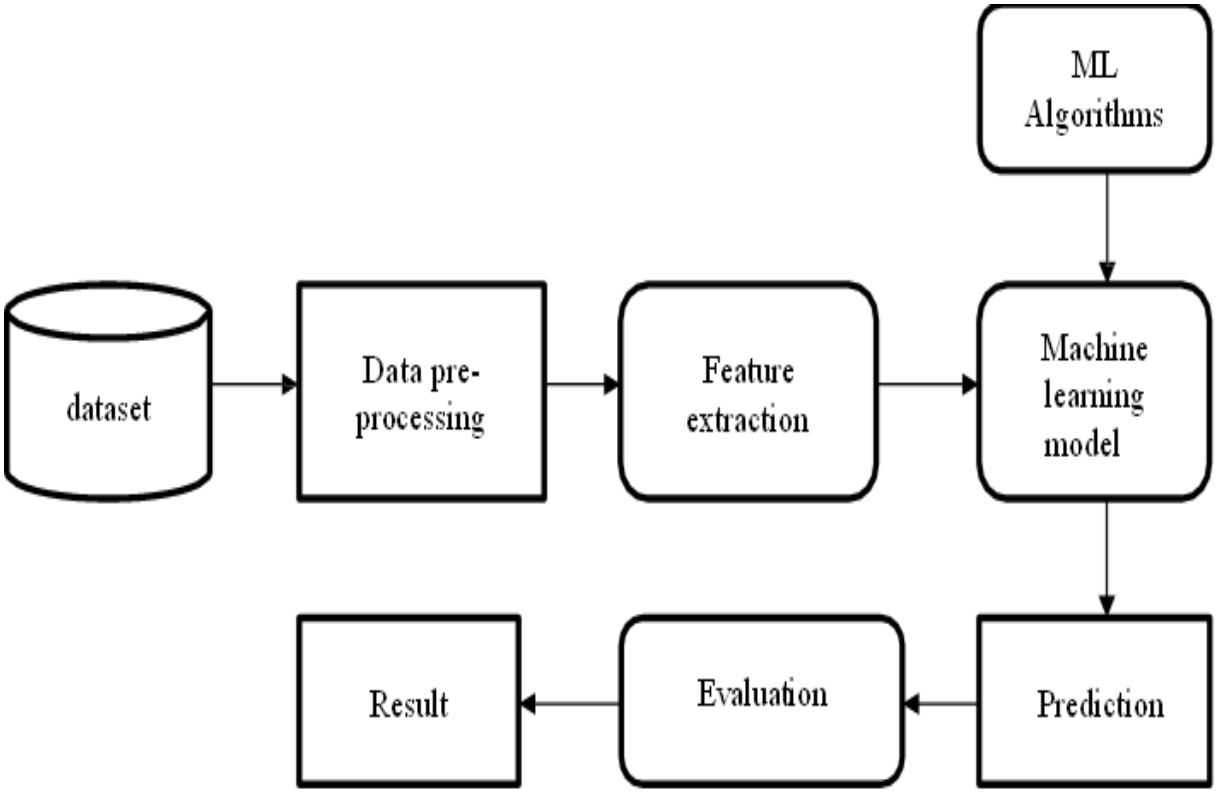


Fig 4.1 System Architecture

5. REQUIREMENTS

5.1 System Requirements

- Minimum Requirements:
  - OS: Windows 10 / Ubuntu 20.04+
  - RAM: 8 GB
  - Processor: Intel i5 / AMD Ryzen 5 or above
  - Disk Space: 5 GB free
  - Python Version: 3.8 or higher

- Software:
  - Visual Studio Code (VS Code)
  - Jupyter Extension for VS Code
  - Anaconda (optional but helpful for managing environments)

5.2 Python Libraries

- pandas
- scikit-learn
- xgboost
- matplotlib
- seaborn
- joblib

6. SOURCE CODE

```
import os
import joblib
import pandas as pd
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from src.model import get_model
import matplotlib.pyplot as plt
from utils.evaluate import evaluate_model,shap_explainer
from xgboost import plot_importance
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
import shap
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
```

Fig 6.1 Importing packages

```
# Load data
data = pd.read_csv(r'C:\Users\Preema S\Desktop\AI-Intrusion-Detection\dataset\intrusion_detection_dataset.csv')
print(data.head())
# Optional: Sample numeric columns for visualization
numeric_cols = data.select_dtypes(include='number').columns
```

```
   feature_1  feature_2  feature_3  feature_4  feature_5  feature_6  \
0   1.242872  -2.846242   1.631471   0.616130   1.024352   3.776219
1   4.910051  -0.803401  -1.785864  -1.706847  -0.070520  -2.213141
2   0.995002   4.472860   0.396552  -0.774943  -0.539313   1.983609
3  -6.064907  -0.861145  -0.663774   0.639216   1.399097   0.464887
4  -3.663457  -2.540896  -0.362081  -1.018162   1.939464  -1.736997

   feature_7  feature_8  feature_9  feature_10  ...  feature_12  feature_13  \
0   2.207521  -4.202171   0.464731   -3.293098  ...   -2.365571    1.826469
1  -0.034315   9.526943   3.740554   -6.521879  ...    0.685556   -5.243079
2   1.571684  -1.178277   1.175303    1.666403  ...    2.167063   -1.271422
3  -7.810023  -0.833776   1.906510    1.299077  ...    0.287488    0.340744
4  -2.255150   6.621085  -3.086259   -5.767685  ...    3.516086    2.774681

   feature_14  feature_15  feature_16  feature_17  feature_18  feature_19  \
0    0.942230   -0.011058   -3.312649   -3.227888    0.703651    4.432210
1   -0.352180    1.901790   -0.846546    0.195492    5.229226    8.772250
2   -0.150584   -2.553335    2.556613    0.109192   -2.650392   -0.891219
3    0.145996    0.116981    3.652516    0.400101   -1.136077   -4.633471
4    0.623417    0.288686    3.795548   -0.541960   -2.213839   -8.462932

   feature_20  intrusion
0   -0.429676          1
1   -1.564854          0
2    0.936507          1
3    0.620830          1
4    0.789908          1

[5 rows x 21 columns]
```

Fig 6.2 Loading and Displaying data

```
# Boxplot before clipping
plt.figure(figsize=(12, 6))
sns.boxplot(data=data[numeric_cols])
plt.title("Before Outlier Clipping")
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('outputs/Before_Outlier_Clipping.jpg')
plt.show()
```

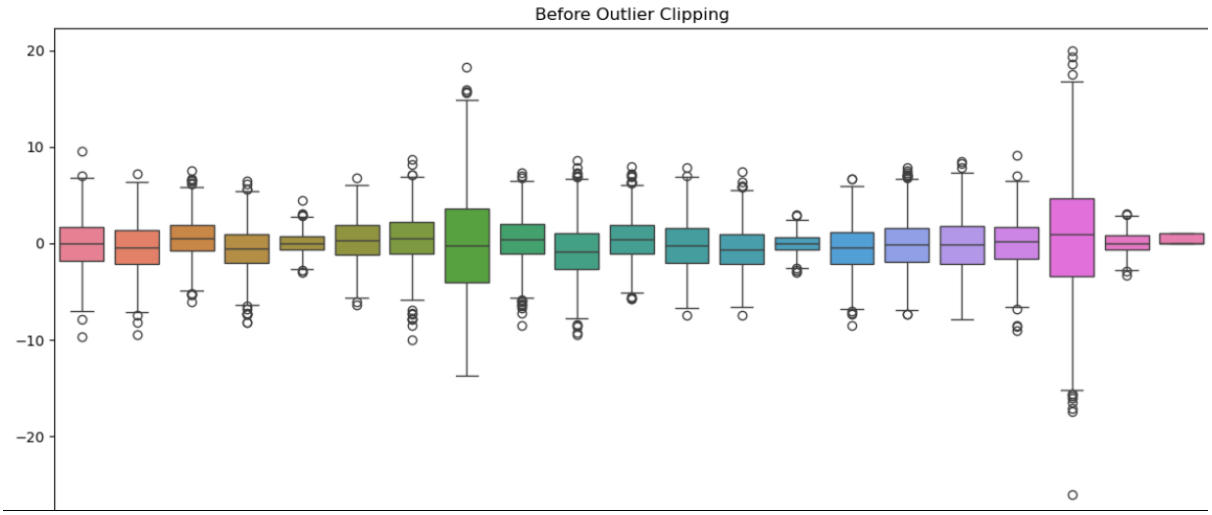


Fig 6.3 Before Clipping (Boxplot)

```
# Perform clipping (as before)
for col in data.select_dtypes(include='number').columns:
    lower = data[col].quantile(0.01)
    upper = data[col].quantile(0.99)
    data[col] = data[col].clip(lower, upper)
```

```
# Boxplot after clipping
plt.figure(figsize=(12, 6))
sns.boxplot(data=data[numeric_cols])
plt.title("After Outlier Clipping")
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('outputs/After_Outlier_Clipping.jpg')
plt.show()
```

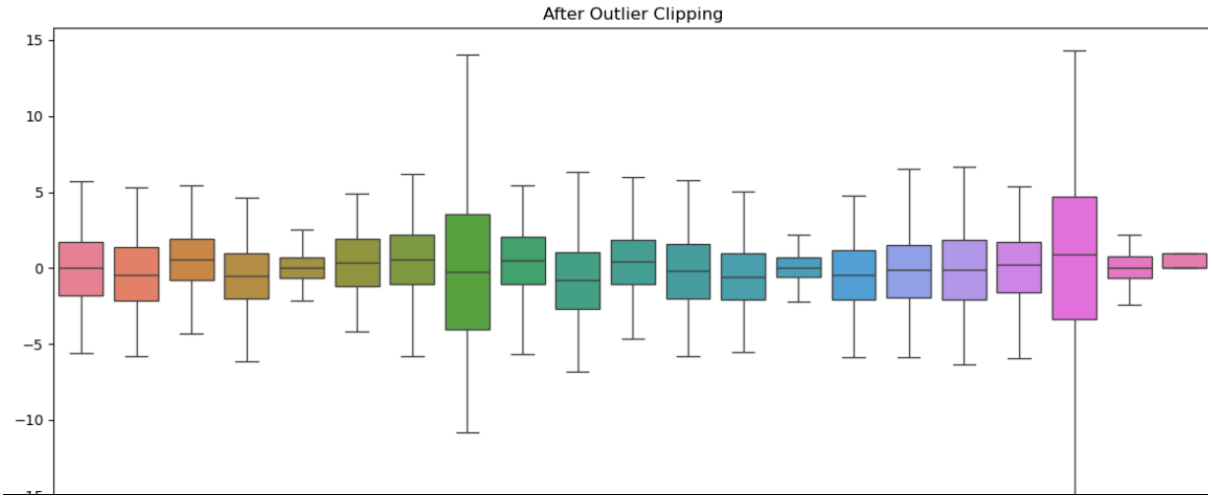


Fig 6.4 Clipping

```
# Assume last column is label
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
```

```
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
X_resampled
```

	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	feature_10
0	1.242872	-2.846242	1.631471	0.616130	1.024352	3.776219	2.207521	-4.202171	0.464731	-3.293098
1	4.910051	-0.803401	-1.785864	-1.706847	-0.070520	-2.213141	-0.034315	9.526943	3.740554	-6.521879
2	0.995002	4.472860	0.396552	-0.774943	-0.539313	1.983609	1.571684	-1.178277	1.175303	1.666403
3	-5.622982	-0.861145	-0.663774	0.639216	1.399097	0.464887	-5.800266	-0.833776	1.906510	1.299077
4	-3.663457	-2.540896	-0.362081	-1.018162	1.939464	-1.736997	-2.255150	6.621085	-3.086259	-5.767685
...	...	...	...	...	...	...	...	...	...	...
997	0.409122	1.001912	-1.288229	-0.403827	0.492523	-1.441248	-1.885862	-1.762801	-4.082646	-0.875871

Fig 6.5 SMOTE

```
# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Fit on training data, transform both train and test
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
joblib.dump(scaler, 'pickle/Standard_Scaler.pkl')
```

```
['pickle/Standard_Scaler.pkl']
```

Fig 6.6 Splitting data and Scaling data

```python
# Get model
def get_model(use_grid_search=False):
    base_model = XGBClassifier(
        use_label_encoder=False,
        eval_metric='logloss'
    )

    if use_grid_search:
        param_grid = {
            'max_depth': [4, 6],
            'learning_rate': [0.1, 0.2],
            'n_estimators': [100, 200]
        }

        grid_search = GridSearchCV(
            estimator=base_model,
            param_grid=param_grid,
            scoring='roc_auc',
            cv=3,
            verbose=1,
            n_jobs=-1
        )
        return grid_search

    return base_model


model = get_model()
model.fit(X_train, y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
```

Fig 6.7 Model Training

```python
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))


# Create SHAP explainer
def shap_explainer(model,X_test):
    explainer = shap.TreeExplainer(model)
    shap_values = explainer.shap_values(X_test)

    # Summary plot (feature impact)
    plt.figure()
    shap.summary_plot(shap_values, X_test, show=False)

    # Save plot as image (optional)
    plt.savefig('outputs/shap_summary_plot.png',bbox_inches='tight')
    plt.show()
```

```python
evaluate_model(model, X_test, y_test)
shap_explainer(model,X_test)
```

```
Confusion Matrix:
[[98 14]
 [ 8 80]]

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.88      0.90       112
           1       0.85      0.91      0.88        88

    accuracy                           0.89       200
   macro avg       0.89      0.89      0.89       200
weighted avg       0.89      0.89      0.89       200
```

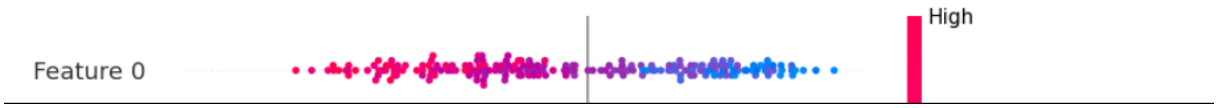


Fig 6.8 Evaluating Model and Shap Explainer

```
# Confusion matrix
y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap="Blues")
plt.title("Confusion Matrix")
plt.savefig("outputs/Confusion_Matrix.jpg")
plt.show()
```

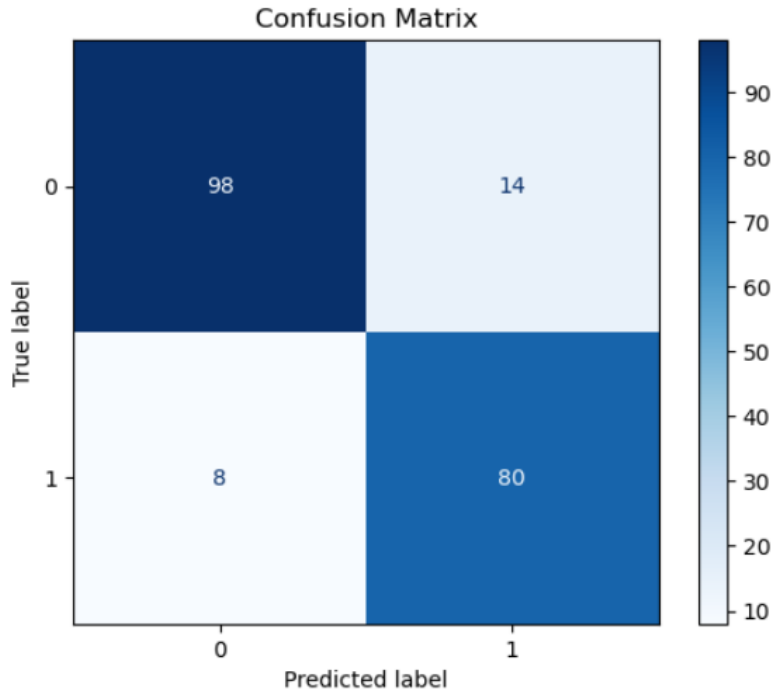


Fig 6.9 Confusion Matrix

```python
# ROC Curve
y_prob = model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"ROC curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend(loc="lower right")
plt.grid(True)
plt.savefig('outputs/ROC_Curve.jpg')
plt.show()
```

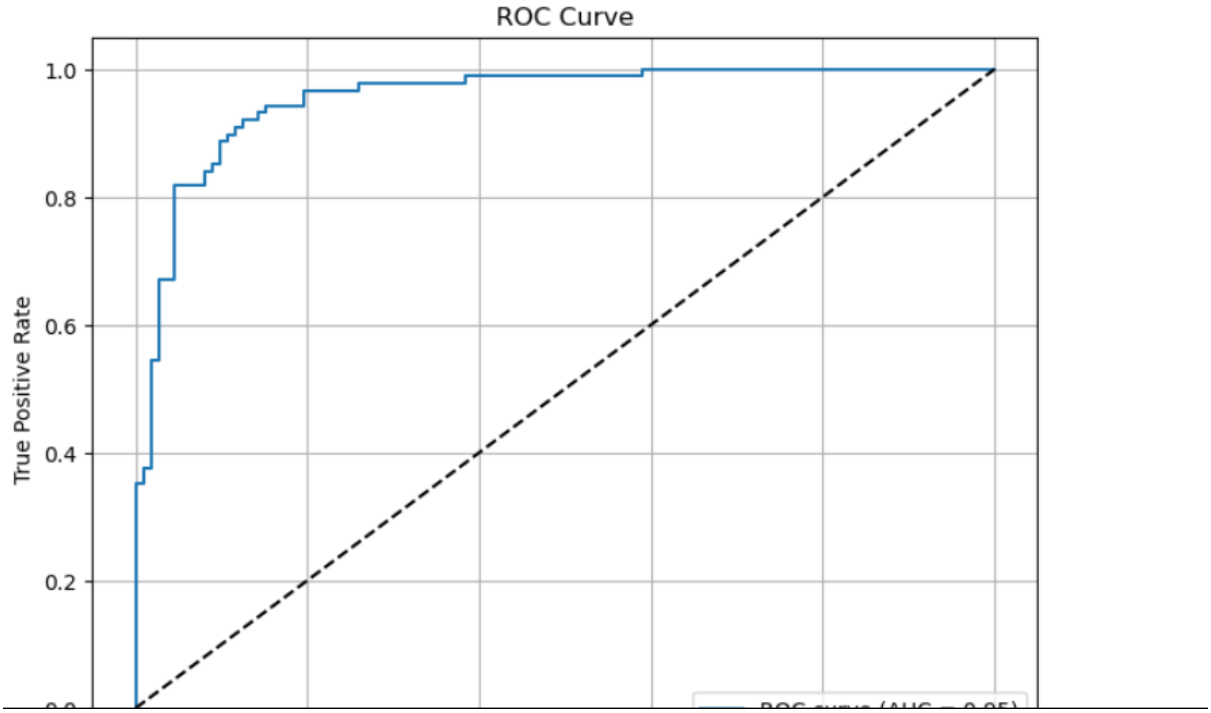
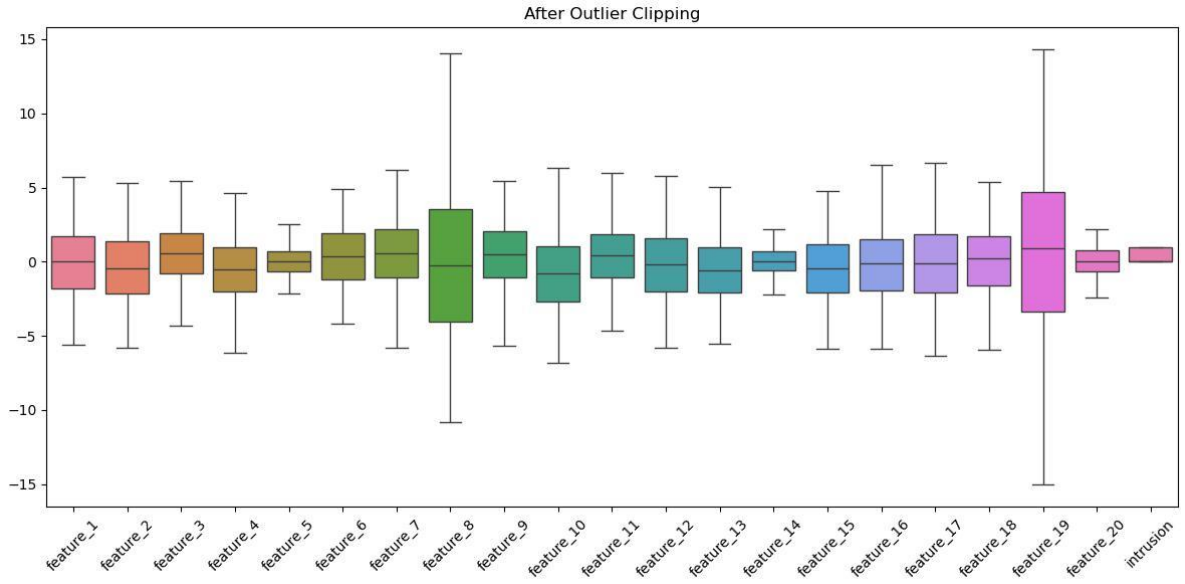


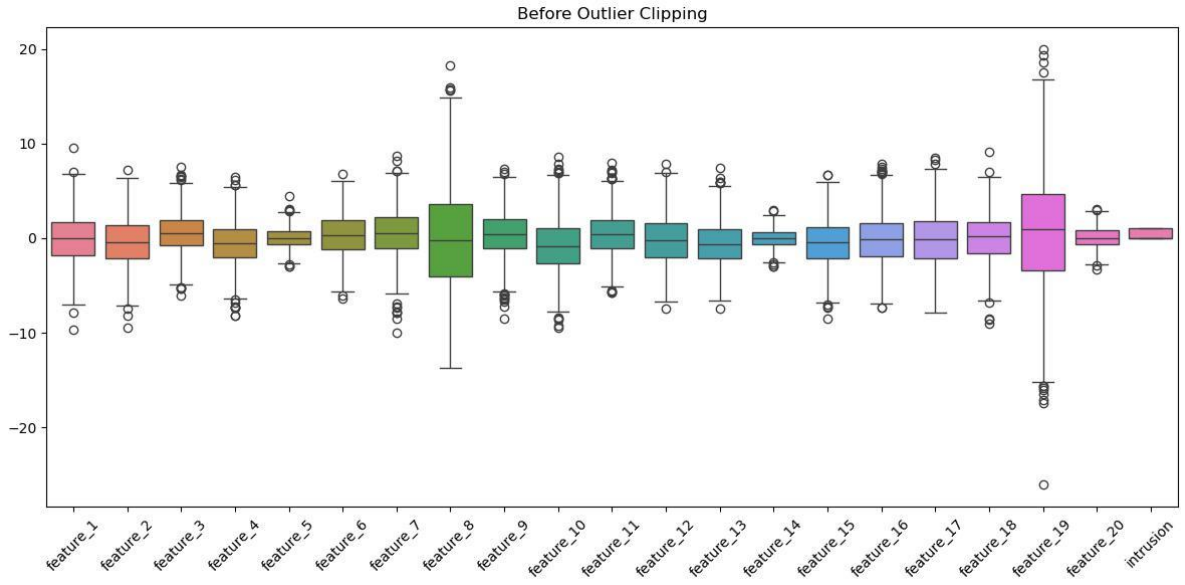
Fig 6.10 ROC Curve

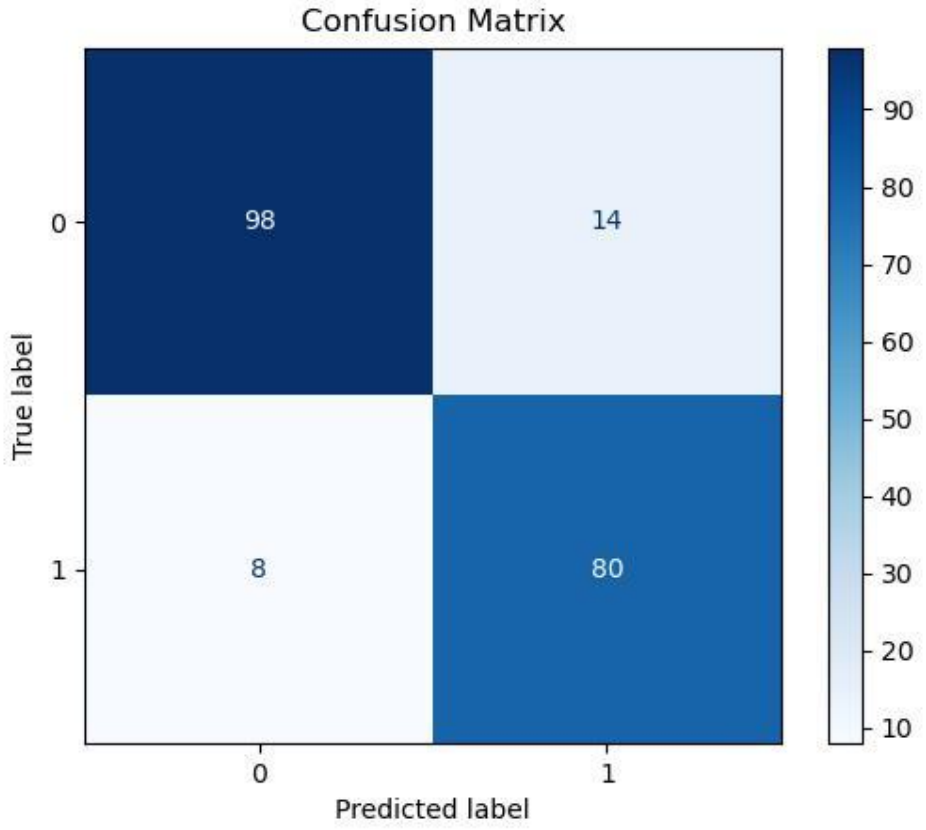
7. CONCLUSION

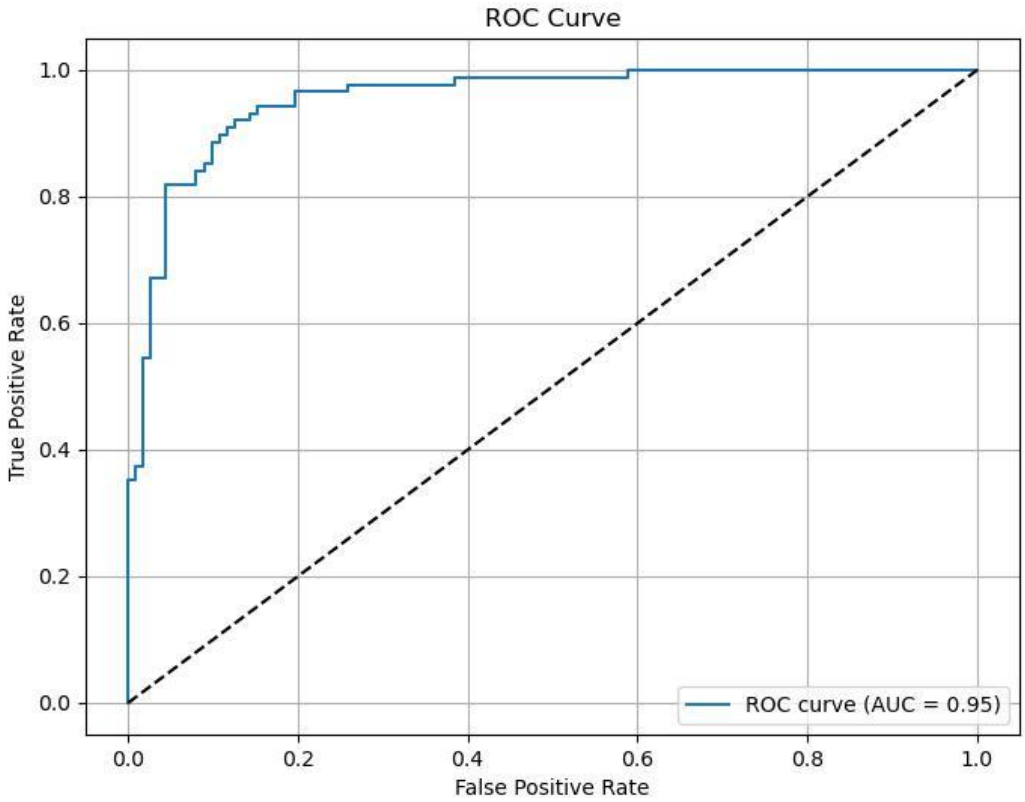
In conclusion, this project demonstrates that integrating machine-learning—specifically an XGBoost-based pipeline—into a Network Intrusion Detection System markedly improves the speed, accuracy, and adaptability of threat detection compared with traditional signature- or rule-driven methods. Through rigorous preprocessing (outlier clipping, scaling, SMOTE) and thorough validation (accuracy, ROC-AUC, confusion matrix, SHAP explainability), the model achieves high performance while remaining transparent for security analysts. The resulting, serialized IDS model is deployment-ready and can be seamlessly embedded in SOC workflows or real-time monitoring tools to flag both known and emerging attacks, thereby strengthening an organization's overall cybersecurity posture.

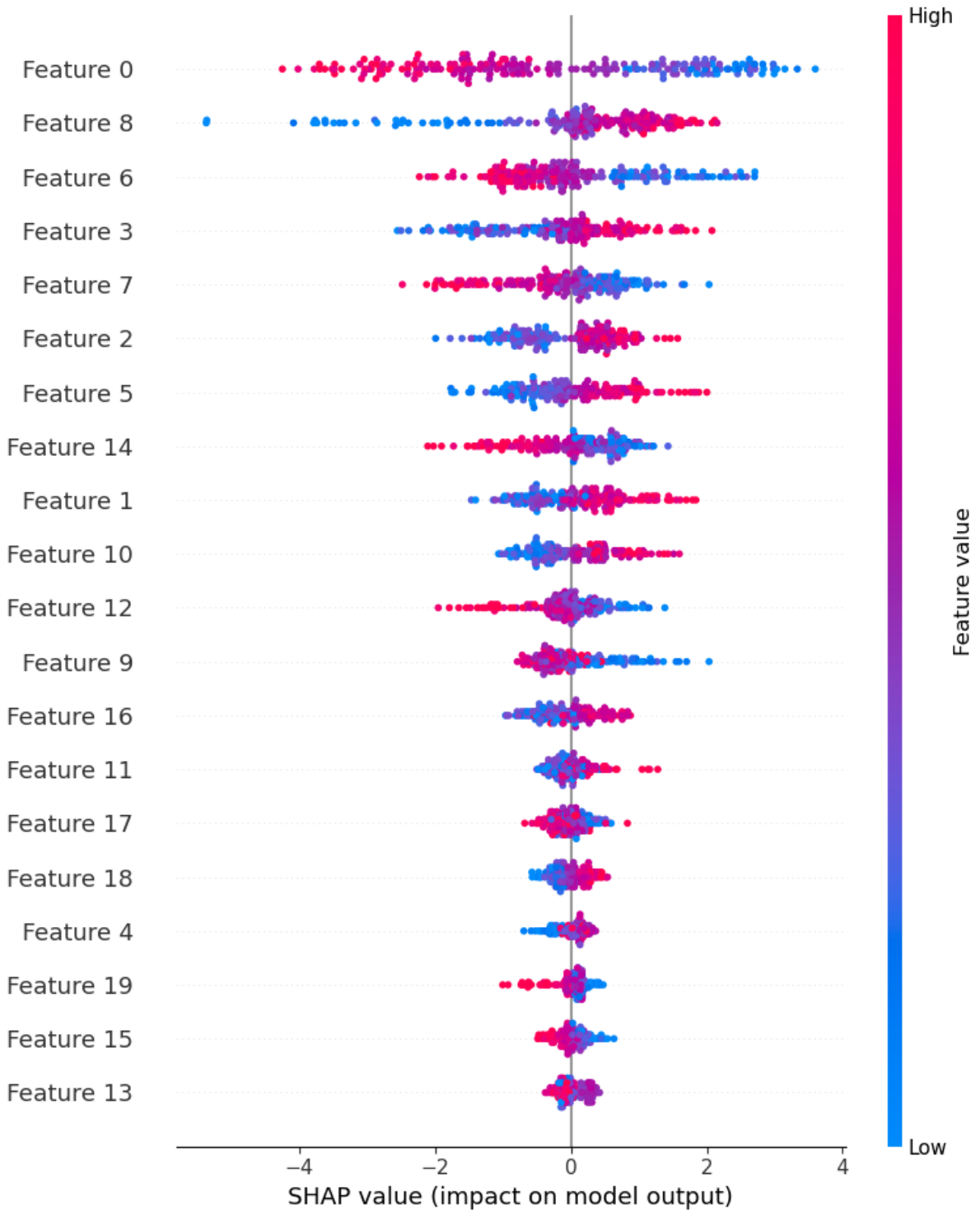
APPENDIX

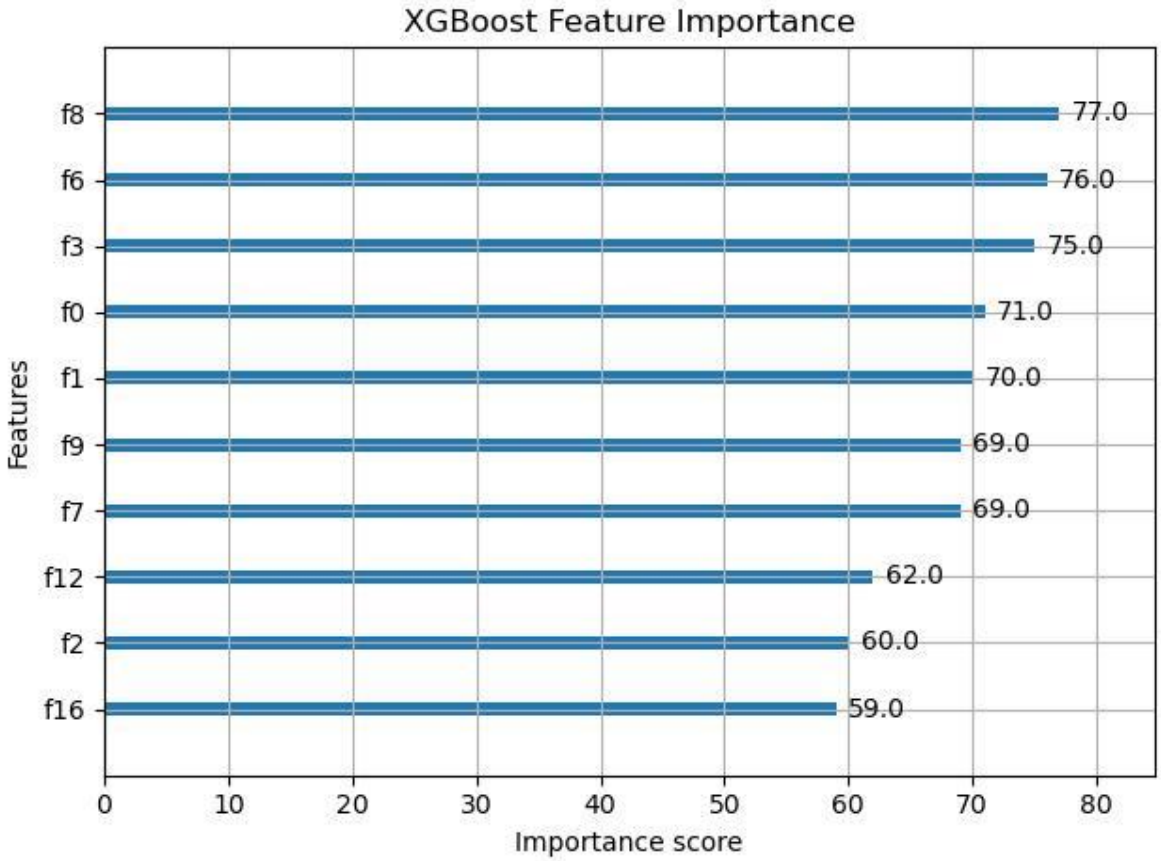

After Outlier Clipping
15
10
5
0
−5
−10
−15
feature_1
feature_2
feature_3
feature_4
feature_5
feature_6
feature_7
feature_8
feature_9
feature_10
feature_11
feature_12
feature_13
feature_14
feature_15
feature_16
feature_17
feature_18
feature_19
feature_20
intrusion


Before Outlier Clipping
20
10
0
−10
−20
feature_1
feature_2
feature_3
feature_4
feature_5
feature_6
feature_7
feature_8
feature_9
feature_10
feature_11
feature_12
feature_13
feature_14
feature_15
feature_16
feature_17
feature_18
feature_19
feature_20
intrusion


Confusion Matrix
98
14
8
80
0
1
True label
0
1
Predicted label
90
80
70
60
50
40
30
20
10


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.95)


Feature 0
Feature 8
Feature 6
Feature 3
Feature 7
Feature 2
Feature 5
Feature 14
Feature 1
Feature 10
Feature 12
Feature 9
Feature 16
Feature 11
Feature 17
Feature 18
Feature 4
Feature 19
Feature 15
Feature 13
−4
−2
0
2
4
SHAP value (impact on model output)
High
Feature value
Low


XGBoost Feature Importance
Features
f8
f6
f3
f0
f1
f9
f7
f12
f2
f16
77.0
76.0
75.0
71.0
70.0
69.0
69.0
62.0
60.0
59.0
0
10
20
30
40
50
60
70
80
Importance score