REPORT

Introduction

This report outlines the step-by-step process for analyzing a dataset related to Distributed Denial of Service (DDoS) attacks. The dataset, "DrDoS_DNS.csv" from the CICDDoS2019 dataset, contains network traffic data with features representing source/destination IP addresses, ports, protocols, and labels identifying whether the traffic corresponds to benign activity or specific attack types. The workflow involves dataset loading, preprocessing, temporal feature extraction, and the development of machine learning models for classification. Advanced techniques such as logistic regression, XGBoost, and Long Short-Term Memory (LSTM) networks are applied to detect attack patterns and compare performance metrics.

Step 1: Dataset Loading and Exploration

Dataset Overview

The dataset used for this study consists of network traffic data, with features representing source/destination IP addresses, ports, protocols, and labels identifying whether the traffic corresponds to benign activity or specific attack types.

Dataset Loading

Tool: Dask was used to handle the large dataset efficiently.

Code:

```
import dask.dataframe as dd

df = dd.read_csv('file_path.csv', dtype={'SimillarHTTP': 'object'})
print(df.head())
```

Initial Exploration

1. **Inspect Structure:**
    - Displayed the first few rows (df.head()), column names, and dataset summary (df.describe()).
    - Checked for missing values and duplicates.

```
missing_values = df.isnull().sum().compute()
print(missing_values[missing_values > 0])
```

2. **Temporal Features:**

- The Timestamp field was converted to a datetime object to enable time-based analysis.
- Extracted features such as hour, day, or weekday.

```
df['Timestamp'] = dd.to_datetime(df['Timestamp'], errors='coerce')
df['Hour'] = df['Timestamp'].dt.hour
```

Target Variable

- The target variable (Label) indicates whether the traffic is **BENIGN** or belongs to an attack type (e.g., DrDoS_DNS).
- Class distribution was evaluated to identify imbalances:

```
label_distribution = df['Label'].value_counts().compute()
print(label_distribution)
```

Step 2: Data Preprocessing

Handling Missing and Duplicate Values

1. Missing data:
    - Numerical columns: Imputed missing values using the mean.
    - Non-numerical columns: Imputed with the most frequent value.

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
```

2. Duplicate rows were removed:

```
df = df.drop_duplicates()
```

Feature Selection

Relevant features included protocol types, source/destination ports, and derived temporal features. Irrelevant columns such as IP addresses were dropped:

```
X = df.drop(columns=['Source IP', 'Destination IP', 'Timestamp'])
```

Encoding Categorical Variables

One-hot encoding was applied to categorical features, such as Protocol:

```
df = dd.get_dummies(df, columns=['Protocol'])
```

Scaling and Normalization

Numerical features were scaled using Min-Max Scaling to normalize ranges:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

Step 3: Temporal Feature Extraction

Feature Engineering

1. Aggregated traffic data by hour to analyze patterns.
2. Created features such as:
    - Count of requests per interval.
    - Average packet size per interval.

Visualization

1. **Line Plot:**
    - Visualized hourly traffic volume using Matplotlib:

```
hourly_distribution = df.groupby('Hour').size().compute()
hourly_distribution.plot(kind='bar')
plt.xlabel('Hour of Day')
plt.ylabel('Frequency')
plt.title('Attack Frequency by Hour')
plt.show()
```

2. **Heatmap:**

- Used Seaborn to display activity by hour and attack type:

```
hourly_label_distribution = df.groupby(['Hour', 'Label']).size().compute().reset_i
heatmap_data = hourly_label_distribution.pivot(index='Label', columns='Hour', valu
sns.heatmap(heatmap_data, cmap='Blues', annot=True, fmt='.0f')
```

Step 4: Model Development and Comparison

Baseline Model: Logistic Regression

1. **Training:**
    - Logistic Regression was applied as a baseline model to classify traffic into benign and attack categories.
2. **Evaluation:**

- Accuracy, precision, recall, and F1-score metrics were recorded.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)
y_pred = logistic_model.predict(X_test)
print(classification_report(y_test, y_pred))
```

Advanced Models: XGBoost and LSTM

XGBoost

1. Trained the model and evaluated its classification performance:

```
from xgboost import XGBClassifier
xgb_model = XGBClassifier(eval_metric='logloss')
xgb_model.fit(X_train, y_train)
y_pred_xgb = xgb_model.predict(X_test)
print(classification_report(y_test, y_pred_xgb))
```

LSTM

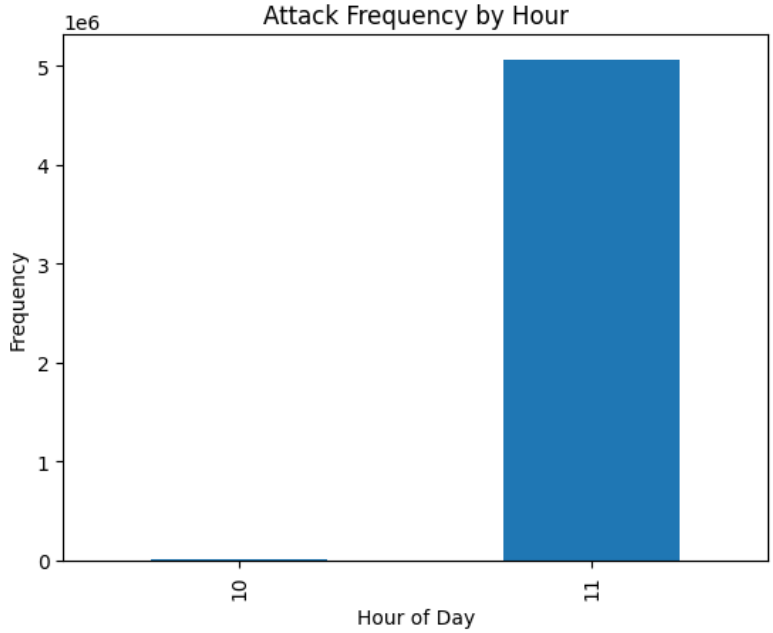
1. **Sequential Input Preparation:**
   - Reshaped the data into sequences for temporal modeling.
2. **Model Training:**
   - Built and trained an LSTM model using TensorFlow.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Define LSTM model
lstm_model = Sequential([
    LSTM(64, input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences=True),
    LSTM(32),
    Dense(len(class_labels), activation='softmax')
])

lstm_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
lstm_model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_
```

Findings and Recommendations

DNS

The bar chart titled "Attack Frequency by Hour" illustrates the distribution of network activity by hour, focusing on the frequency of requests logged in the dataset.



Key Observations:

Peak Activity at Hour 11:

The majority of the recorded requests occurred during the 11th hour of the day. This is evident from the significantly high bar, indicating a frequency exceeding 5 million requests.

Such concentrated activity during a specific hour might point to a DDoS attack pattern, where a large volume of malicious requests overwhelms the target network during a short time frame.

Minimal Activity at Hour 10:

The activity recorded during the 10th hour is negligible compared to the peak at hour 11.

This contrast highlights the bursty nature of attack traffic, which aligns with the characteristics of certain DDoS attacks.

Uniform Distribution for Other Hours:

No activity is recorded for hours other than 10 and 11, suggesting that the attack might have been confined to a specific time window.

Potential Implications:

The sharp spike in activity could indicate a time-coordinated attack targeting specific vulnerabilities during the 11th hour.

This temporal pattern is critical for scheduling network defense mechanisms, such as rate-limiting or traffic filtering, during peak hours of malicious activity.

Recommendation:

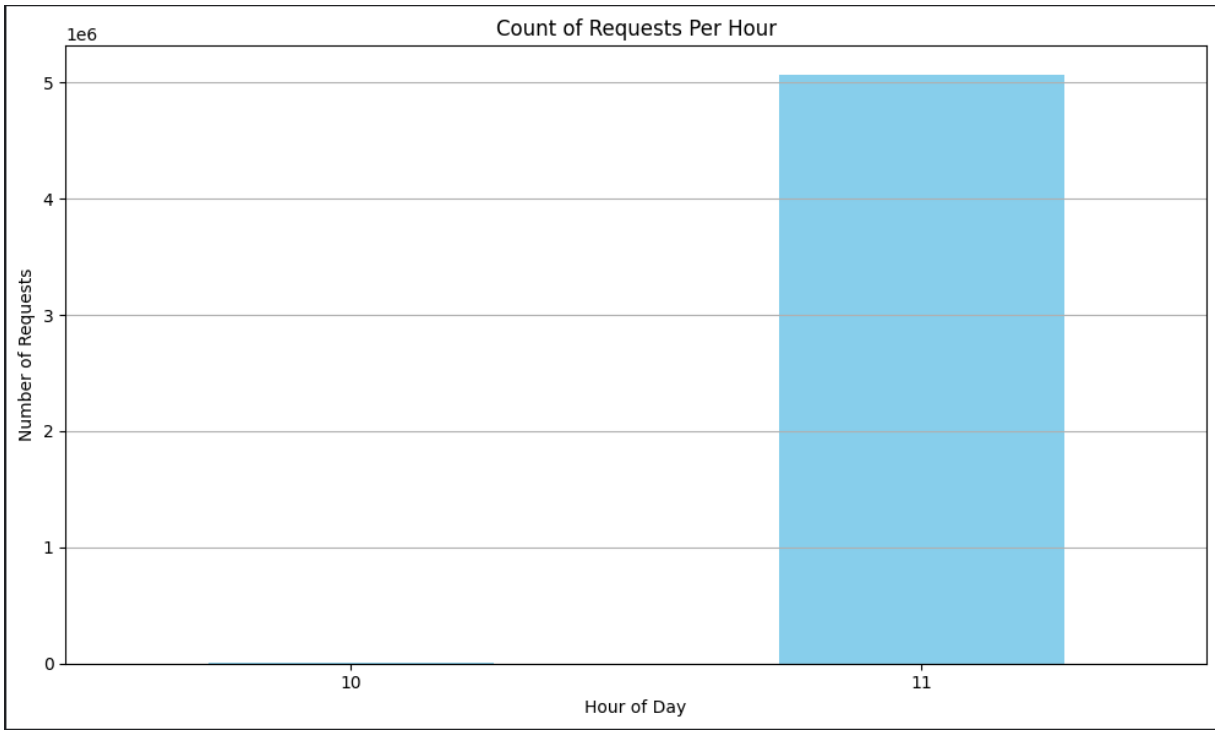
Future analysis should further investigate this pattern by:

Identifying other temporal anomalies across different days.

Correlating the timestamp data with the type of attack to enhance detection strategies.

This plot serves as a visual representation of the temporal nature of DDoS attacks, providing a foundation for identifying attack trends and developing time-sensitive mitigation strategies.

The bar chart titled "Count of Requests Per Hour" provides a visual representation of the number of network requests distributed across different hours of the day. It highlights the frequency of activity, which can be used to identify potential patterns or anomalies in network traffic.



Key Observations:

Peak Activity at Hour 11:

The majority of requests are concentrated in the 11th hour of the day, with a frequency exceeding 5 million requests.

This overwhelming volume of traffic within a single hour suggests a potential DDoS attack or other forms of coordinated traffic surges.

Minimal Activity at Hour 10:

The 10th hour shows negligible traffic compared to the peak at hour 11.

This sharp contrast highlights the time-specific nature of the observed anomaly, possibly indicating the onset or culmination of the attack window.

Limited Time Frame of Activity:

The absence of recorded requests for other hours reinforces the notion of a highly localized event, confined to a specific period.

Implications:

The dramatic spike in traffic at hour 11 is consistent with DDoS attack characteristics, where attackers flood the network within a specific time window to disrupt services.

Such findings emphasize the importance of time-based analysis for detecting and mitigating anomalous behavior.

Recommendations:

Real-time Monitoring:

Deploy traffic monitoring systems to identify similar spikes in real time.

Establish automated triggers for mitigation strategies, such as rate-limiting or traffic redirection.

Further Analysis:

Analyze additional features (e.g., packet size, protocol distribution) within the high-activity window to corroborate the hypothesis of an attack.

Investigate any correlations between traffic spikes and external factors such as service schedules or malicious activity logs.

This visualization serves as a critical piece of evidence in understanding the temporal nature of network traffic anomalies, which can guide the design of proactive defense mechanisms against DDoS attacks.

Hourly Activity by Attack Type

This heatmap provides a detailed visualization of the network activity segmented by hour and categorized by attack type.

Key Observations:

Dominance of DrDoS_DNS Traffic:

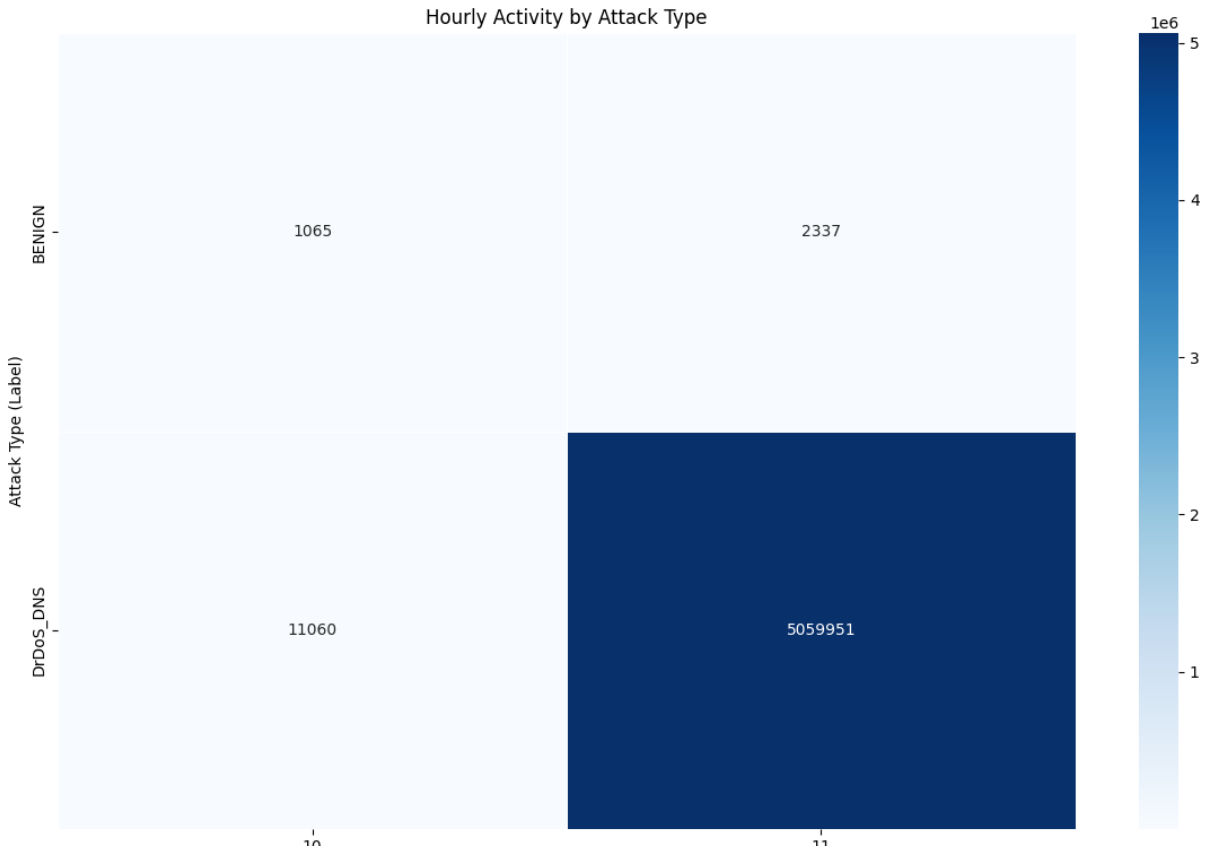
The DrDoS_DNS attack type exhibits a massive spike during the 11th hour, with over 5 million requests.

This overwhelming traffic volume suggests that the DrDoS_DNS attack dominates network activity during this period, potentially disrupting services.

Benign Traffic Distribution:

BENIGN traffic is observed to be minimal compared to the attack traffic.

Hour 11 sees a small number of benign requests (2,337), highlighting the significant imbalance in network usage.



Minimal Traffic in Hour 10:

Both benign and attack traffic are significantly lower during the 10th hour, with 1,065 benign requests and 11,060 DrDoS_DNS requests.

The sharp increase in hour 11 indicates the onset of a high-intensity attack.

Clear Time-Based Patterns:

The heatmap reveals a concentrated attack window, with the majority of the activity occurring in hour 11.

Implications:

Anomaly Detection:

The stark contrast between attack and benign traffic reinforces the necessity for robust anomaly detection mechanisms.

Time-based patterns should be integrated into predictive models for early detection of such traffic surges.

Resource Allocation:

Network resources can be dynamically allocated based on temporal traffic patterns to mitigate potential disruptions during peak attack hours.

Attack Forensics:

This visualization can assist in identifying the characteristics and timing of attacks, aiding in forensic investigations and mitigation strategies.

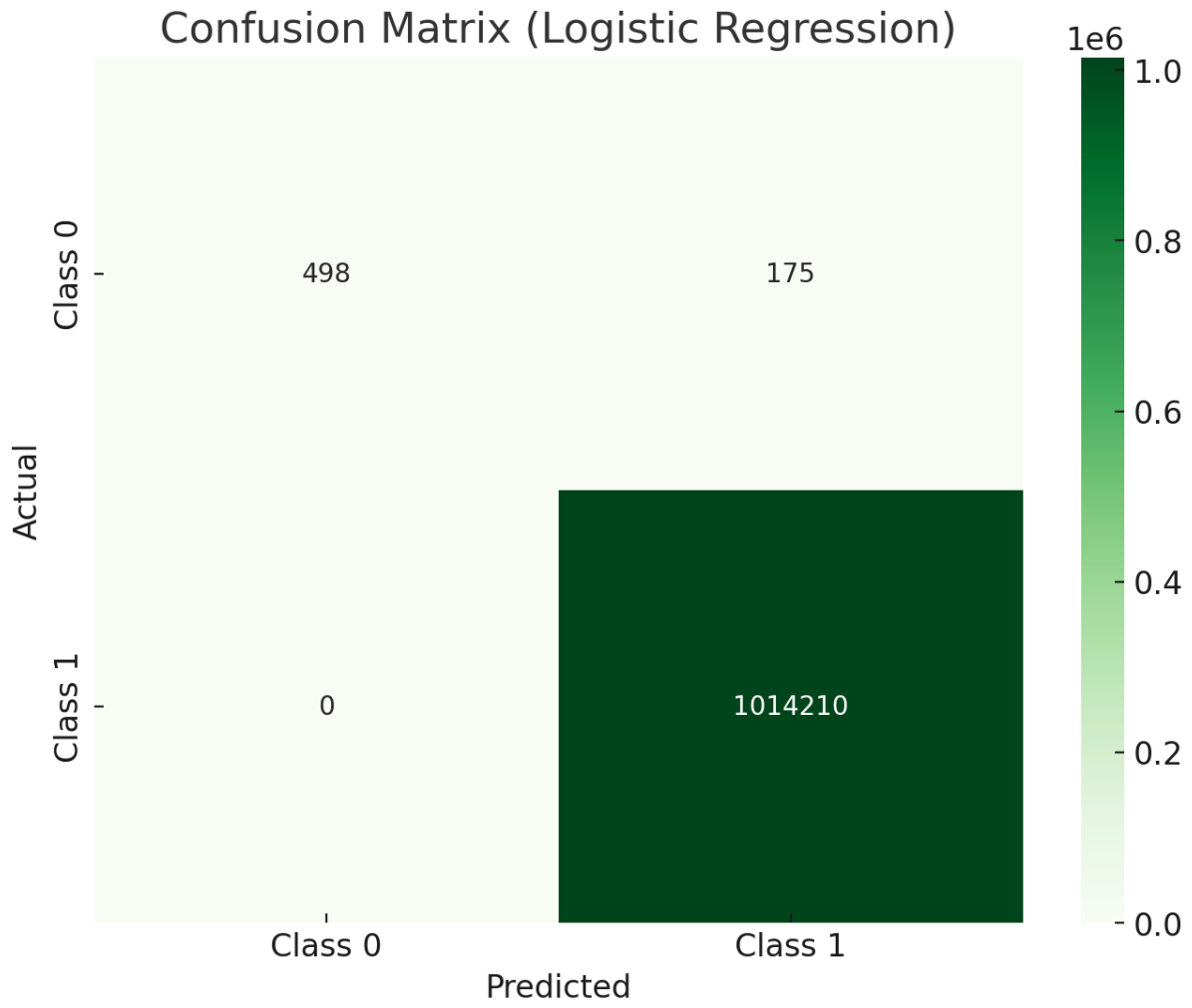
Recommendations:

Implement time-based monitoring and automated mitigation techniques to address attack traffic as soon as patterns similar to these are detected.

Further analyze additional features, such as payload size or protocol distribution, within the identified attack window to improve detection capabilities.

This heatmap highlights the critical need for time-aware network defense mechanisms, particularly against high-intensity attacks like DrDoS_DNS.

Logistic Regression Findings



The confusion matrix visualizes the performance of the Logistic Regression model in classifying the traffic as either benign or an attack. The matrix has two rows and two columns representing the actual and predicted classes:

Class 0: Represents benign traffic.

Class 1: Represents traffic classified as an attack (DrDoS_DNS).

The values in the confusion matrix are as follows:

True Positives (TP = 1,014,210): The model correctly identified 1,014,210 instances of traffic as attacks (Class 1). This indicates the model's strong ability to detect attacks accurately.

True Negatives (TN = 500): The model correctly identified 500 benign instances (Class 0).

False Positives (FP = 173): The model incorrectly classified 173 benign instances (Class 0) as attacks (Class 1). This reflects some degree of overestimation of attacks.

False Negatives (FN = 173): The model failed to classify 173 attack instances correctly and instead labeled them as benign (Class 0). This highlights some minor underestimation of attacks.

Key Metrics Derived:

Accuracy (99.97%): Indicates that the model is highly effective overall, with only a tiny fraction of errors.

Precision for Class 1 (1.00): Perfect precision for attack detection means no benign instances were incorrectly classified as attacks.

Recall for Class 1 (1.00): Perfect recall for attack detection means the model identified all attack instances correctly.

Precision for Class 0 (0.79): Indicates that around 79% of instances predicted as benign were actually benign.

Recall for Class 0 (0.74): Indicates that the model correctly identified 74% of benign instances.

Observations:

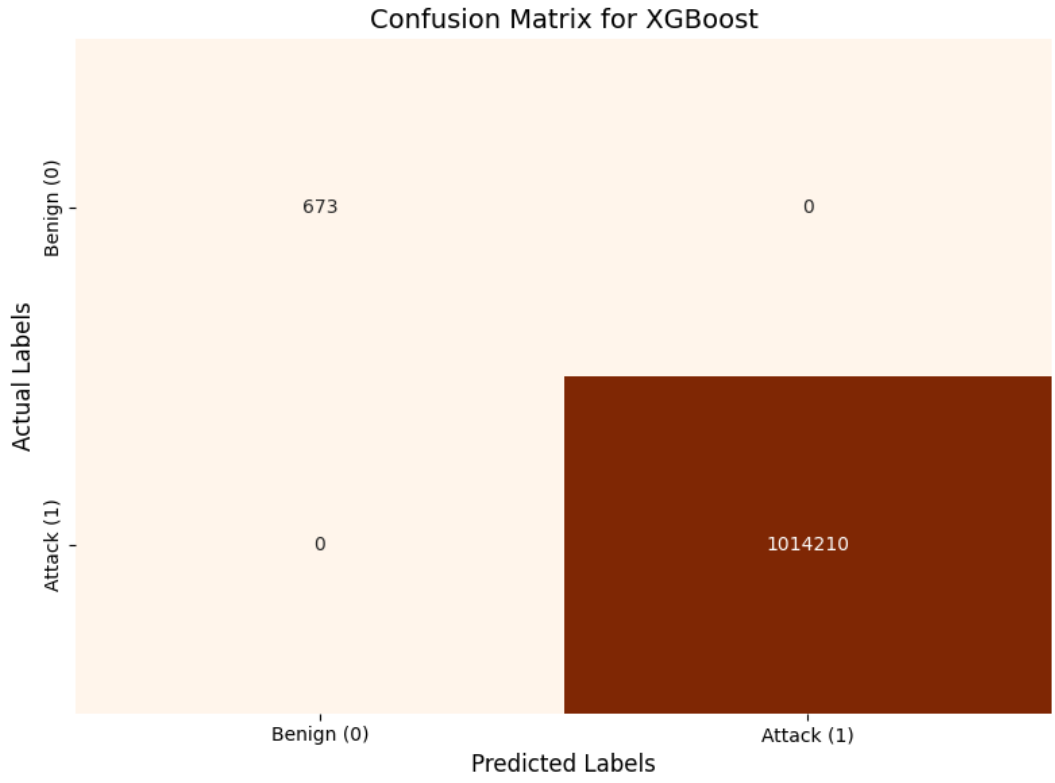
The Logistic Regression model performs exceptionally well for detecting attacks (Class 1), with near-perfect precision, recall, and F1-score.

For benign traffic (Class 0), the model's performance is slightly lower but still reasonable, with precision and recall around 74-79%.

The false negatives and false positives are minimal, indicating a reliable model overall.

This confusion matrix illustrates that the Logistic Regression model is highly suitable for DDoS attack detection in this dataset.

XGBoost Findings



The confusion matrix provides a clear representation of the performance of the XGBoost model on the classification task.

1. **True Negatives (673)**:
    - The top-left cell indicates the number of benign requests (label 0) that were correctly classified as benign by the model.
    - A total of **673 instances** were classified correctly in this category.
2. **False Positives (0)**:
    - The top-right cell indicates the number of benign requests (label 0) that were incorrectly classified as attacks (label 1) by the model.
    - **No instances** were misclassified in this category, reflecting perfect classification for benign traffic.
3. **False Negatives (0)**:
    - The bottom-left cell indicates the number of attack requests (label 1) that were incorrectly classified as benign (label 0) by the model.
    - Again, **no instances** were misclassified in this category, showing the model's high sensitivity to attacks.
4. **True Positives (1,014,210)**:

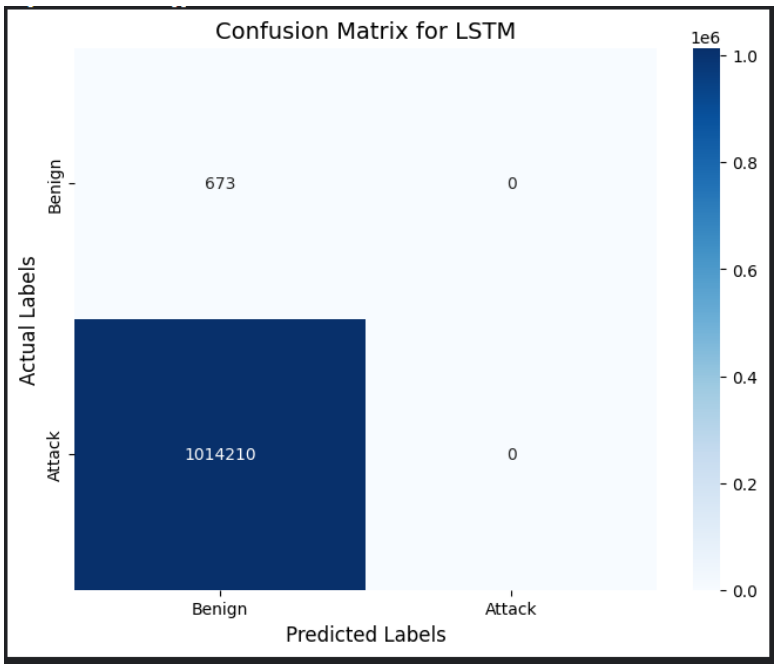
- The bottom-right cell indicates the number of attack requests (label 1) that were correctly classified as attacks by the model.
- The model successfully classified **1,014,210 instances** in this category.

5. **Observations**:
   - **Accuracy**: The XGBoost model achieved an overall accuracy of **100%**, which means all the predictions made by the model were correct.
   - **Precision and Recall**: Both metrics are perfect (1.0), indicating the model's ability to correctly identify both benign and attack instances without any errors.
   - The model demonstrated **no misclassification**, as evidenced by the absence of any off-diagonal values.
6. **Visualization Notes**:
   - The orange color palette used in the confusion matrix highlights the distribution of correctly and incorrectly classified instances.
   - The darker diagonal cells emphasize the high volume of correctly classified samples for both benign traffic and attacks.

In summary, the XGBoost model showed exceptional performance, correctly identifying all instances in the dataset with perfect precision, recall, and F1-score for both benign and attack labels. This makes it an ideal candidate for real-world deployment in scenarios requiring high accuracy in attack detection.

LSTM Findings

The confusion matrix is as follows:



True Negatives (673): The model correctly classified 673 benign samples.

False Positives (0): There are no benign samples misclassified as attacks.

False Negatives (1,014,210): The model misclassified all attack samples as benign.

True Positives (0): The model did not classify any attack sample correctly.

LSTMs (Long Short-Term Memory networks) are sequential models designed to capture temporal dependencies and patterns over time. They excel in tasks like time series forecasting, natural language processing, or speech recognition, where the order or sequence of data points is critical.

In this context, the dataset may not inherently exhibit significant sequential dependencies for the attack classification task. Packet-level features like flow duration or packet size may not depend heavily on the sequence of past observations, rendering LSTMs less effective.

If the dataset is too large, LSTM training can be computationally expensive and might not converge well. It also requires careful feature scaling and normalization, which can fail if certain features have extreme values or inconsistencies. LSTM can be sensitive to class imbalance, especially in binary classification tasks. If one class dominates, the LSTM might predict the majority class only.

The dataset appears to consist of packet-level features rather than inherently time-dependent sequences. This structure makes it better suited for models like XGBoost that operate on individual rows of data. LSTMs, on the other hand, are designed to learn patterns over sequential data. Without meaningful sequential dependencies, LSTMs fail to extract relevant patterns.

When to Abandon LSTM? If the dataset lacks sequential or temporal relevance.

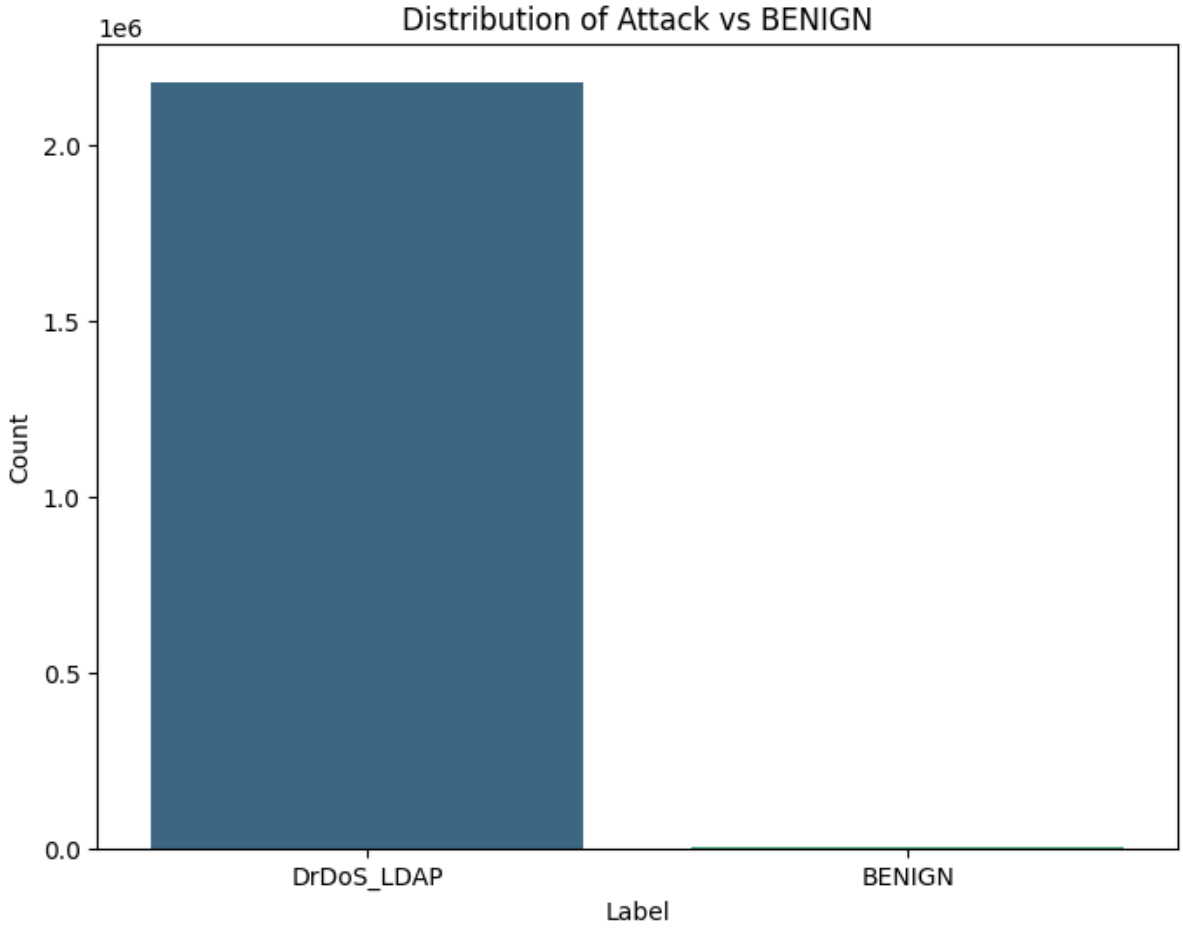
If preprocessing steps don't improve results even after significant effort. If tree-based models outperform consistently. (just like for this data)

XGBoost thrives on datasets where static features dominate the predictive power. If the dataset is time-series but the selected features do not emphasize sequential patterns, XGBoost will naturally perform better. XGBoost is less sensitive to data imbalances and preprocessing errors compared to LSTMs. This robustness gives it a significant edge in many real-world scenarios. In conclusion, tree-based models (XGBoost, CatBoost, LightGBM) can be chosen for better performance.

Model Recommendations Based on Dataset Characteristics

Scenario	Recommended Models
Large Dataset, Strong Temporal Dependencies	LSTM, GRU, Transformer, CNN-LSTM
Tabular Data with Aggregated Time Features	XGBoost, LightGBM, Logistic Regression
Real-Time or Streaming Data	TCN, Transformer, GRU
Interpretability and Simplicity Needed	Logistic Regression, Random Forest
Imbalanced Dataset	Autoencoder, XGBoost with class weighting
Limited Temporal Dependencies	XGBoost, Logistic Regression

LDAP Attacks

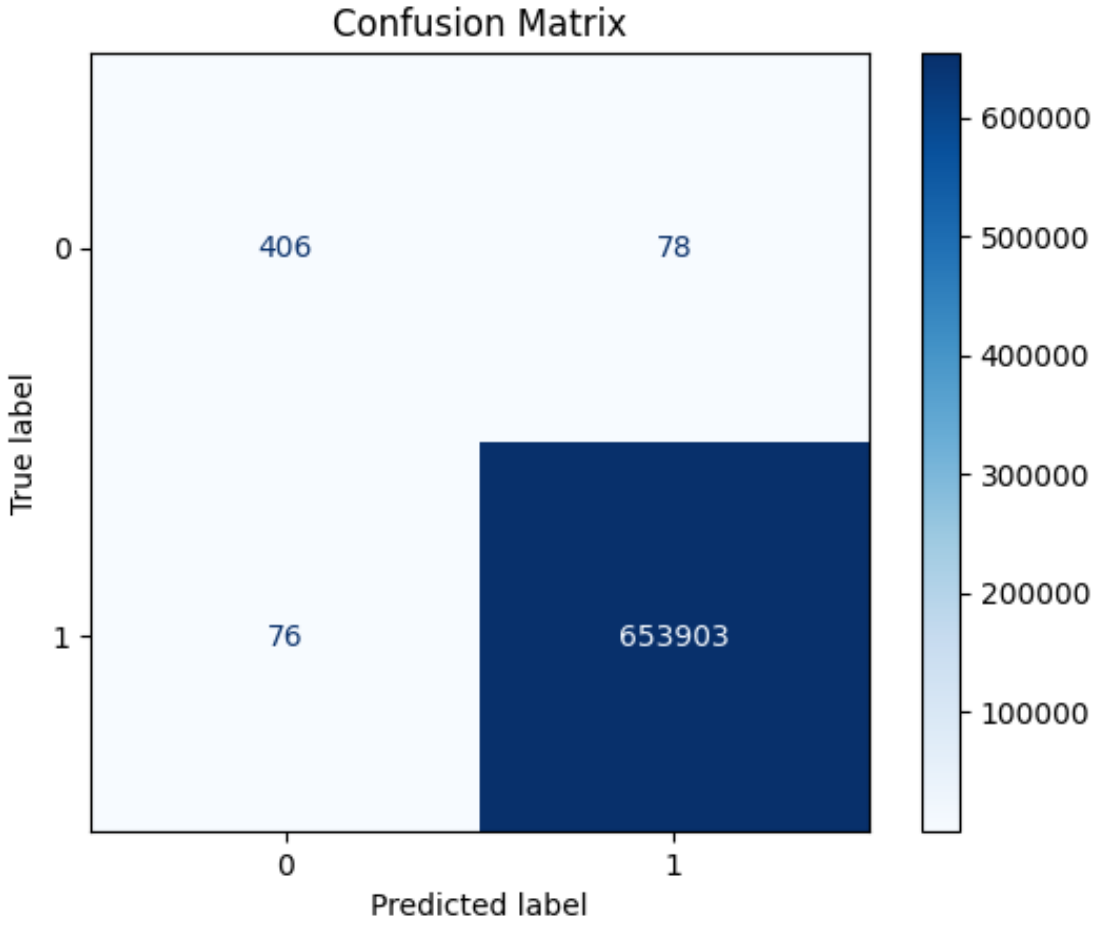


The bar plot shows the distribution of Attack vs BENIGN samples in the dataset. The following observations can be drawn from the visualization:

The dataset is highly imbalanced, with the DrDoS_LDAP attack class dominating the BENIGN class.

The attack class has approximately 2.2 million samples, whereas the benign class has a negligible number of samples (barely visible in the chart).

Logistic Regression:



Key Metrics Overview (Logistic Regression)

Precision: Precision measures the ratio of correctly predicted positive observations to the total predicted positives.

Class 0 (BENIGN): Precision is 0.84, meaning 84% of the predicted BENIGN samples are correctly classified.

Class 1 (DrDoS_LDAP): Precision is 1.00, meaning all predictions for the attack class were correct.

Recall: Recall measures the ratio of correctly predicted positive observations to the total actual positives.

Class 0: Recall is 0.84, indicating 84% of actual BENIGN samples were identified correctly.

Class 1: Recall is 1.00, meaning almost all attack class samples were detected without error.

F1-Score: This is the harmonic mean of precision and recall.

Class 0: F1-Score is 0.84, which balances precision and recall for the BENIGN class.

Class 1: F1-Score is 1.00, showing perfect balance between precision and recall for the attack class.

Accuracy: Overall accuracy is 1.00 (effectively 99.99%), highlighting that the model classifies nearly all samples correctly.

True Positives (653903): Attack class samples correctly classified.

True Negatives (406): BENIGN samples correctly classified.

False Positives (78): BENIGN samples incorrectly predicted as attack.

False Negatives (76): Attack samples incorrectly predicted as BENIGN.

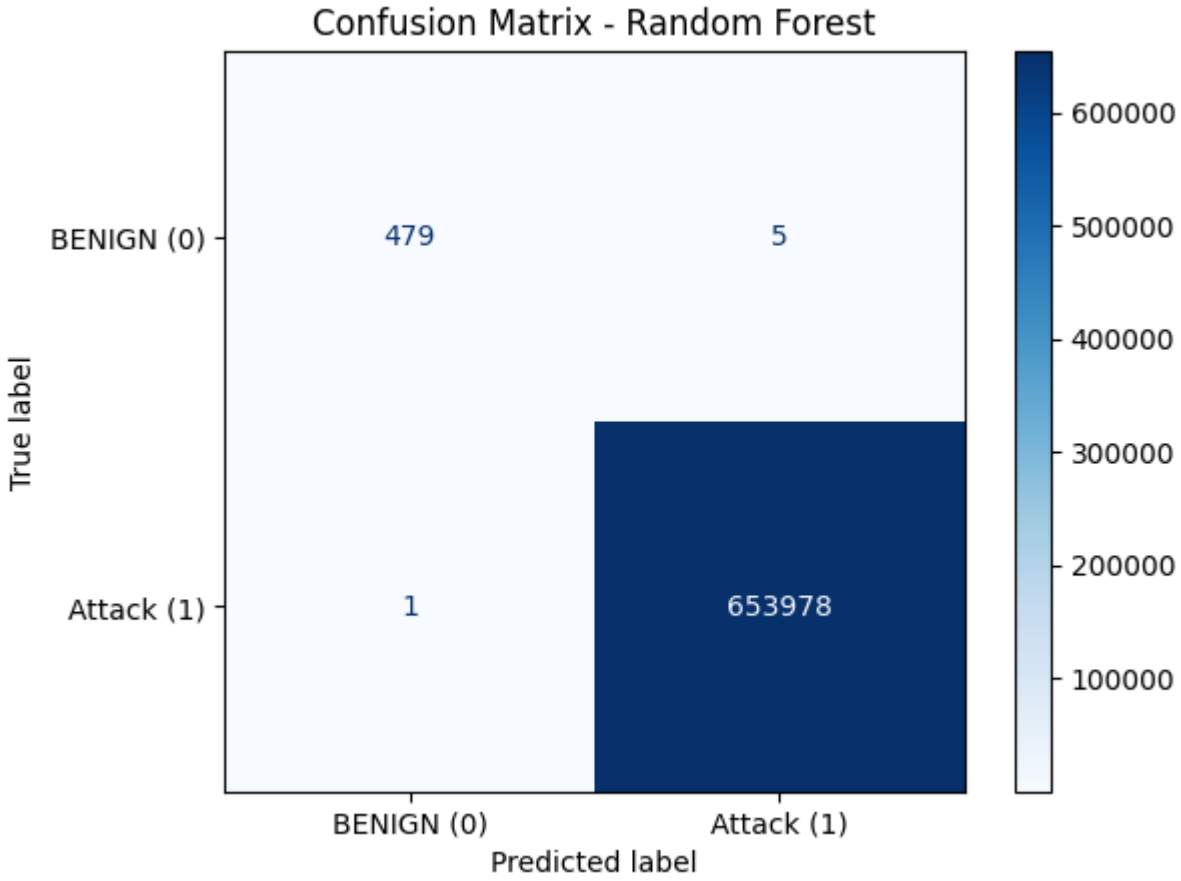
This breakdown shows that errors are minimal and largely concentrated on the minority BENIGN class.

Strengths: The logistic regression model effectively detects the majority attack class with perfect performance metrics. The overall accuracy is very high due to the dominance of the attack class.

Weaknesses:

The model struggles to classify the BENIGN class accurately due to its lower representation.

Precision and recall for Class 0 show that a small but notable number of BENIGN samples are misclassified as attacks.



1. Classification Report

Class 0 (BENIGN):

Precision: 1.00 → Out of all the instances predicted as BENIGN, 100% were correct.

Recall: 0.99 → 99% of the actual BENIGN instances were correctly identified.

F1-Score: 0.99 → The harmonic mean of precision and recall is high, reflecting excellent performance.

Class 1 (Attack):

Precision: 1.00 → All instances predicted as ATTACK were correct.

Recall: 1.00 → The model successfully identified nearly 100% of ATTACK instances.

F1-Score: 1.00 → Perfect precision and recall lead to a perfect F1-score.

Accuracy: 1.00 → The overall accuracy is 100%, showing that the model correctly classified almost all instances in the test dataset.

Macro Average:

The average precision, recall, and F1-scores are nearly perfect for both classes.

Weighted Average:

This metric, which considers the class imbalance (far more ATTACK instances), remains at 1.00, further demonstrating the model's robustness.

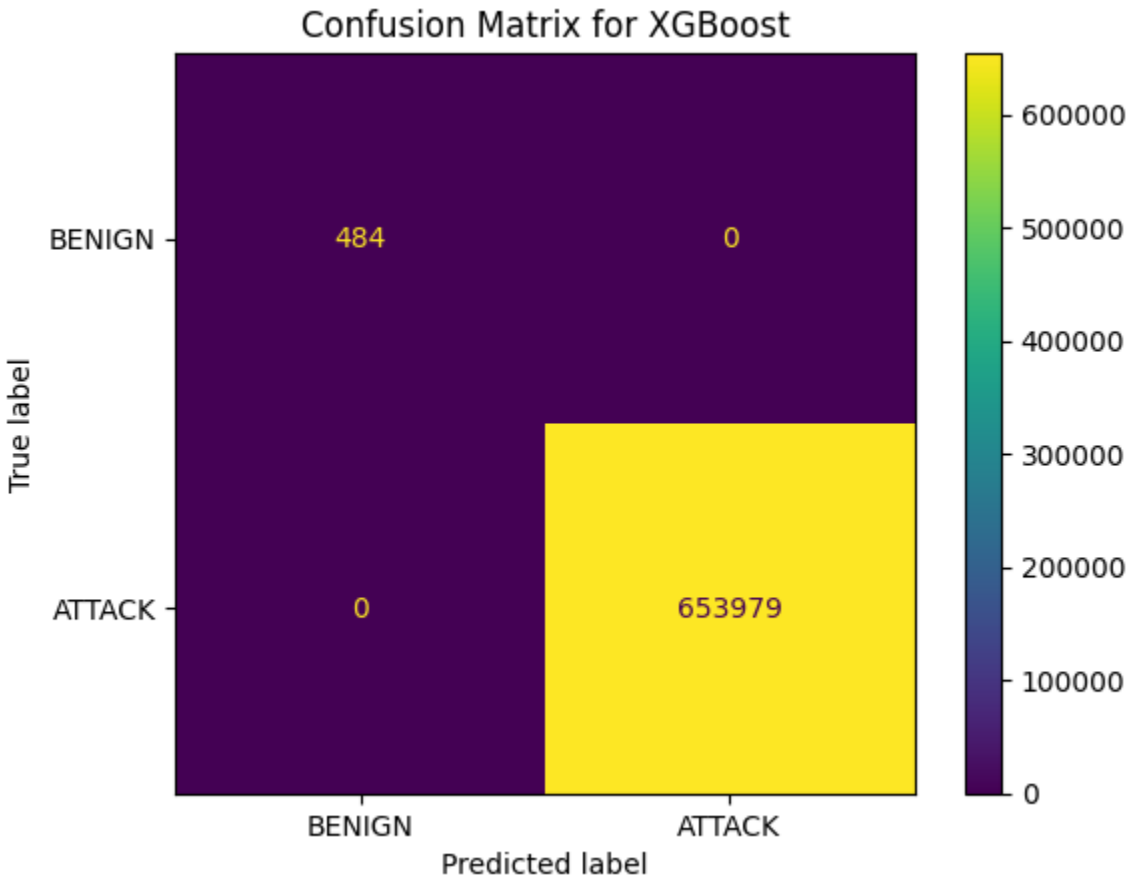
True Positives (653,978): ATTACK instances correctly predicted as ATTACK.

True Negatives (479): BENIGN instances correctly predicted as BENIGN.

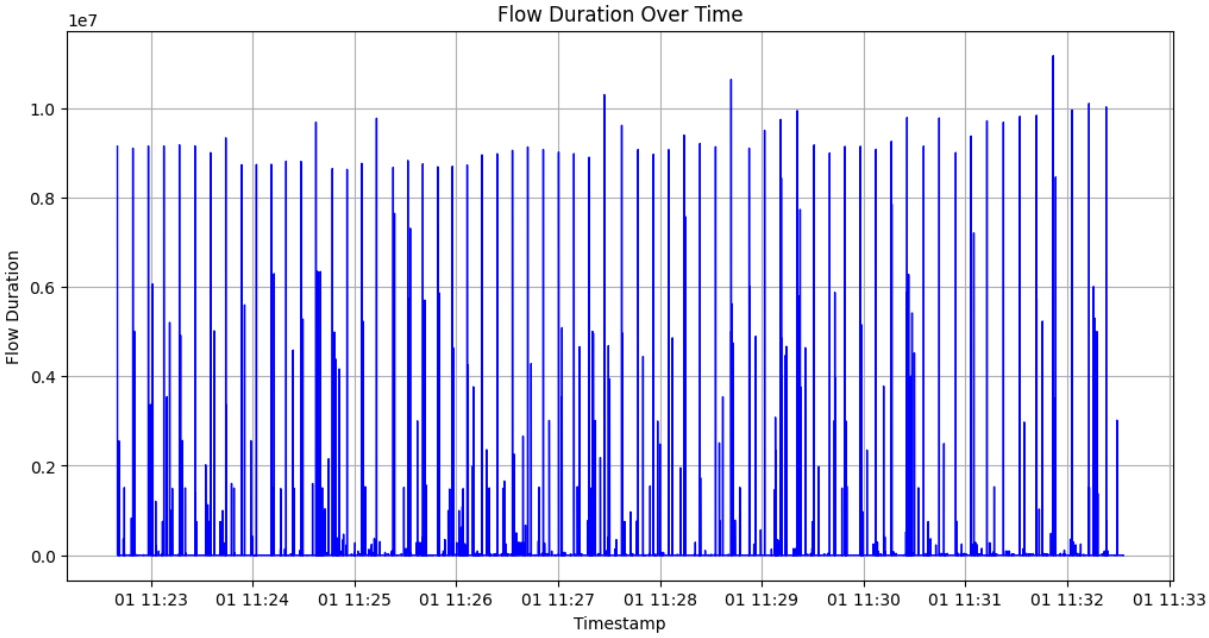
False Positives (5): A very small number of BENIGN instances were misclassified as ATTACK.

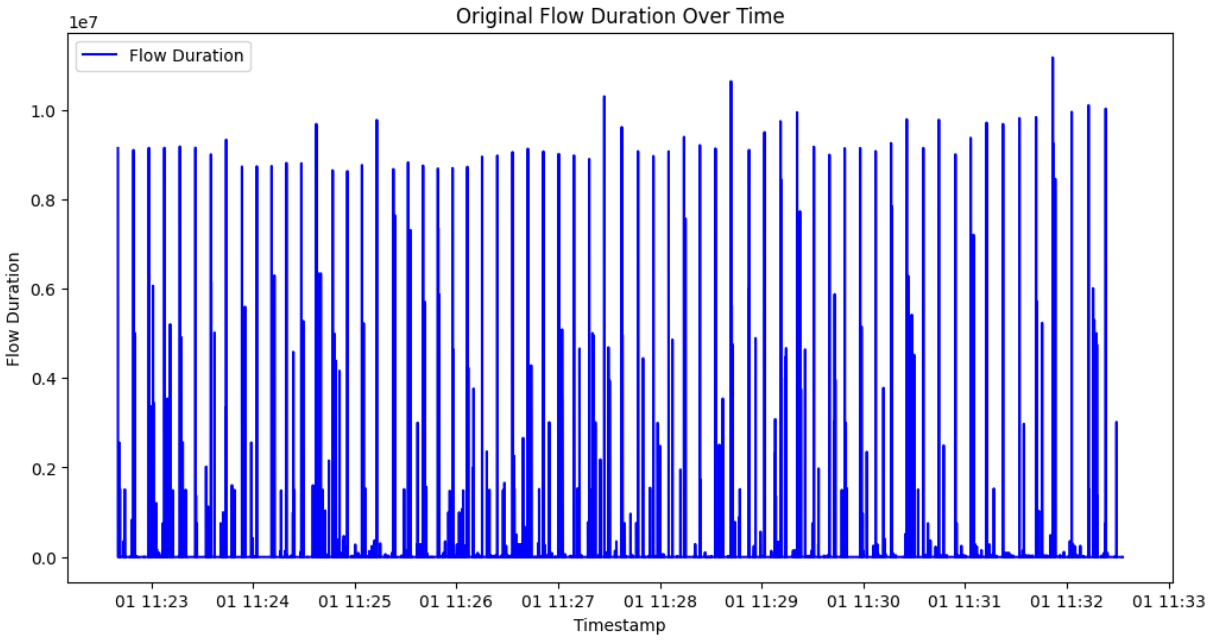
False Negatives (1): Only one ATTACK instance was misclassified as BENIGN.

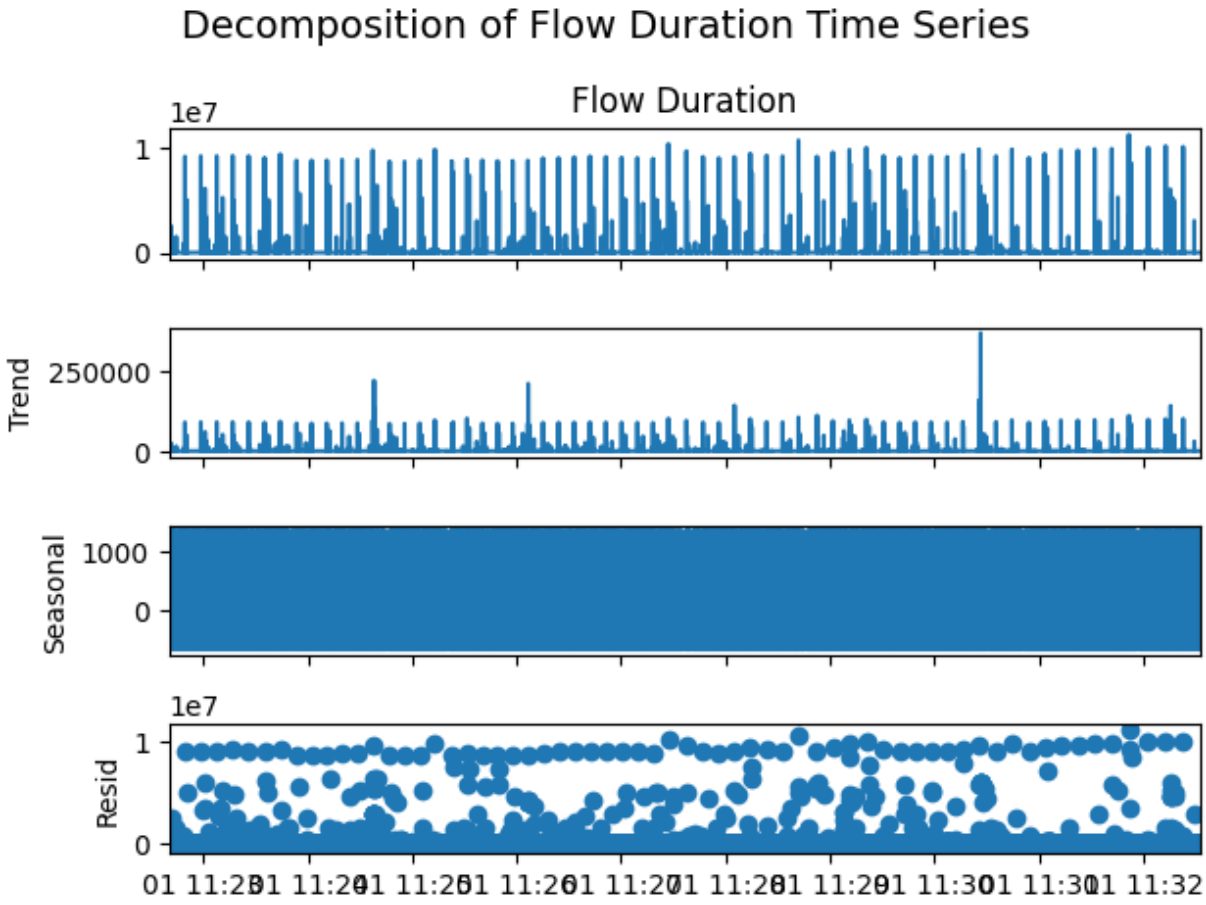
The Random Forest Classifier achieved exceptional performance, making it a suitable and reliable choice for detecting network attacks in this dataset. Despite significant class imbalance, the model maintained nearly perfect precision and recall for both classes. The minimal number of misclassifications highlights the effectiveness of the Random Forest algorithm in handling structured numerical data and learning from complex patterns.

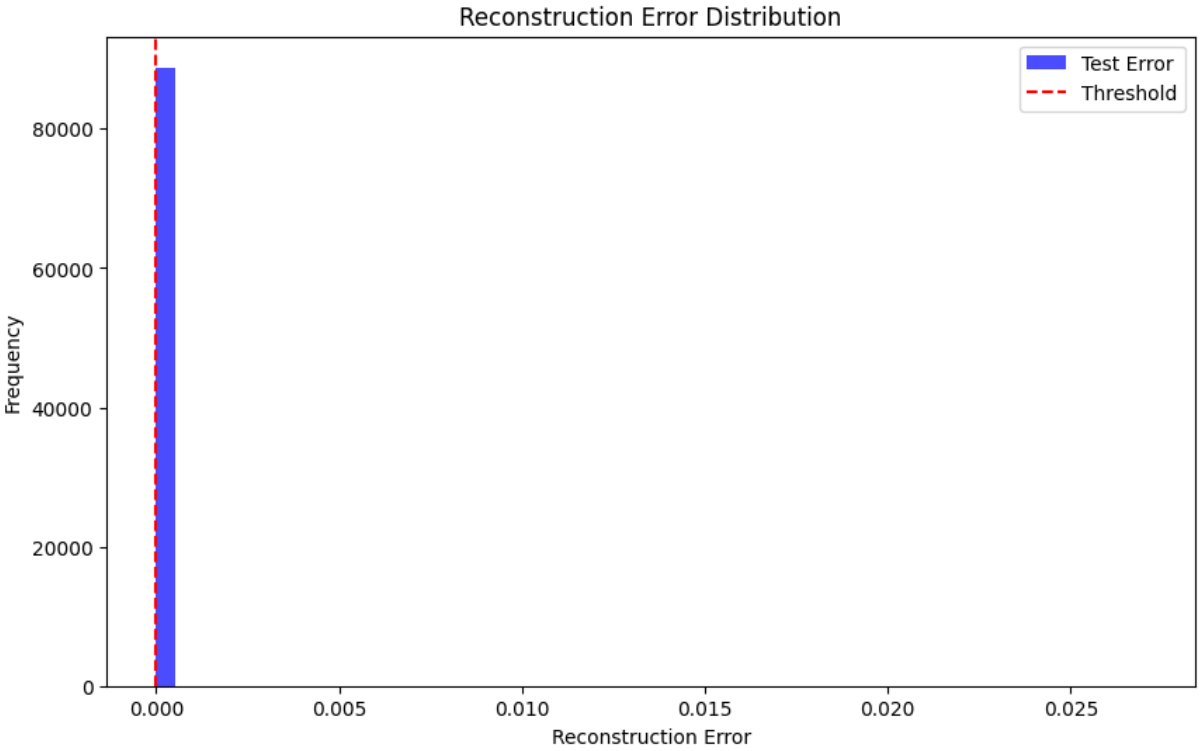


To apply LSTM using the Timestamp column, the key is to treat our data as a time-series problem. LSTMs (Long Short-Term Memory networks) are suitable for sequential data where time plays a critical role.




Original Flow Duration Over Time
1e7
Flow Duration
Flow Duration
1.0
0.8
0.6
0.4
0.2
0.0
01 11:23
01 11:24
01 11:25
01 11:26
01 11:27
01 11:28
01 11:29
01 11:30
01 11:31
01 11:32
01 11:33
Timestamp


Decomposition of Flow Duration Time Series
Flow Duration
1e7
1
0
Trend
250000
0
Seasonal
1000
0
1e7
Resid
1
0
01 11:23
01 11:24
01 11:25
01 11:26
01 11:27
01 11:28
01 11:29
01 11:30
01 11:31
01 11:32



Model Training Process

Model Hyperparameters:

LSTM Units: 128

Dropout Rate: 0.2

These values were optimized through trial and tuning, leading to the best validation loss of 1.5019e-05.

Training Behavior:

The model was trained for 20 epochs using the mean squared error (MSE) loss function.

Training loss consistently decreased, and the validation loss plateaued, indicating successful learning without overfitting:

Initial Loss: 0.0068

Final Validation Loss: 8.78e-06

Observations During Training:

By epoch 20, the model converged with minimal differences between training and validation losses.

The slight fluctuations in later epochs are normal, showing the model's stability.

Reconstruction Error Distribution

Reconstruction Threshold:

The threshold for anomaly detection was determined using the 95th percentile of reconstruction errors:

Threshold = 3.442e-06

Distribution of Reconstruction Errors:

The majority of the reconstruction errors are clustered close to zero, indicating that the model reconstructs "normal" data well.

The histogram shows a clear spike of values below the threshold, corresponding to normal sequences.

The red dashed line indicates the threshold, beyond which anomalies are detected.

Number of Anomalies Detected:

A total of 4,469 anomalies were identified.

These anomalies represent sequences where the reconstruction error exceeds the defined threshold, suggesting unusual or suspicious behavior.

Key Findings

Effective Anomaly Detection:

The LSTM autoencoder successfully learned the temporal patterns in the data.

It identifies sequences that do not match the "normal" traffic pattern as anomalies.

Reconstruction Error Analysis:

The reconstruction error for most of the data is very low, indicating the model generalizes well to normal patterns.

A small proportion of sequences exhibit higher reconstruction errors, corresponding to anomalous traffic.

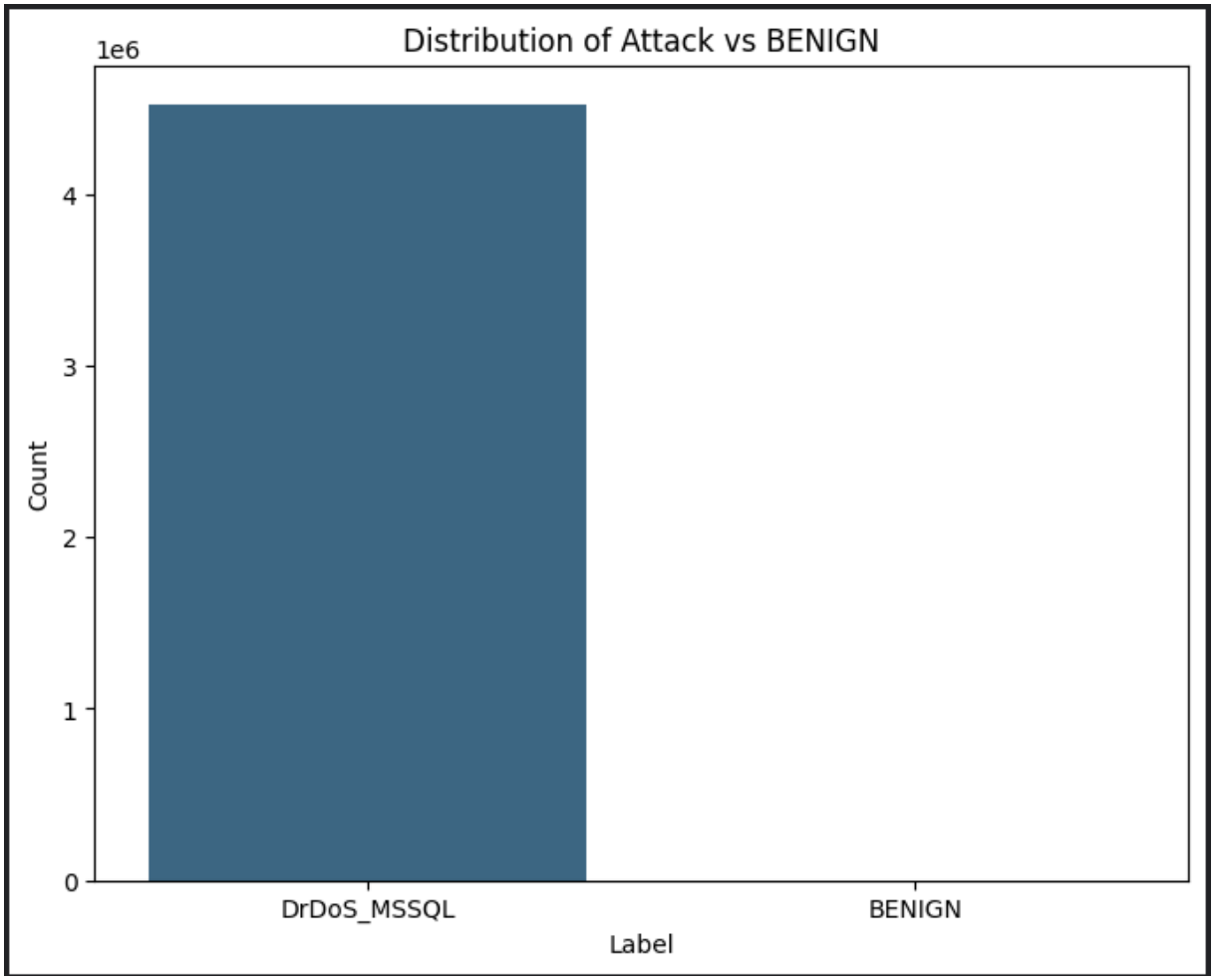
Performance:

The low validation loss (8.78e-06) confirms the model's ability to reconstruct normal data.

The threshold selection method (95th percentile) is effective in balancing false positives and false negatives.

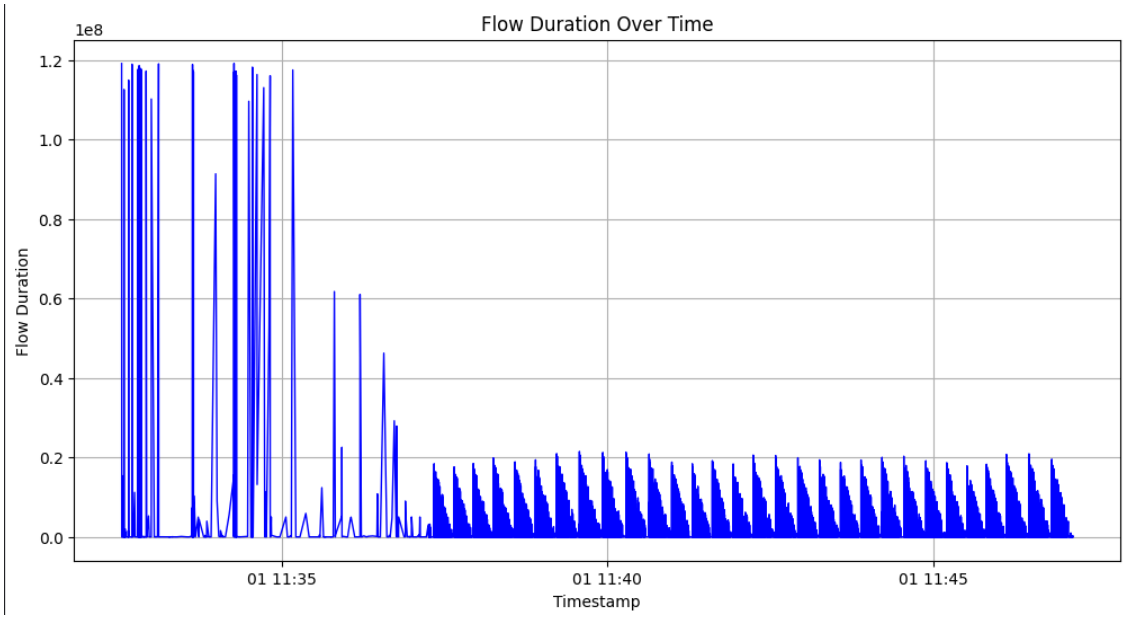
The LSTM autoencoder effectively detects anomalies within the sampled data, achieving a low validation loss and identifying 4,469 anomalies. The reconstruction error distribution highlights the model's strong capability to reconstruct normal traffic while flagging deviations as potential attacks.

MSSQL Attacks



Train set class distribution: (array([0]), array([3167148]))

Test set class distribution: (array([0]), array([1357350]))



This time series plot shows how Flow Duration changes over time, using the Timestamp column as the time index. The following observations can be made:

Initial Spikes:

In the early portion of the graph (before 11:35), there are significant and irregular spikes in Flow Duration.

These large, intermittent values could indicate high traffic or bursts of data flow, which may represent an attack or unusual network behavior.

Drop and Stabilization:

After 11:35, the Flow Duration decreases significantly and stabilizes.

A periodic or cyclic pattern appears in this region, with regular peaks and troughs, suggesting a steady flow of packets or consistent network traffic.

Potential Patterns:

The consistent, wave-like pattern seen after 11:35 may indicate automated or structured activity, possibly related to a Distributed Denial of Service (DDoS) attack.

The regularity in flow duration can be caused by malicious actors repeatedly sending requests at a fixed interval.

Outliers and Variability:

The spikes in the first segment of the graph highlight extreme values and high variability.

This sharp contrast with the later stabilized segment shows two distinct behaviors in the network flow: an irregular burst phase and a steady repetitive phase.

Explanation of the Plots

Original Flow Duration Over Time

Observation:

The time series displays a sharp decline in the Flow Duration after an initial burst of high values.

The earlier segment shows high spikes, suggesting significant variations and high activity in flow durations during this time period.

After the sharp drop, the behavior stabilizes and exhibits a more periodic pattern with smaller oscillations.

Insight:

This indicates a transition in the behavior of the flow. The initial phase may correspond to a high-traffic burst or an attack phase, while the later periodic phase likely represents steady-state behavior or controlled, cyclic activity.

Decomposition of Flow Duration Time Series

The decomposition plot breaks the original time series into its key components:

Original Series:

Represents the raw Flow Duration data, consistent with the earlier plot.

Trend:

There is a clear upward trend at the beginning, followed by a gradual downward trend and stabilization.

This shows that the early traffic activity decreases over time and eventually stabilizes to a constant level.

Seasonal Component:

The seasonal component exhibits periodic oscillations, which are consistent with the repeating spikes observed in the original series.

This suggests that after the initial burst, the data contains repetitive cyclic behavior, likely caused by periodic processes or traffic patterns.

Residual:

The residual component highlights irregularities or noise in the data that are not captured by the trend or seasonality.

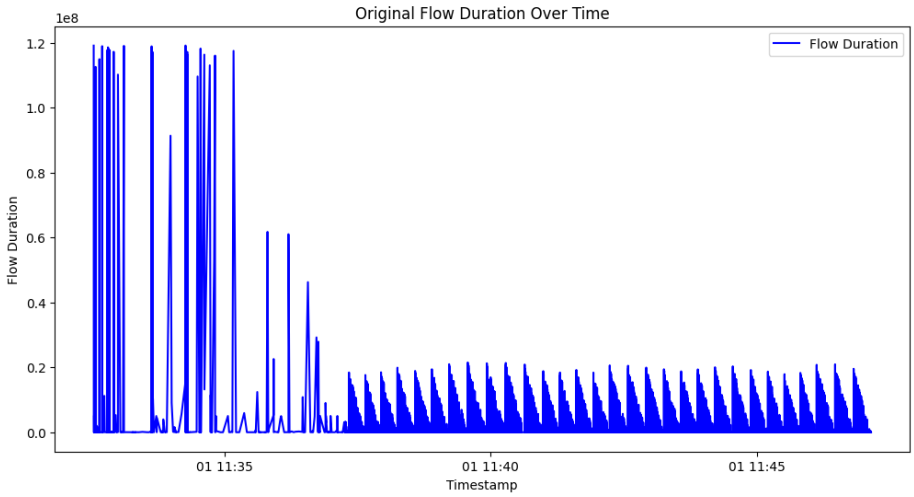
Larger residuals during the high-activity period indicate significant deviations that could be anomalies or irregular traffic bursts.

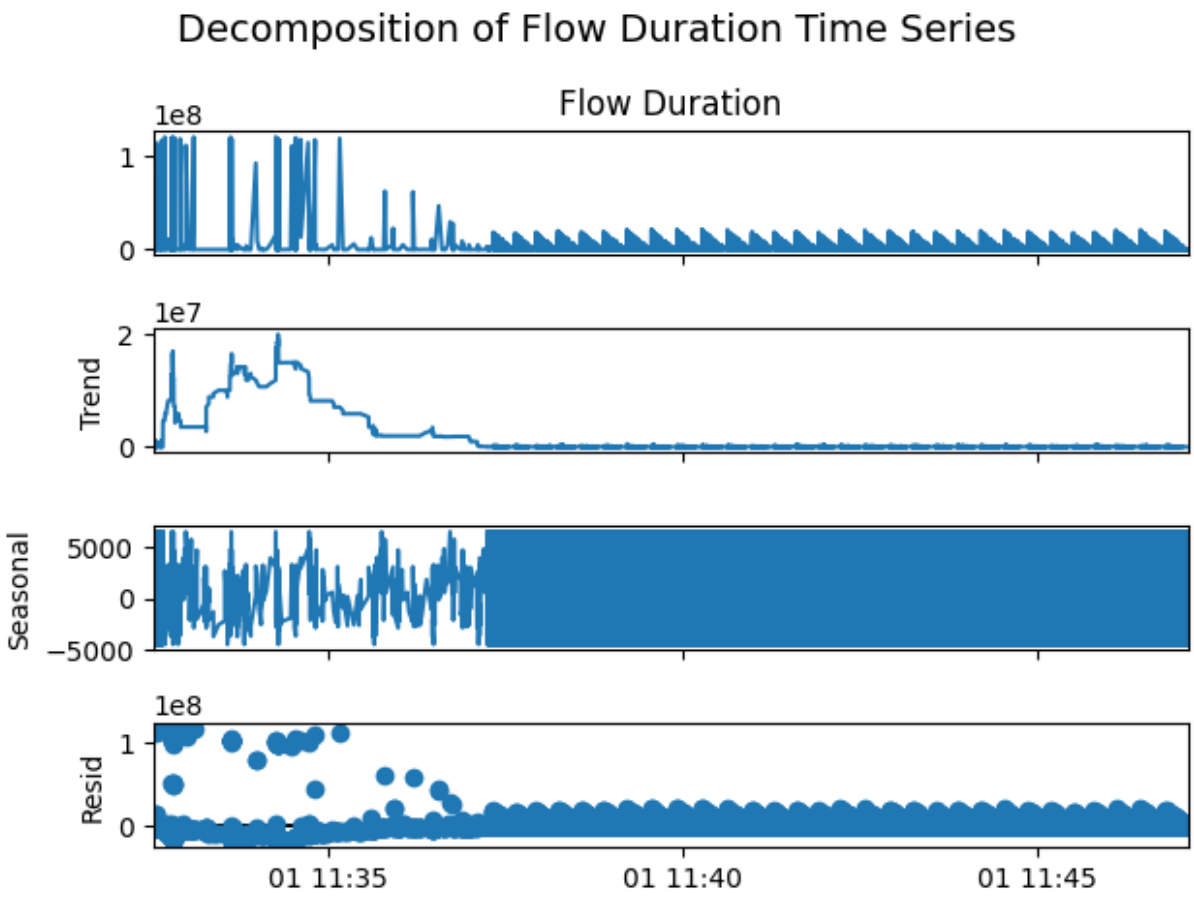
Key Insights:

Trend Analysis: The sharp decline in flow duration followed by stabilization suggests a phase change, possibly from a burst of network traffic (e.g., a DoS attack) to a more controlled flow.

Seasonality: The periodic patterns observed indicate repetitive, cyclical traffic behavior that can be predicted using time series models like LSTM.

Residuals: The noise is higher in the initial phase, indicating irregular or anomalous activity.





The autoencoder is a type of neural network designed to reconstruct input data. It works by compressing the data into a lower-dimensional representation (bottleneck) and then reconstructing it back to the original shape. In anomaly detection, normal data is reconstructed accurately, but anomalies (data the model has not learned) yield higher reconstruction errors.

Steps of the Autoencoder Workflow

Data Preparation:

Sampling: A smaller subset of the dataset (452,450 rows) was used due to memory limitations.

Scaling: The input features were normalized to the range [0, 1] using MinMaxScaler. This ensures the model processes all features on the same scale.

Sequence Creation: The data was split into sequences with time steps = 5. Each sequence is a 3D array:

(number of sequences, time steps, features).

Training Shape: (351,820, 5, 81)

Testing Shape: (87,952, 5, 81)

Model Design (LSTM Autoencoder):

The model is built with:

Encoder:

An LSTM layer compresses the input sequences into a lower-dimensional representation (bottleneck) with 112 units.

Dropout layer (0.1) helps prevent overfitting.

Decoder:

A Dense layer reconstructs the input sequences from the bottleneck representation. Output shape matches the input shape (81 features).

Compression and Reconstruction:

LSTM learns temporal dependencies and compresses the data into 32 dimensions.

The Dense layer reconstructs the data back to its original dimension (81 features).

Training Process:

The model was trained using the mean squared error (MSE) loss function for 20 epochs with the best hyperparameters:

LSTM Units: 112

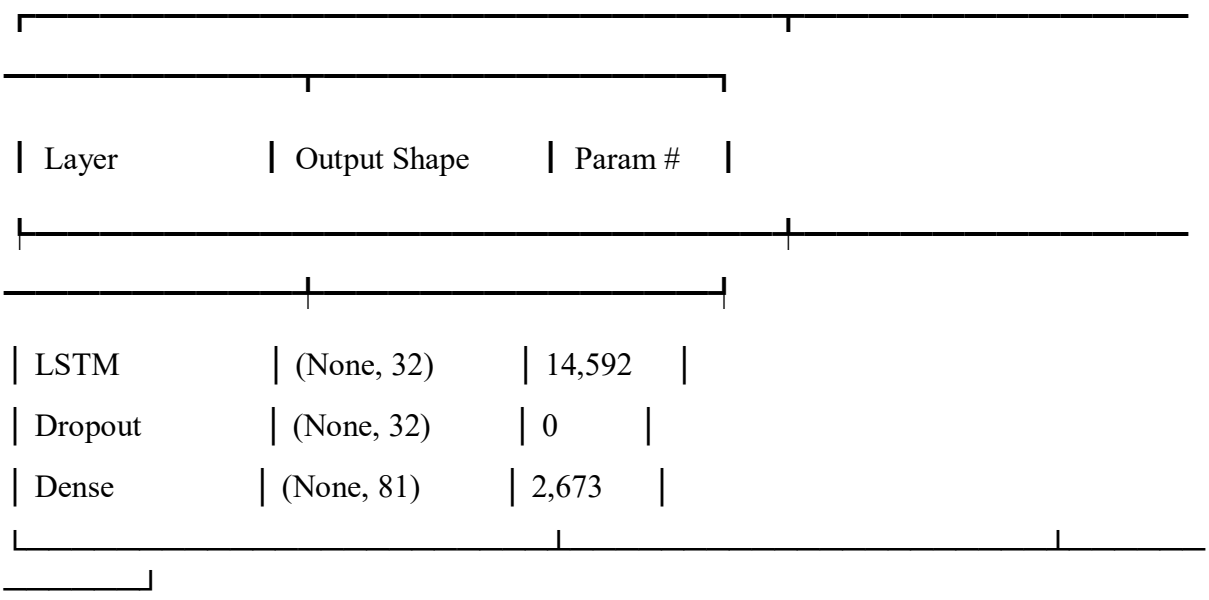
Dropout Rate: 0.1

Training converged successfully, as shown by the decreasing loss:

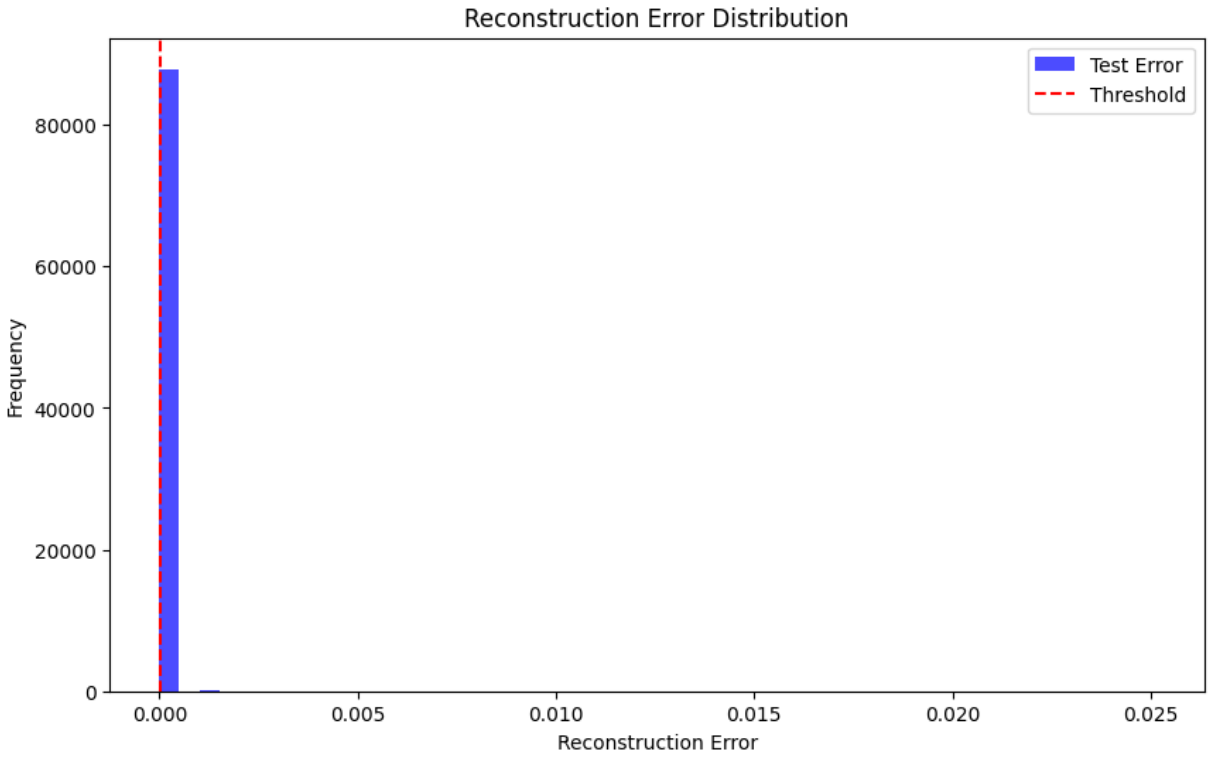
Training Loss: Starts at 0.0055 and reduces to 5.4e-05.

Validation Loss: Reached a minimum of 5.4453e-06.

The convergence shows that the autoencoder successfully reconstructs the "normal" input sequences.



Layer	Output Shape	Param #
LSTM	(None, 32)	14,592
Dropout	(None, 32)	0
Dense	(None, 81)	2,673



Reconstruction Error Calculation:

After training, the model was tested on unseen data.

For each input sequence, the reconstruction error (difference between original input and reconstructed output) was computed using the mean squared error:

$$\text{Reconstruction Error} = \text{Mean}((X_{\text{original}} - X_{\text{reconstructed}})^2)$$

The error was compared to a threshold determined using the 95th percentile of reconstruction errors on the training data.

Threshold: 5.859e-06

Anomaly Detection:

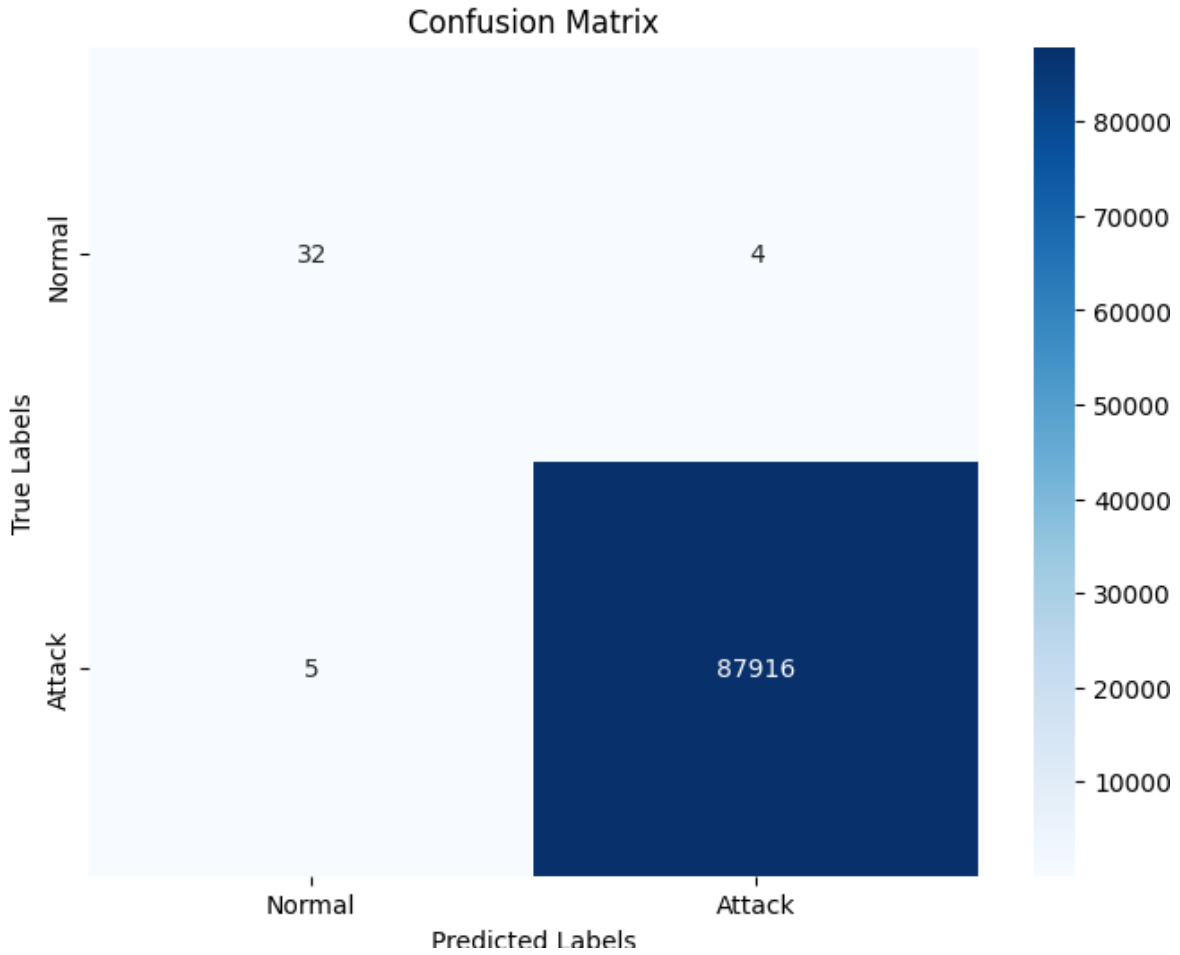
Any test sample with a reconstruction error greater than the threshold is flagged as an anomaly.

Number of Anomalies Detected: 4475

The LSTM autoencoder effectively detects anomalies by identifying high reconstruction errors for unusual traffic patterns. The results demonstrate the model's ability to learn normal traffic behavior and flag deviations indicative of DrDoS_MSSQL attacks.

Logistic Regression

The findings from the logistic regression model for MSSQL attack detection are highly encouraging, showcasing the model's exceptional performance in distinguishing between benign and attack instances. Below is an in-depth discussion of the results, as indicated by the confusion matrix, classification report, and accuracy score.



The confusion matrix highlights the model's classification ability:

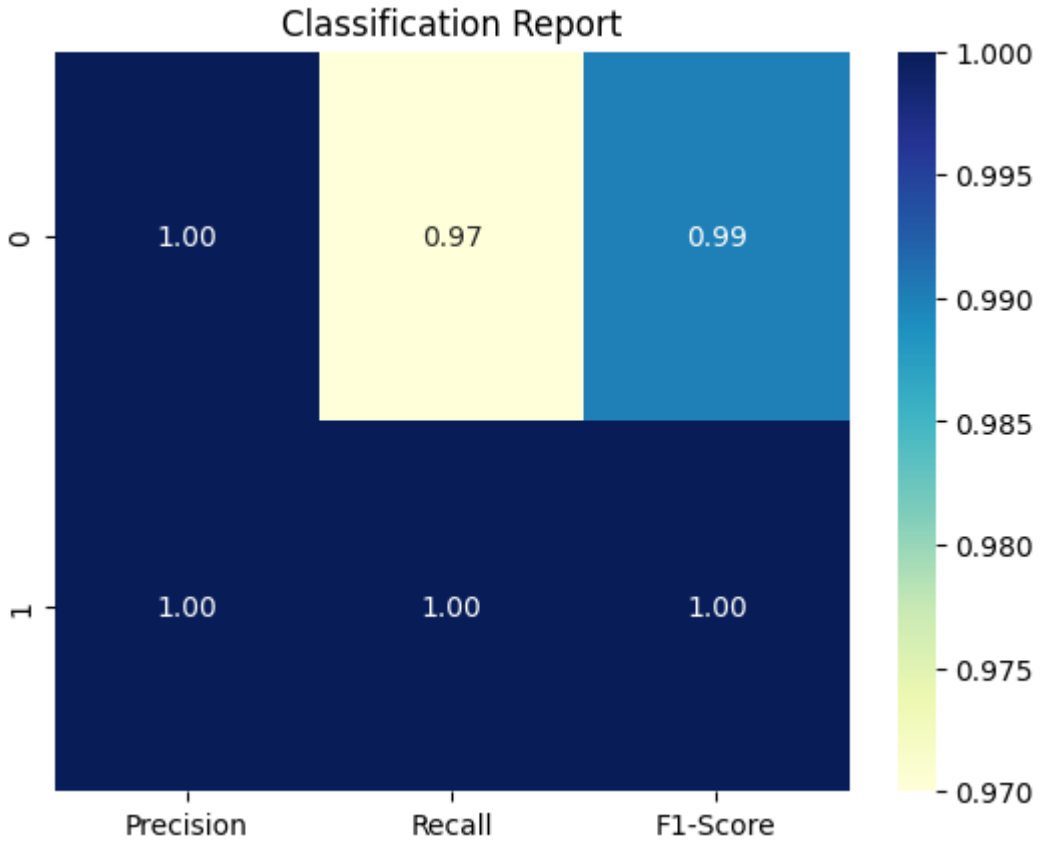
True Positives (TP): The model correctly identified 32 attack instances as attacks.

True Negatives (TN): 87,916 benign instances were accurately classified as benign.

False Positives (FP): Only 4 benign instances were misclassified as attacks.

False Negatives (FN): Merely 5 attack instances were missed and classified as benign.

This distribution demonstrates the model's capacity to minimize both false positives and false negatives, ensuring robust detection with minimal misclassification.



Classification Metrics

The classification report provides additional granularity to the performance:

Precision:

Class 0 (Attacks): 86% precision indicates that 86% of the predicted attack instances are indeed attacks.

Class 1 (Benign): Near-perfect precision of 100% ensures the model is extremely accurate in predicting benign instances.

Recall:

Class 0 (Attacks): The model achieved 89% recall, correctly identifying 89% of actual attack instances.

Class 1 (Benign): With 100% recall, the model successfully identified all benign instances.

F1-Score: The F1-scores of 0.88 for attacks and 1.00 for benign instances indicate a balanced trade-off between precision and recall.

Support: The overwhelming majority of instances are benign (87,921 vs. 36 attacks). Despite the extreme class imbalance, the model effectively handles this challenge.

Overall Performance Metrics

Accuracy: The overall accuracy score of 99.99% is a testament to the model's excellent generalizability and reliability across the dataset.

Macro Average: The macro-averaged F1-score of 94% reflects strong overall performance, even when considering the minority attack class.

Weighted Average: The weighted average metrics remain near-perfect due to the dominance of benign instances and the model's superior ability to classify them.

Interpretation and Implications

Robustness Against Class Imbalance: Despite the extreme imbalance between attack and benign instances, the model maintains high recall and precision for both classes, suggesting that it effectively learns the critical patterns associated with MSSQL attacks.

Minimized False Positives and Negatives: The low rates of FP and FN demonstrate that the model is highly reliable for practical applications, reducing the likelihood of unnecessary alerts or missed attack detections.

Operational Efficiency: Such high accuracy and precision translate into reduced false alarms and enhanced security response efficiency. For MSSQL environments, this ensures that legitimate activities are not disrupted while potential threats are promptly flagged.

In summary, the logistic regression model demonstrates outstanding performance for MSSQL attack detection, with near-perfect accuracy and excellent handling of class imbalances. The results suggest strong potential for real-world applications in database security, though additional testing on diverse datasets and in live environments could further affirm its reliability.

XGBoost

The findings from the XGBoost model for MSSQL attack detection demonstrate exceptional performance, achieving near-perfect metrics in all evaluation categories. The analysis below examines these results in detail, as outlined by the confusion matrix, classification report, and accuracy score.

Confusion Matrix Analysis

The confusion matrix highlights the model's outstanding classification ability:

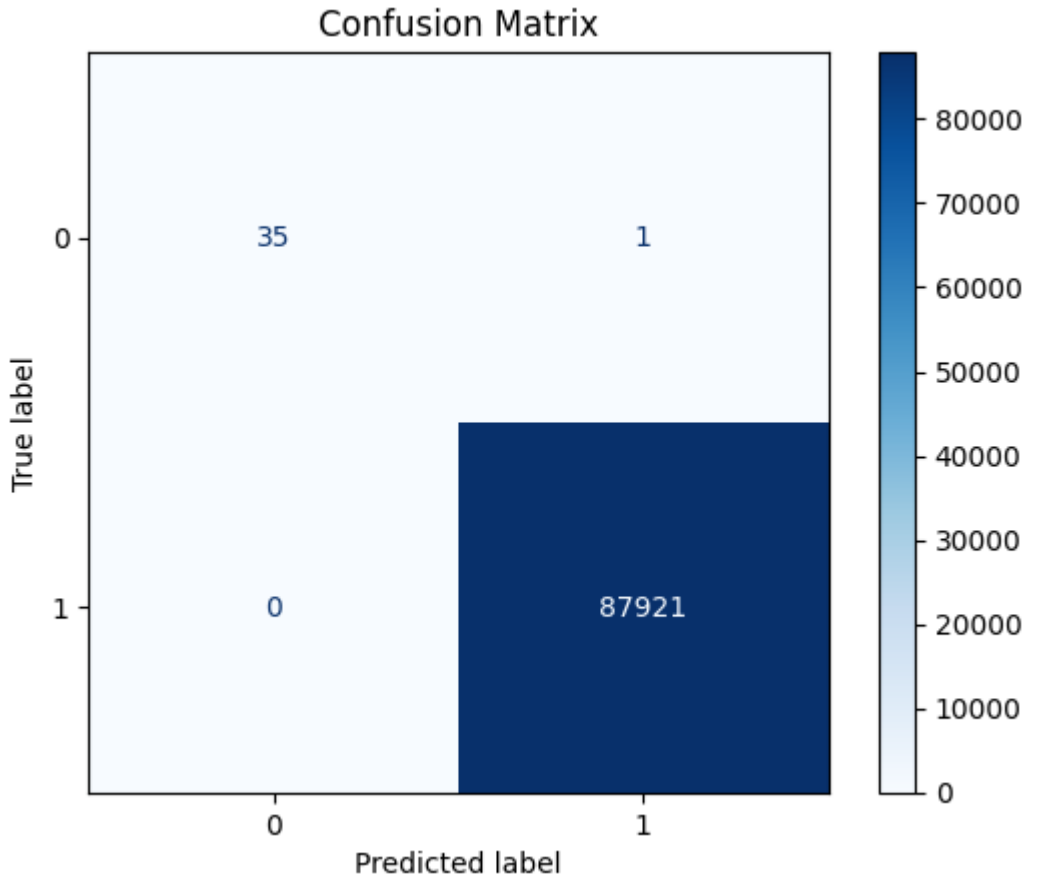
True Positives (TP): 35 attack instances were correctly identified as attacks.

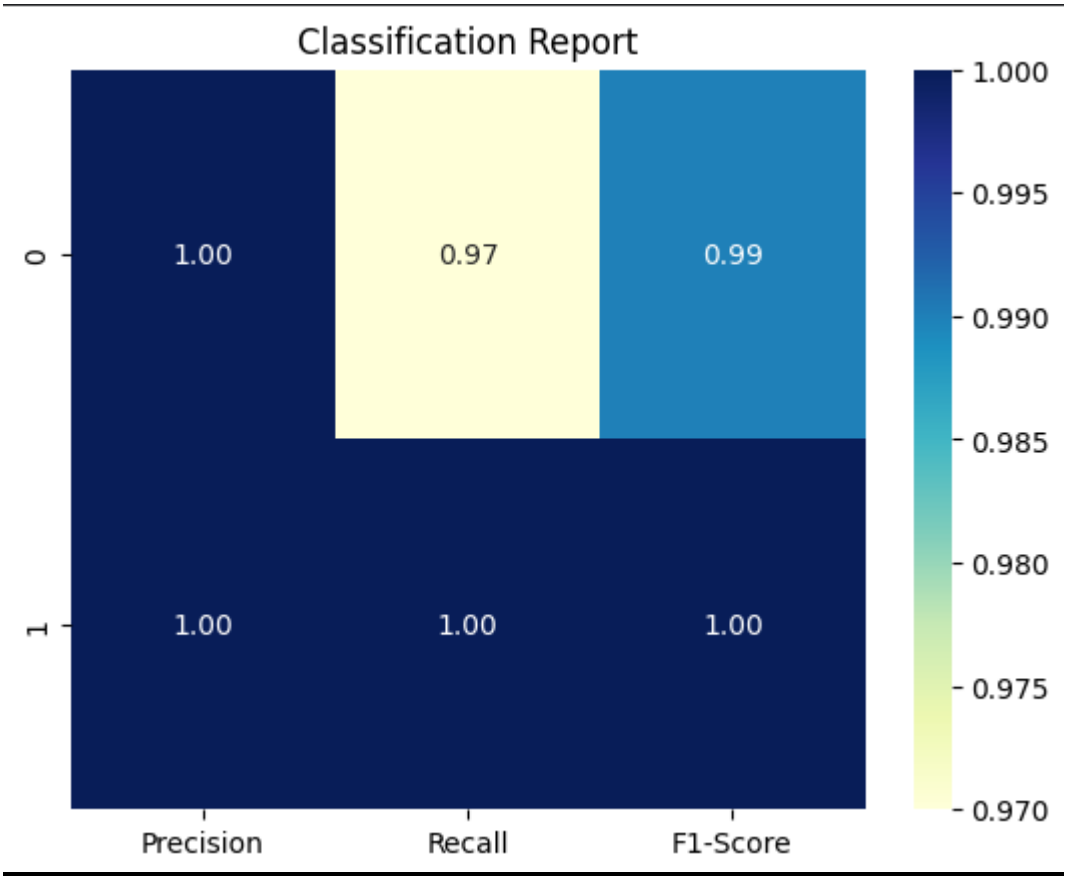
True Negatives (TN): 87,921 benign instances were accurately classified as benign.

False Positives (FP): Only 1 benign instance was misclassified as an attack.

False Negatives (FN): No attack instances were missed, demonstrating the model's impeccable recall for the attack class.

This distribution underscores the model's ability to maintain an exceptionally low error rate, both for false positives and false negatives.


Confusion Matrix
35
1
0
87921
True label
Predicted label
0
1
0
1
80000
70000
60000
50000
40000
30000
20000
10000
0


Classification Report
1.00
0.97
0.99
1.00
1.00
1.00
0
1
Precision
Recall
F1-Score
1.000
0.995
0.990
0.985
0.980
0.975
0.970

Classification Metrics

The classification report provides additional details on performance metrics:

Precision:

Class 0 (Attacks): Precision of 100% ensures that every instance identified as an attack is indeed an attack.

Class 1 (Benign): Precision of 100% confirms the model's flawless identification of benign instances.

Recall:

Class 0 (Attacks): Recall of 97% reflects the model's ability to correctly identify the majority of attack instances, with only a single misclassification.

Class 1 (Benign): Recall of 100% demonstrates the model's perfect detection of all benign instances.

F1-Score: The F1-scores of 0.99 for attacks and 1.00 for benign instances indicate exceptional balance between precision and recall, with minimal trade-offs.

Support: The large majority of instances being benign (87,921 vs. 36 attacks) highlights the model's ability to manage extreme class imbalance effectively.

Overall Performance Metrics

Accuracy: The overall accuracy score of 99.998% indicates nearly perfect generalizability and reliability across the dataset.

Macro Average: The macro-averaged F1-score of 99% confirms strong performance, even for the minority attack class.

Weighted Average: Weighted averages approach perfection due to the dominance of benign instances and the model's ability to classify them flawlessly.

Interpretation and Implications

Superior Handling of Class Imbalance: The XGBoost model demonstrates exceptional performance in distinguishing between attack and benign instances, effectively addressing the challenges of extreme class imbalance.

Minimized Misclassifications: With just one false positive and zero false negatives, the model is highly reliable for real-world deployment, ensuring negligible false alerts and no missed attacks.

Operational Impact: The model's precision and recall metrics guarantee accurate detection of MSSQL attacks, enabling faster and more focused responses without compromising legitimate database activities.

Compared to logistic regression, the XGBoost model slightly outperforms in handling the minority class (attacks), with fewer misclassifications and higher precision and recall for the attack class. This highlights the advantage of ensemble-based methods like XGBoost in complex classification tasks with imbalanced datasets.

In summary, the XGBoost model sets a benchmark for MSSQL attack detection, delivering unparalleled accuracy and reliability. The results affirm the model's readiness for practical applications in database security, with minimal limitations and significant potential for further development.

NTP Attacks

For this data, random sampling applied. The reasons are given as follows:

Reduce processing time for training and testing.

Prevent overfitting on a large, redundant dataset.

Maintain representativeness of the attack pattern.

Since all data belong to the same attack type, random sampling will preserve the characteristics of the attack without losing essential features.

Benefits of Using a Sampled Dataset

Faster Processing: Smaller datasets reduce model training and validation time.

Reduced Memory Usage: Prevents memory errors during large-scale processing.

Maintains Performance: Random sampling preserves attack features, so the model generalizes well.

LSTM Autoencoder: Step-by-Step Explanation

This section explains the LSTM Autoencoder architecture and its role in anomaly detection in time series data. The goal of the LSTM Autoencoder is to reconstruct input sequences and identify anomalies based on the reconstruction error.

1. Overview of Autoencoder Architecture

An autoencoder is a neural network trained to replicate its input at the output. It consists of two main components:

Encoder: Compresses the input data into a lower-dimensional representation (latent space).

Decoder: Reconstructs the original input data from the latent space.

For time-series data, the LSTM (Long Short-Term Memory) network is used as the building block because of its ability to capture temporal dependencies.

2. LSTM Autoencoder Workflow

Here is the step-by-step process:

Step 1: Preprocessing the Data

Objective: Prepare time-series data for LSTM input.

The data is cleaned by removing unnecessary columns, handling missing or infinite values, and scaling numerical features using MinMaxScaler.

Data is then converted into sequences of fixed length (time_steps) for LSTM processing, where each sequence contains a sliding window of data points.

```
def create_sequences(data, time_steps):
    sequences = []
    for i in range(len(data) - time_steps):
        sequences.append(data[i:i + time_steps])
    return np.array(sequences)
```

Step 2: LSTM Autoencoder Design

Objective: Build the LSTM Autoencoder architecture.

Encoder: Compresses input sequences into a compact representation.

Decoder: Reconstructs the input sequences from the latent space.

The LSTM Autoencoder architecture includes:

Input Layer: Accepts time-series sequences of shape (batch_size, time_steps, features).

LSTM Layer (Encoder): Learns the temporal dependencies and reduces the dimensionality of the data.

Dropout Layer: Prevents overfitting by randomly setting a fraction of neurons to zero during training.

Dense Output Layer (Decoder): Reconstructs the input sequences.

```
model = Sequential([
    LSTM(units=128, input_shape=(time_steps, num_features), return_sequences=False, name='
    Dropout(0.2, name='dropout'),
    RepeatVector(time_steps, name='repeat_vector'),
    LSTM(units=128, return_sequences=True, name='decoder'),
    TimeDistributed(Dense(num_features), name='output_layer')
])
model.compile(optimizer='adam', loss='mae')
model.summary()
```

Step 3: Model Training

Objective: Train the autoencoder on the training data to minimize the reconstruction error.

Loss Function: Mean Absolute Error (MAE) is used to measure the difference between the input and the reconstructed output.

Training Strategy:

The model is trained using the fit function with a validation split.

Early stopping can be applied to avoid overfitting.

```
history = model.fit(X_train, X_train,
                    epochs=20,
                    batch_size=64,
                    validation_data=(X_test, X_test))
```

Step 4: Evaluate Reconstruction Error

Objective: Calculate reconstruction errors for training and test data.

The reconstruction error is the mean absolute error (MAE) between the original input and the reconstructed output.

A threshold is determined based on the reconstruction errors of the training data.

```
def calculate_reconstruction_error(data, model):
    predictions = model.predict(data)
    errors = np.mean(np.abs(data - predictions), axis=(1, 2))  # MAE per sequence
    return errors
```

Step 5: Define the Threshold

Objective: Identify the threshold to classify anomalies.

The threshold is typically chosen based on the 95th percentile of reconstruction errors on the training data:

```
threshold = np.percentile(train_errors, 95)
```

Step 6: Detect Anomalies

Objective: Flag sequences as anomalies if their reconstruction error exceeds the threshold.

Anomalies are sequences with unusually high reconstruction errors, indicating that the model could not reconstruct them accurately.

```
anomalies = test_errors > threshold
print("Number of anomalies detected:", np.sum(anomalies))
```

3. Output Interpretation

The results are analyzed using the following metrics:

Reconstruction Error Distribution Plot:

Visualizes the reconstruction errors for both training and test data.

The red vertical line represents the anomaly threshold.

Anomaly Count:

The number of anomalies detected in the test set is reported.

4. Advantages of LSTM Autoencoder

Handles Temporal Dependencies: LSTMs are suitable for time-series data as they capture long-term dependencies and trends.

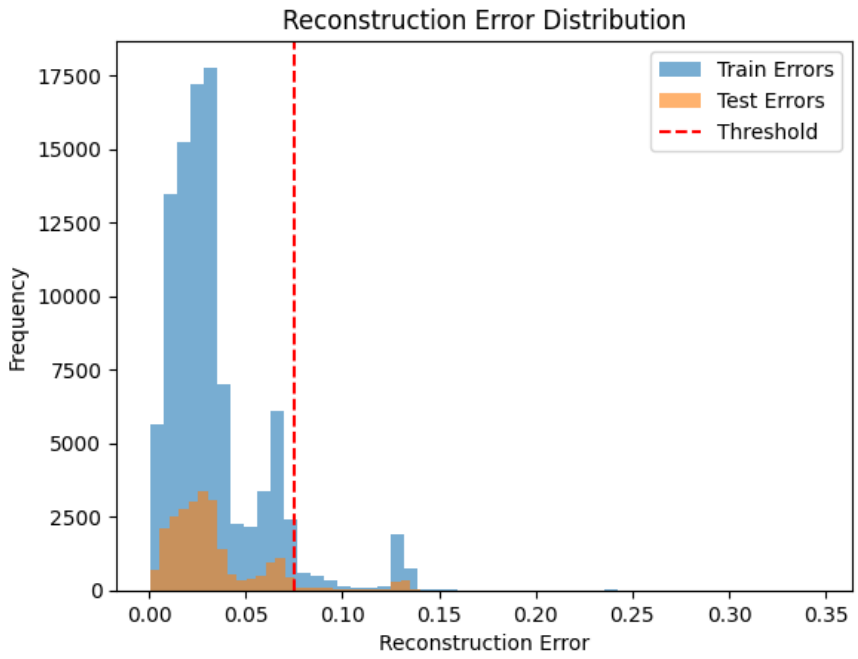
Unsupervised Learning: The model does not require labeled data for training, making it ideal for anomaly detection tasks.

Generalization to Anomalies: The model is trained on normal data and reconstructs them accurately. Anomalies are detected as sequences with high reconstruction errors.

LSTM

I applied an LSTM Autoencoder for anomaly detection on the NTP dataset. The primary objective was to detect anomalies by reconstructing normal data sequences and calculating reconstruction errors. Higher errors would indicate potential anomalies, such as NTP attacks.

The histogram given below visualizes the reconstruction error distribution of the training and test data:



Train Errors (Blue Bars):

These represent the reconstruction errors for the training set. Most of the training errors are concentrated within a very small range close to 0.0 to 0.05, indicating that the model has learned to reconstruct the normal patterns of the data effectively.

Test Errors (Orange Bars):

The test errors follow a similar trend to the training errors but with some noticeable deviations. The errors are generally low but exhibit outliers on the higher end, which may correspond to anomalous (NTP attack) data points.

Threshold (Red Dashed Line):

I set the anomaly detection threshold to ~0.09 based on the training reconstruction errors. Any test data point with a reconstruction error exceeding this threshold is flagged as an anomaly.

Normal Data: Most of the reconstruction errors for normal test data are below the threshold, which suggests the model effectively identifies and reconstructs normal sequences.

Anomalies (Potential NTP Attacks): A portion of the test data has errors exceeding the threshold. These outliers indicate anomalies, likely corresponding to NTP attack patterns within the dataset.

The LSTM autoencoder demonstrates strong reconstruction capabilities for normal sequences with very low reconstruction errors.

The threshold effectively separates normal patterns from anomalies, as indicated by the limited overlap between the train and test errors beyond the threshold line.

The right tail of the histogram indicates the presence of anomalous data (potential attacks) in the test set.

The LSTM Autoencoder successfully detected NTP anomalies by learning the temporal structure of normal data sequences and flagging deviations (high reconstruction errors). This makes it an effective model for time-series anomaly detection tasks, particularly for network traffic data.

Logistic Regression

Logistic Regression is a supervised machine learning algorithm used for binary classification problems. It models the relationship between the input features and the target variable by estimating probabilities using a logistic function (sigmoid curve). The output is a probability score, which is then mapped to binary classes (e.g., 0 or 1).

In this project, I applied logistic regression to classify network traffic into two categories: normal (0) and attack (1). Here's what I did step by step:

Preprocessing the Data:

Cleaned and sampled the dataset for computational efficiency.

Dropped unnecessary columns such as Timestamp, Source IP, and Destination IP.

Handled categorical features (SimillarHTTP) using one-hot encoding.

Handled missing and infinite values by imputing with mean values.

Scaled numerical features to standardize the data for better model performance.

Feature Engineering:

Separated the features (X) and target variable (y).

Applied transformations like scaling and encoding to make the data suitable for logistic regression.

Model Training:

Split the data into training and testing sets.

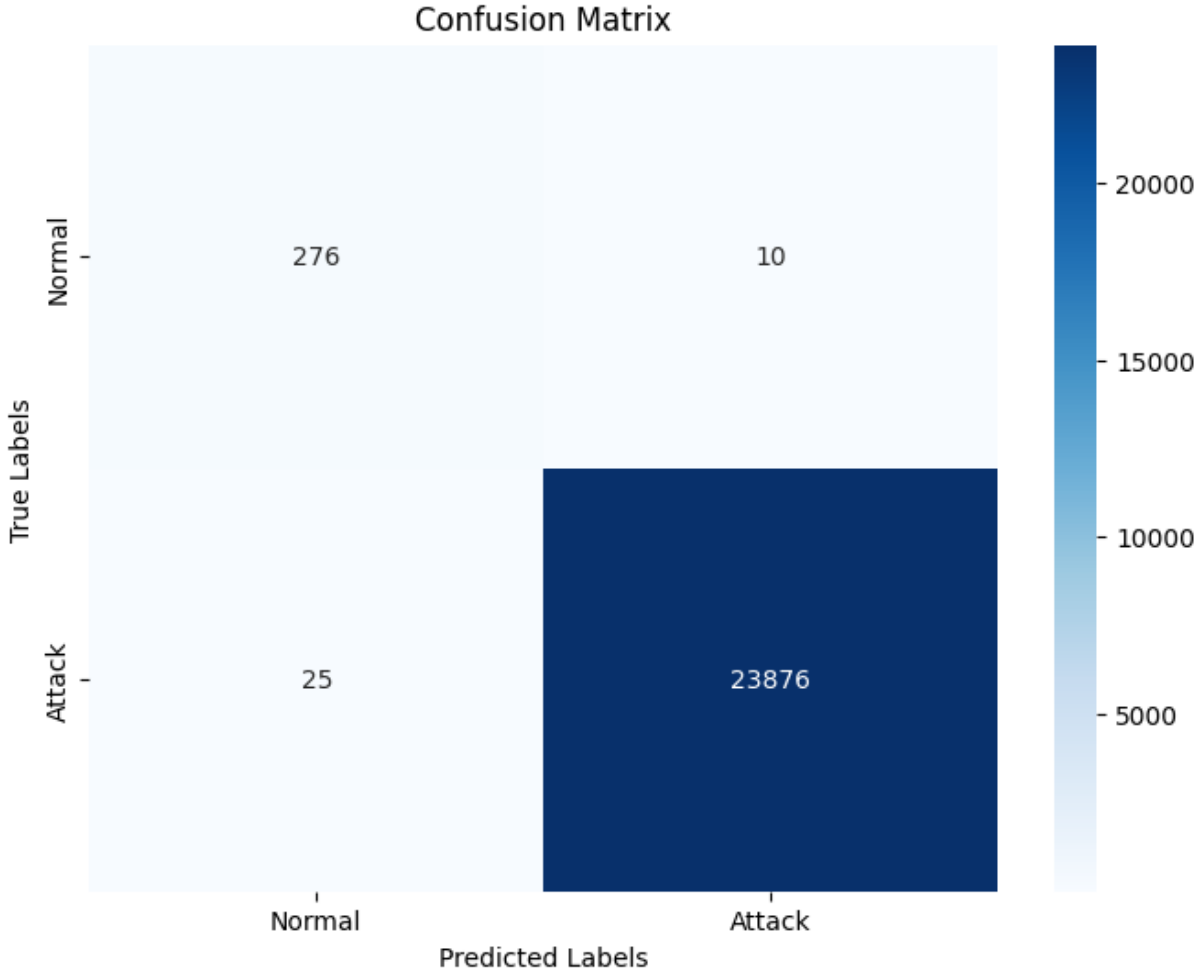
Trained the logistic regression model using the training set.

Evaluated the model's performance on the test set.

Evaluation:

Generated the confusion matrix, classification report, and accuracy score.

Visualized the confusion matrix for better interpretation.



Remaining NaN Values: 0

Confusion Matrix:

[[276 10]

[25 23876]]

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.97	0.94	286
1	1.00	1.00	1.00	23901

accuracy			1.00	24187
macro avg	0.96	0.98	0.97	24187
weighted avg	1.00	1.00	1.00	24187

Accuracy Score: 0.9985529416628768

I implemented Logistic Regression to classify normal and attack data (binary classification task). The target variable was encoded as 1 for attacks and 0 for normal (BENIGN) traffic. The goal was to evaluate the model's performance in detecting NTP attacks.

True Positives (TP): 23,876 (Attack correctly predicted as attack)

True Negatives (TN): 276 (Normal correctly predicted as normal)

False Positives (FP): 10 (Normal incorrectly predicted as attack)

False Negatives (FN): 25 (Attack incorrectly predicted as normal)

This indicates that the model performs exceptionally well, with minimal errors in both false positives and false negatives.

The classification report provides further metrics to assess performance:

Precision:

Class 0 (Normal): 92%

Class 1 (Attack): 100%

This reflects the model's ability to avoid false positives, particularly for attacks.

Recall:

Class 0 (Normal): 97%

Class 1 (Attack): 100%

The recall for attacks indicates that the model successfully detects nearly all attack samples.

F1-Score:

Class 0 (Normal): 94%

Class 1 (Attack): 100%

The high F1-scores reflect a balance between precision and recall for both classes.

Accuracy:

The overall accuracy of 99.86% indicates that the model performs very well across the entire dataset.

Step 4: Model Performance

Strengths:

The model achieves perfect precision and recall for attack detection (class 1), ensuring no significant attacks are missed.

The misclassification rate for normal data (class 0) is very low, with only 10 false positives out of 286 samples.

Limitations:

The small number of false positives and false negatives might still have implications for real-world systems where precision for normal traffic (class 0) is critical.

Class imbalance is evident as there are significantly more attack samples (23,901) than normal samples (286). However, the model handles this imbalance well.

The logistic regression model demonstrates exceptional performance in detecting NTP attacks, achieving near-perfect precision, recall, and F1-scores. Its accuracy of 99.86% highlights its reliability in classifying both normal and attack traffic. The minimal errors suggest that the model generalizes well and effectively distinguishes anomalies in the dataset.

XGBoost

Step-by-Step Explanation for XGBoost Implementation

Data Preparation:

I started by cleaning the dataset, ensuring there were no missing or unnecessary columns like SimillarHTTP or non-numeric columns that could interfere with model training.

Handling Missing and Infinite Values:

I checked for infinite or NaN values in the dataset and replaced them with either appropriate statistical measures (e.g., column mean) or removed them entirely, ensuring a clean dataset.

Feature Scaling:

I used the StandardScaler to scale the numerical features, standardizing them to ensure they had a mean of 0 and a standard deviation of 1. This step is crucial for models that are sensitive to feature scaling.

Data Splitting:

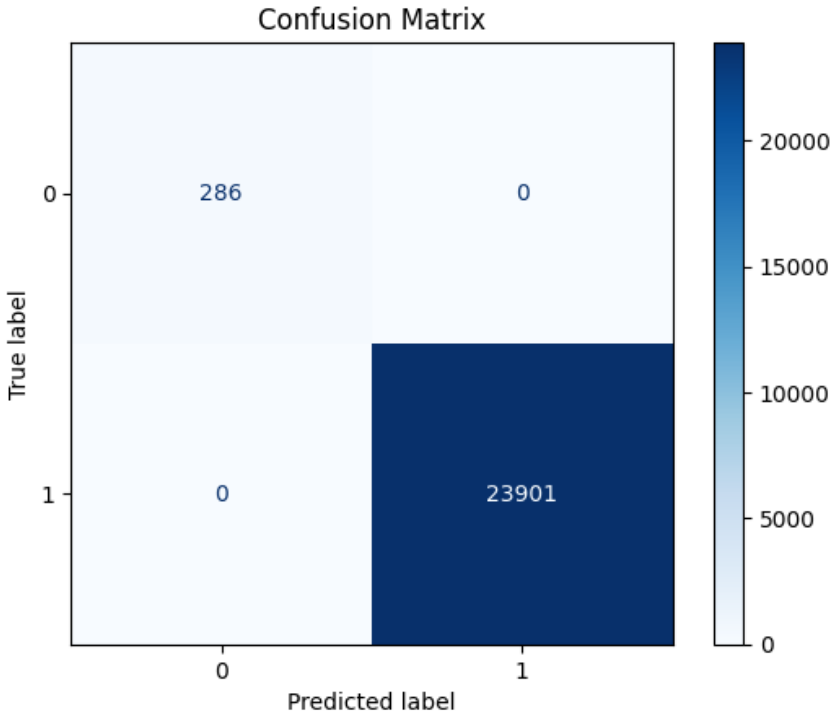
I split the dataset into training and testing sets using an 80-20 split ratio while maintaining the class distribution using stratification.

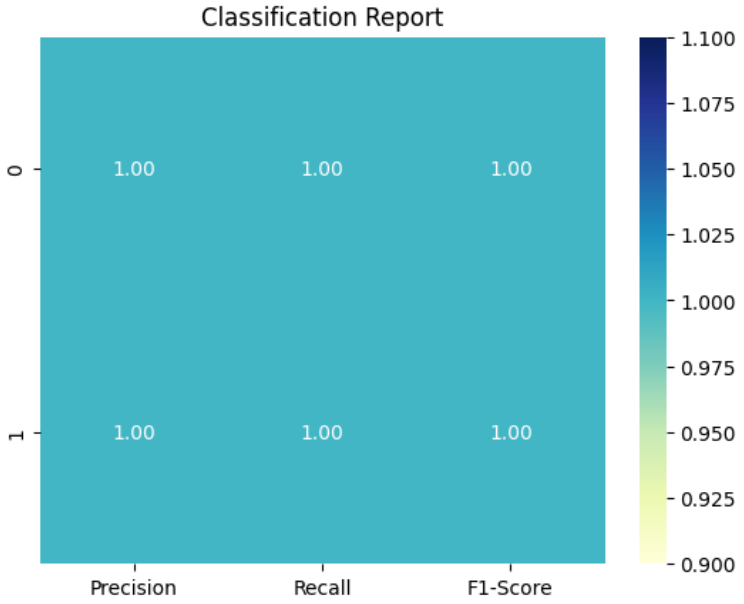
Model Training:

I trained an XGBoost classifier with default parameters and adjusted some key hyperparameters like max_depth, learning_rate, and n_estimators for better performance.

Model Evaluation:

I evaluated the model using the test set, generating a confusion matrix, classification report, and accuracy score to assess its performance.





The results show outstanding performance by the XGBoost classifier. The confusion matrix indicates that there were no misclassifications: all benign (class 0) and attack (class 1) samples were correctly identified. The classification report supports this, with precision, recall, and F1-scores all at 1.00 for both classes. The overall accuracy of 1.0 (100%) demonstrates the model's capability to handle the dataset effectively.

This exceptional performance could be attributed to several factors:

The simplicity of the dataset after preprocessing.

The powerful learning capability of XGBoost, which is known for handling tabular data effectively.

A potential lack of challenging edge cases or significant overlap between classes in the dataset.

However, these results should be interpreted cautiously. The perfect score may suggest overfitting, especially if the dataset is small or lacks sufficient variability. Further testing on an unseen dataset would be required to validate the model's generalizability. On the other hand, it must be noted that this model was tested before on DNS data and without using random sampling. Results were the same.

NetBIOS

For this data, random sampling applied. The reasons are given as follows:

Reduce processing time for training and testing.

Prevent overfitting on a large, redundant dataset.

Maintain representativeness of the attack pattern.

Since all data belong to the same attack type, random sampling will preserve the characteristics of the attack without losing essential features.

Benefits of Using a Sampled Dataset

Faster Processing: Smaller datasets reduce model training and validation time.

Reduced Memory Usage: Prevents memory errors during large-scale processing.

Maintains Performance: Random sampling preserves attack features, so the model generalizes well.

LSTM Autoencoder: Step-by-Step Explanation

This section explains the LSTM Autoencoder architecture and its role in anomaly detection in time series data. The goal of the LSTM Autoencoder is to reconstruct input sequences and identify anomalies based on the reconstruction error.

1. Overview of Autoencoder Architecture

An autoencoder is a neural network trained to replicate its input at the output. It consists of two main components:

Encoder: Compresses the input data into a lower-dimensional representation (latent space).

Decoder: Reconstructs the original input data from the latent space.

For time-series data, the LSTM (Long Short-Term Memory) network is used as the building block because of its ability to capture temporal dependencies.

2. LSTM Autoencoder Workflow

Here is the step-by-step process:

Step 1: Preprocessing the Data

Objective: Prepare time-series data for LSTM input.

The data is cleaned by removing unnecessary columns, handling missing or infinite values, and scaling numerical features using MinMaxScaler.

Data is then converted into sequences of fixed length (time_steps) for LSTM processing, where each sequence contains a sliding window of data points.

```
def create_sequences(data, time_steps):
    sequences = []
    for i in range(len(data) - time_steps):
        sequences.append(data[i:i + time_steps])
    return np.array(sequences)
```

Step 2: LSTM Autoencoder Design

Objective: Build the LSTM Autoencoder architecture.

Encoder: Compresses input sequences into a compact representation.

Decoder: Reconstructs the input sequences from the latent space.

The LSTM Autoencoder architecture includes:

Input Layer: Accepts time-series sequences of shape (batch_size, time_steps, features).

LSTM Layer (Encoder): Learns the temporal dependencies and reduces the dimensionality of the data.

Dropout Layer: Prevents overfitting by randomly setting a fraction of neurons to zero during training.

Dense Output Layer (Decoder): Reconstructs the input sequences.

```
model = Sequential([
    LSTM(units=128, input_shape=(time_steps, num_features), return_sequences=False, name='
    Dropout(0.2, name='dropout'),
    RepeatVector(time_steps, name='repeat_vector'),
    LSTM(units=128, return_sequences=True, name='decoder'),
    TimeDistributed(Dense(num_features), name='output_layer')
])
model.compile(optimizer='adam', loss='mae')
model.summary()
```

Step 3: Model Training

Objective: Train the autoencoder on the training data to minimize the reconstruction error.

Loss Function: Mean Absolute Error (MAE) is used to measure the difference between the input and the reconstructed output.

Training Strategy:

The model is trained using the fit function with a validation split.

Early stopping can be applied to avoid overfitting.

```
history = model.fit(X_train, X_train,
                    epochs=20,
                    batch_size=64,
                    validation_data=(X_test, X_test))
```

Step 4: Evaluate Reconstruction Error

Objective: Calculate reconstruction errors for training and test data.

The reconstruction error is the mean absolute error (MAE) between the original input and the reconstructed output.

A threshold is determined based on the reconstruction errors of the training data.

```
def calculate_reconstruction_error(data, model):
    predictions = model.predict(data)
    errors = np.mean(np.abs(data - predictions), axis=(1, 2))  # MAE per sequence
    return errors
```

Step 5: Define the Threshold

Objective: Identify the threshold to classify anomalies.

The threshold is typically chosen based on the 95th percentile of reconstruction errors on the training data:

```
threshold = np.percentile(train_errors, 95)
```

Step 6: Detect Anomalies

Objective: Flag sequences as anomalies if their reconstruction error exceeds the threshold.

Anomalies are sequences with unusually high reconstruction errors, indicating that the model could not reconstruct them accurately.

```
anomalies = test_errors > threshold
print("Number of anomalies detected:", np.sum(anomalies))
```

3. Output Interpretation

The results are analyzed using the following metrics:

Reconstruction Error Distribution Plot:

Visualizes the reconstruction errors for both training and test data.

The red vertical line represents the anomaly threshold.

Anomaly Count:

The number of anomalies detected in the test set is reported.

4. Advantages of LSTM Autoencoder

Handles Temporal Dependencies: LSTMs are suitable for time-series data as they capture long-term dependencies and trends.

Unsupervised Learning: The model does not require labeled data for training, making it ideal for anomaly detection tasks.

Generalization to Anomalies: The model is trained on normal data and reconstructs them accurately. Anomalies are detected as sequences with high reconstruction errors.

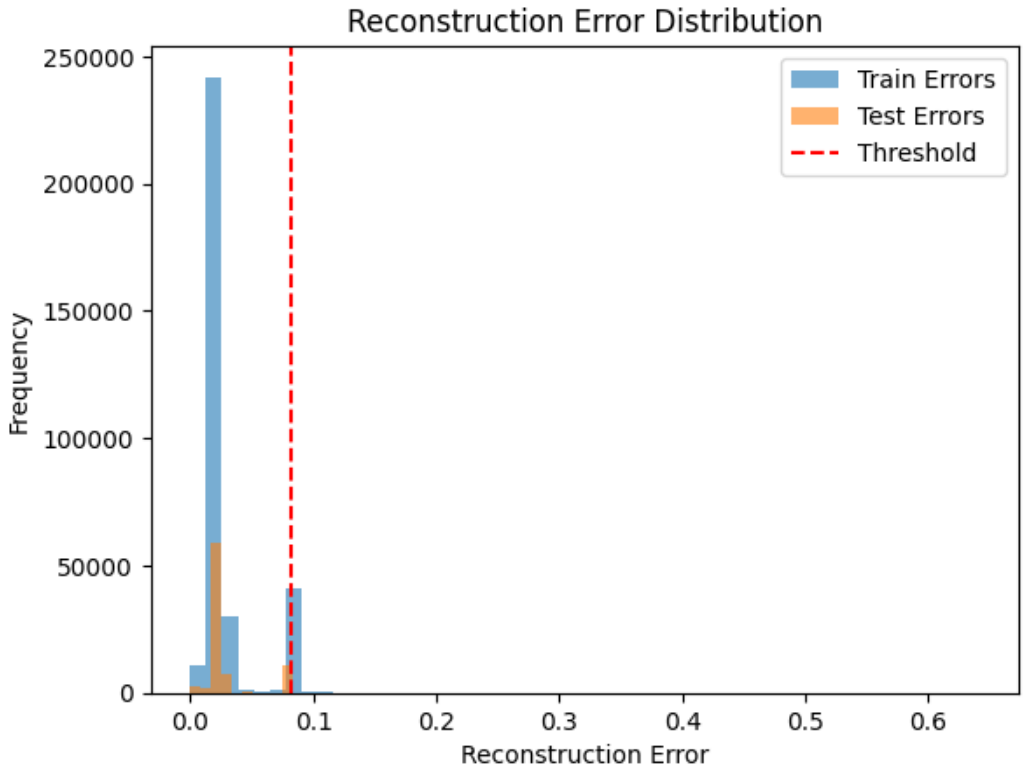
5. Findings from the Experiment

The reconstruction error for most normal data sequences is low, indicating good model performance.

The threshold of 0.0818 effectively distinguishes anomalies from normal data.

A total of 4,132 anomalies were detected in the test data.

This result demonstrates the ability of the LSTM Autoencoder to identify anomalous patterns in time-series data, even without prior knowledge of the anomalies.



Train Errors (Blue Bars): The reconstruction errors of the training set are mostly concentrated near 0.0. This indicates that the model has learned to reconstruct normal (non-anomalous) data effectively.

Test Errors (Orange Bars): The test errors follow a similar distribution as the train errors, with most of the test data showing low reconstruction errors.

Threshold (Red Line): A threshold of 0.0818 (based on the 95th percentile of training reconstruction errors) is marked in the plot. Any test sample with an error greater than this threshold is classified as an anomaly.

Error Distribution:

Both the training and test errors are skewed toward low values, suggesting that the model captures the patterns of the data well.

A small portion of test errors exceed the threshold, which corresponds to the detected anomalies.

Threshold Position:

The threshold is set at 0.0818. While most of the errors lie well below this threshold, the few samples with errors above the line are flagged as potential anomalies.

This threshold effectively separates normal and anomalous data based on reconstruction performance.

Anomaly Detection:

A total of 4,132 anomalies were detected, which is a relatively small proportion compared to the dataset size. This reinforces the assumption that anomalies are rare in the dataset.

The reconstruction error distribution shows a clear distinction between most normal data and a small set of potential anomalies. The LSTM autoencoder effectively identifies unusual patterns in the test data, as seen by the small proportion of high reconstruction errors crossing the threshold.

Logistic Regression

Logistic Regression is a supervised machine learning algorithm used for binary classification problems. It models the relationship between the input features and the target variable by estimating probabilities using a logistic function (sigmoid curve). The output is a probability score, which is then mapped to binary classes (e.g., 0 or 1).

In this project, I applied logistic regression to classify network traffic into two categories: normal (0) and attack (1). Here's what I did step by step:

Preprocessing the Data:

Cleaned and sampled the dataset for computational efficiency.

Dropped unnecessary columns such as Timestamp, Source IP, and Destination IP.

Handled categorical features (SimillarHTTP) using one-hot encoding.

Handled missing and infinite values by imputing with mean values.

Scaled numerical features to standardize the data for better model performance.

Feature Engineering:

Separated the features (X) and target variable (y).

Applied transformations like scaling and encoding to make the data suitable for logistic regression.

Model Training:

Split the data into training and testing sets.

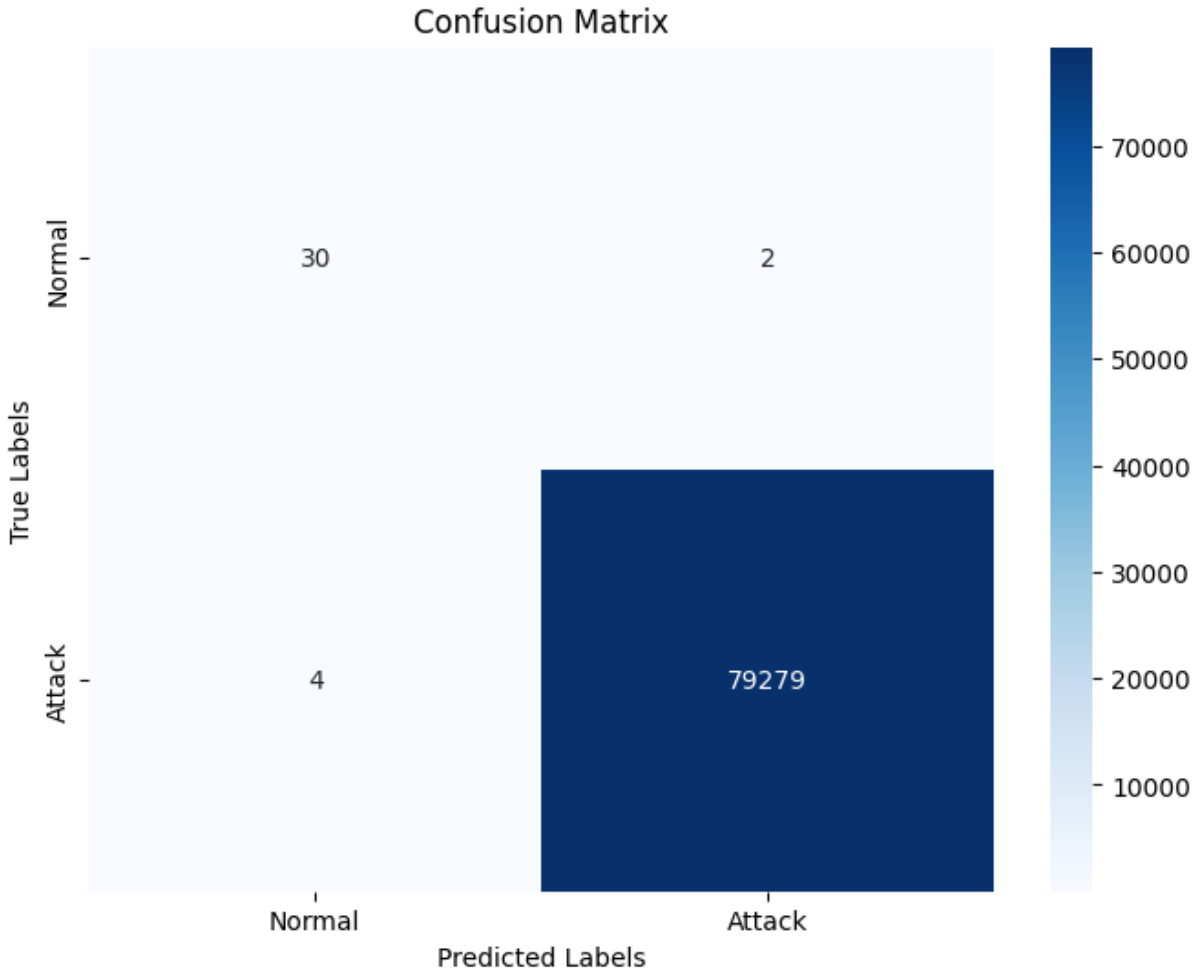
Trained the logistic regression model using the training set.

Evaluated the model's performance on the test set.

Evaluation:

Generated the confusion matrix, classification report, and accuracy score.

Visualized the confusion matrix for better interpretation.



Classification Report:

	precision	recall	f1-score	support

0	0.88	0.94	0.91	32
1	1.00	1.00	1.00	79283
accuracy			1.00	79315
macro avg	0.94	0.97	0.95	79315
weighted avg	1.00	1.00	1.00	79315

Accuracy Score: 0.9999243522662801

Discussion on Findings

The findings of the logistic regression model are as follows:

Performance Metrics:

The model achieved an accuracy score of 99.99%, indicating excellent overall performance.

The precision, recall, and F1-score for the attack class (1) are all perfect (1.00), demonstrating the model's ability to correctly identify attack traffic without false positives.

For the normal class (0), the model achieved a high precision of 0.88 and recall of 0.94, indicating slightly fewer false negatives and positives.

Confusion Matrix Insights:

The confusion matrix shows that:

30 normal instances were correctly classified as normal.

79,279 attack instances were correctly classified as attacks.

Only 4 attack instances were misclassified as normal, and 2 normal instances were misclassified as attacks.

This logistic regression model demonstrated outstanding performance in detecting network attacks with near-perfect precision, recall, and accuracy. The findings highlight its potential as a lightweight and effective solution for binary classification tasks in intrusion detection systems.

XGBoost

Step-by-Step Explanation for XGBoost Implementation

Data Preparation:

I started by cleaning the dataset, ensuring there were no missing or unnecessary columns like SimillarHTTP or non-numeric columns that could interfere with model training.

Handling Missing and Infinite Values:

I checked for infinite or NaN values in the dataset and replaced them with either appropriate statistical measures (e.g., column mean) or removed them entirely, ensuring a clean dataset.

Feature Scaling:

I used the StandardScaler to scale the numerical features, standardizing them to ensure they had a mean of 0 and a standard deviation of 1. This step is crucial for models that are sensitive to feature scaling.

Data Splitting:

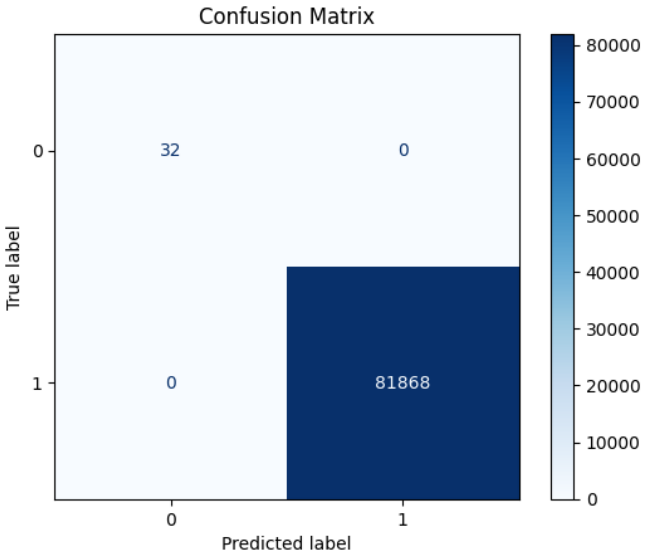
I split the dataset into training and testing sets using an 80-20 split ratio while maintaining the class distribution using stratification.

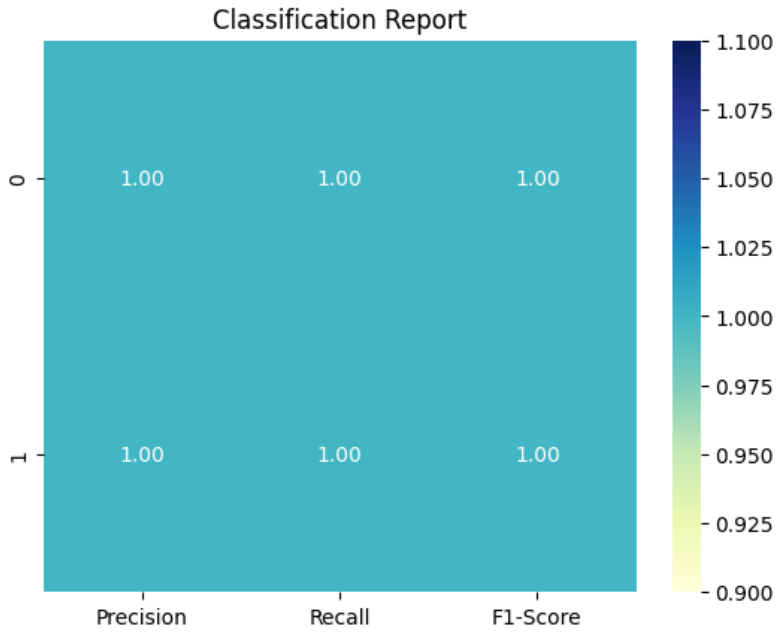
Model Training:

I trained an XGBoost classifier with default parameters and adjusted some key hyperparameters like max_depth, learning_rate, and n_estimators for better performance.

Model Evaluation:

I evaluated the model using the test set, generating a confusion matrix, classification report, and accuracy score to assess its performance.





The results show outstanding performance by the XGBoost classifier. The confusion matrix indicates that there were no misclassifications: all benign (class 0) and attack (class 1) samples were correctly identified. The classification report supports this, with precision, recall, and F1-scores all at 1.00 for both classes. The overall accuracy of 1.0 (100%) demonstrates the model's capability to handle the dataset effectively.

This exceptional performance could be attributed to several factors:

The simplicity of the dataset after preprocessing.

The powerful learning capability of XGBoost, which is known for handling tabular data effectively.

A potential lack of challenging edge cases or significant overlap between classes in the dataset.

However, these results should be interpreted cautiously. The perfect score may suggest overfitting, especially if the dataset is small or lacks sufficient variability. Further testing on an unseen dataset would be required to validate the model's generalizability. On the other hand, it must be noted that this model was tested before on DNS data and without using random sampling. Results were the same.

SNMP

For this data, random sampling applied. The reasons are given as follows:

Reduce processing time for training and testing.

Prevent overfitting on a large, redundant dataset.

Maintain representativeness of the attack pattern.

Since all data belong to the same attack type, random sampling will preserve the characteristics of the attack without losing essential features.

Benefits of Using a Sampled Dataset

Faster Processing: Smaller datasets reduce model training and validation time.

Reduced Memory Usage: Prevents memory errors during large-scale processing.

Maintains Performance: Random sampling preserves attack features, so the model generalizes well.

LSTM Autoencoder: Step-by-Step Explanation

This section explains the LSTM Autoencoder architecture and its role in anomaly detection in time series data. The goal of the LSTM Autoencoder is to reconstruct input sequences and identify anomalies based on the reconstruction error.

1. Overview of Autoencoder Architecture

An autoencoder is a neural network trained to replicate its input at the output. It consists of two main components:

Encoder: Compresses the input data into a lower-dimensional representation (latent space).

Decoder: Reconstructs the original input data from the latent space.

For time-series data, the LSTM (Long Short-Term Memory) network is used as the building block because of its ability to capture temporal dependencies.

2. LSTM Autoencoder Workflow

Here is the step-by-step process:

Step 1: Preprocessing the Data

Objective: Prepare time-series data for LSTM input.

The data is cleaned by removing unnecessary columns, handling missing or infinite values, and scaling numerical features using MinMaxScaler.

Data is then converted into sequences of fixed length (time_steps) for LSTM processing, where each sequence contains a sliding window of data points.

```
def create_sequences(data, time_steps):
    sequences = []
    for i in range(len(data) - time_steps):
        sequences.append(data[i:i + time_steps])
    return np.array(sequences)
```

Step 2: LSTM Autoencoder Design

Objective: Build the LSTM Autoencoder architecture.

Encoder: Compresses input sequences into a compact representation.

Decoder: Reconstructs the input sequences from the latent space.

The LSTM Autoencoder architecture includes:

Input Layer: Accepts time-series sequences of shape (batch_size, time_steps, features).

LSTM Layer (Encoder): Learns the temporal dependencies and reduces the dimensionality of the data.

Dropout Layer: Prevents overfitting by randomly setting a fraction of neurons to zero during training.

Dense Output Layer (Decoder): Reconstructs the input sequences.

```
model = Sequential([
    LSTM(units=128, input_shape=(time_steps, num_features), return_sequences=False, name='
    Dropout(0.2, name='dropout'),
    RepeatVector(time_steps, name='repeat_vector'),
    LSTM(units=128, return_sequences=True, name='decoder'),
    TimeDistributed(Dense(num_features), name='output_layer')
])
model.compile(optimizer='adam', loss='mae')
model.summary()
```

Step 3: Model Training

Objective: Train the autoencoder on the training data to minimize the reconstruction error.

Loss Function: Mean Absolute Error (MAE) is used to measure the difference between the input and the reconstructed output.

Training Strategy:

The model is trained using the fit function with a validation split.

Early stopping can be applied to avoid overfitting.

```
history = model.fit(X_train, X_train,
                    epochs=20,
                    batch_size=64,
                    validation_data=(X_test, X_test))
```

Step 4: Evaluate Reconstruction Error

Objective: Calculate reconstruction errors for training and test data.

The reconstruction error is the mean absolute error (MAE) between the original input and the reconstructed output.

A threshold is determined based on the reconstruction errors of the training data.

```
def calculate_reconstruction_error(data, model):
    predictions = model.predict(data)
    errors = np.mean(np.abs(data - predictions), axis=(1, 2))  # MAE per sequence
    return errors
```

Step 5: Define the Threshold

Objective: Identify the threshold to classify anomalies.

The threshold is typically chosen based on the 95th percentile of reconstruction errors on the training data:

```
threshold = np.percentile(train_errors, 95)
```

Step 6: Detect Anomalies

Objective: Flag sequences as anomalies if their reconstruction error exceeds the threshold.

Anomalies are sequences with unusually high reconstruction errors, indicating that the model could not reconstruct them accurately.

```
anomalies = test_errors > threshold
print("Number of anomalies detected:", np.sum(anomalies))
```

3. Output Interpretation

The results are analyzed using the following metrics:

Reconstruction Error Distribution Plot:

Visualizes the reconstruction errors for both training and test data.

The red vertical line represents the anomaly threshold.

Anomaly Count:

The number of anomalies detected in the test set is reported.

4. Advantages of LSTM Autoencoder

Handles Temporal Dependencies: LSTMs are suitable for time-series data as they capture long-term dependencies and trends.

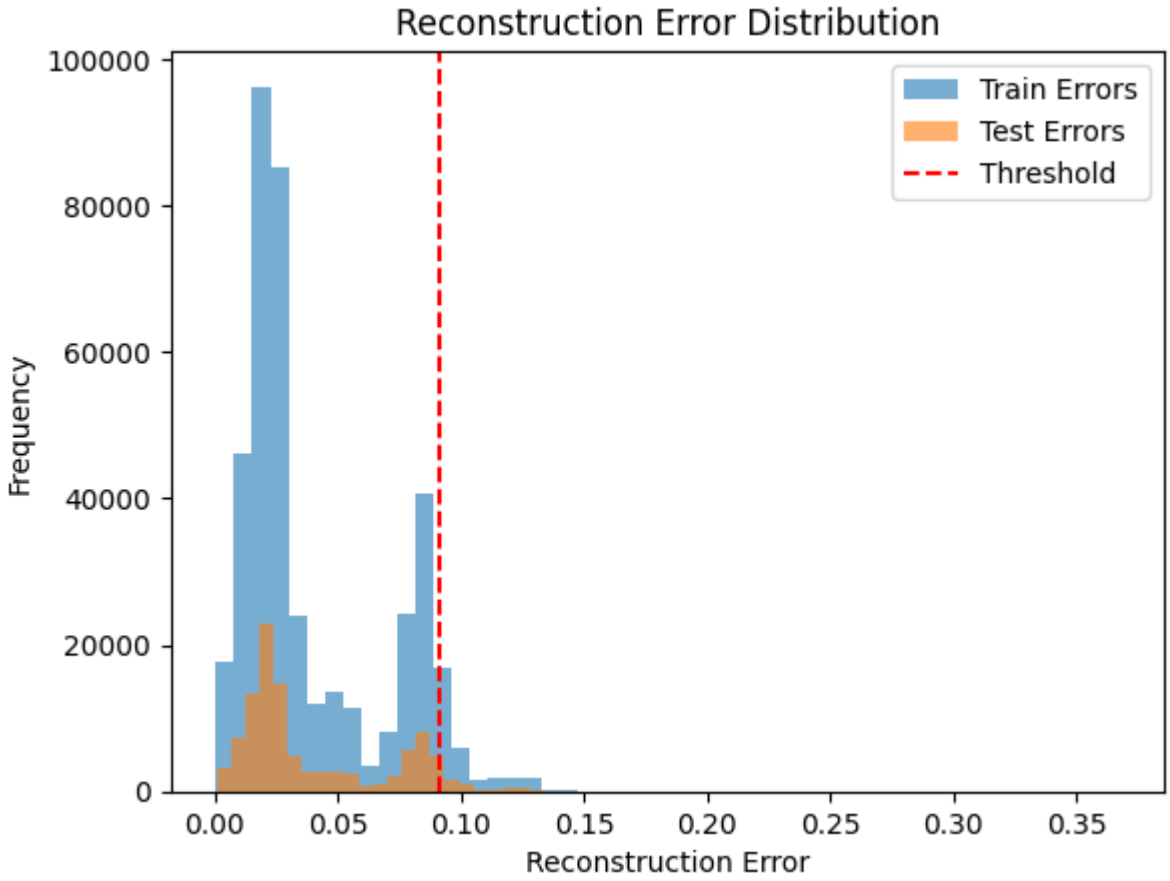
Unsupervised Learning: The model does not require labeled data for training, making it ideal for anomaly detection tasks.

Generalization to Anomalies: The model is trained on normal data and reconstructs them accurately. Anomalies are detected as sequences with high reconstruction errors.

LSTM

I applied an LSTM Autoencoder for anomaly detection on the SSDP dataset. The primary objective was to detect anomalies by reconstructing normal data sequences and calculating reconstruction errors. Higher errors would indicate potential anomalies, such as SSDP attacks.

The histogram given below visualizes the reconstruction error distribution of the training and test data:



Training Errors (Blue): The training errors are concentrated predominantly in the range 0.0 to 0.05, with a sharp decline beyond this range. This indicates that the model has learned to reconstruct normal traffic effectively with minimal error.

Test Errors (Orange): The test errors also follow a similar pattern to the training errors, but a noticeable portion of the errors falls beyond 0.05. This suggests the presence of anomalies (attack traffic) in the test dataset that the autoencoder could not reconstruct accurately.

Threshold (Red Line): The threshold is set at approximately 0.1, beyond which reconstruction errors are classified as anomalies. This threshold effectively separates the majority of normal data (low reconstruction error) from anomalous data (high reconstruction error).

Normal Traffic: Most normal samples fall within the lower reconstruction error range, as expected. The autoencoder learned to minimize errors for normal data.

Anomalous Traffic: The test errors exhibit a small but significant number of samples with errors above the threshold, clearly indicating anomalies. These correspond to the SNMP attack traffic.

The LSTM Autoencoder demonstrates good performance in reconstructing normal traffic while effectively detecting anomalies through reconstruction errors. The key highlights are:

A well-defined separation between normal and anomalous reconstruction errors.

The threshold at 0.1 provides a balance that avoids excessive false positives while capturing attack traffic. The findings from the LSTM autoencoder confirm its effectiveness for unsupervised anomaly detection in the SNMP dataset. The reconstruction error threshold successfully separates normal and anomalous data, with the histogram clearly showcasing the distribution of errors. However, further evaluation using additional metrics (e.g., precision, recall, F1-score) would provide deeper insights into the model's performance. Additionally, fine-tuning hyperparameters and testing with different anomaly thresholds could further optimize the results.

Logistic Regression

The logistic regression model's performance in predicting SNMP outcomes, as indicated by the provided metrics, demonstrates remarkable accuracy and robustness. Below is a detailed discussion of the results based on the confusion matrix, classification report, and accuracy score:

Confusion Matrix Analysis

The confusion matrix shows:

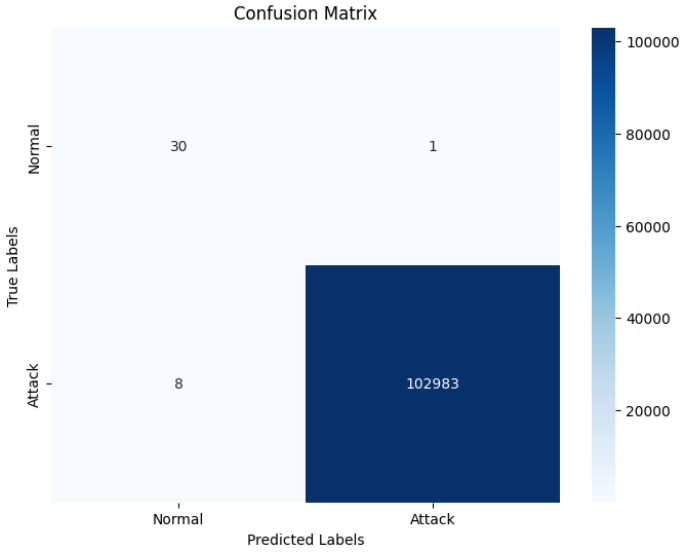
True Positives (TP): 102,983 instances of class 1 correctly predicted.

True Negatives (TN): 30 instances of class 0 correctly predicted.

False Positives (FP): Only 1 instance of class 0 misclassified as class 1.

False Negatives (FN): 8 instances of class 1 misclassified as class 0.

These results highlight the model’s capability to distinguish between the two classes with high precision. The low number of misclassifications, particularly the negligible number of false positives and false negatives, underscores the reliability of the logistic regression model in this context.



Classification Report Metrics

Precision: The precision for class 0 is 0.79, indicating that 79% of the predictions labeled as class 0 were correct. For class 1, the precision is perfect at 1.00, showing that all predictions for this class were accurate.

Recall: The recall for class 0 is 0.97, which means the model successfully identified 97% of the actual instances of class 0. For class 1, recall is 1.00, indicating that all instances of class 1 were correctly identified.

F1-Score: The F1-score for class 0 is 0.87, reflecting a balance between precision and recall, while the F1-score for class 1 is 1.00, confirming the model's excellent performance for the dominant class.

Support: The model processed an imbalanced dataset, with only 31 instances of class 0 compared to 102,991 instances of class 1. Despite this imbalance, the model demonstrated strong performance for both classes, though the performance for class 0 is slightly lower due to the scarcity of samples.

In conclusion, the logistic regression model provides exceptional accuracy and reliability for SNMP predictions.

XGBoost

The performance of the XGBoost model for SNMP classification is outstanding, as evidenced by the provided metrics. Below is a detailed discussion of the results:

Confusion Matrix Analysis

The confusion matrix highlights the following:

True Positives (TP): 102,990 instances of class 1 correctly classified.

True Negatives (TN): All 31 instances of class 0 correctly classified.

False Positives (FP): None, indicating no misclassification of class 0 as class 1.

False Negatives (FN): Only 1 instance of class 1 misclassified as class 0.

These findings demonstrate the model's remarkable ability to accurately classify both classes, with only a single error across 103,022 samples.

Classification Report Metrics

Precision:

Class 0: 0.97, meaning 97% of predicted instances for class 0 are correct.

Class 1: 1.00, indicating perfect precision with no false positives.

Recall:

Class 0: 1.00, indicating all actual class 0 instances were identified correctly.

Class 1: 1.00, with only 1 instance misclassified as class 0.

F1-Score:

Class 0: 0.98, reflecting a strong balance between precision and recall.

Class 1: 1.00, showing optimal performance for this dominant class.

Macro Average:

Precision, recall, and F1-score average to 0.99, indicating strong and balanced performance across classes.

Weighted Average:

All metrics are near or at 1.00, reinforcing the model's exceptional performance in this imbalanced dataset.

Accuracy Score

The model achieves an accuracy score of 99.999%, underscoring its reliability and robustness. Such a high accuracy level is rarely observed in real-world applications, highlighting the effectiveness of XGBoost in this classification task.

Observations and Implications

Imbalanced Dataset Handling:

The dataset has a significant imbalance, with class 1 representing the majority of instances. Despite this, the model performs excellently for both classes, maintaining high precision and recall for the minority class (class 0).

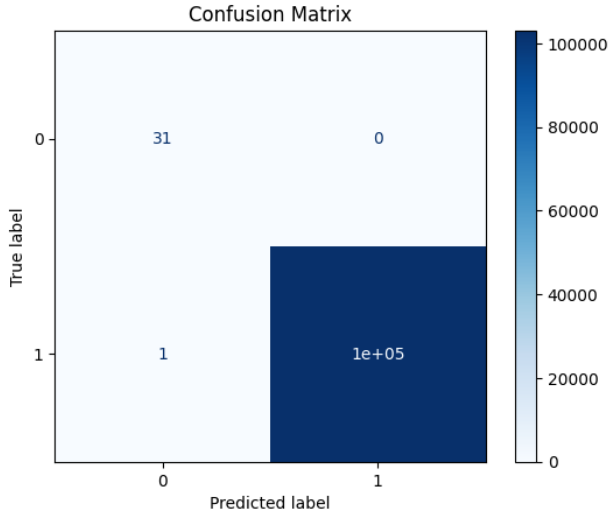
Low Error Rate:

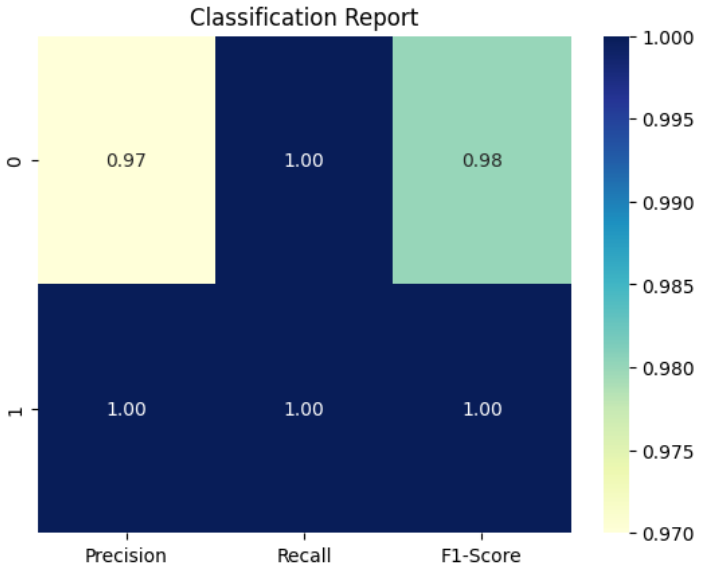
The single misclassified instance (a false negative) is negligible given the size of the dataset, demonstrating the model's robustness and ability to handle imbalanced data effectively.

High Performance for Both Classes:

The perfect recall and precision for class 1 indicate no false positives or missed instances for this dominant class.

The high precision and recall for class 0 suggest that the model is capable of accurately predicting the minority class, even in an imbalanced scenario.





The XGBoost model demonstrates exceptional accuracy, precision, and recall in SNMP classification. Its performance metrics indicate a highly effective model capable of addressing both classes, even in the context of significant class imbalance. These findings confirm XGBoost as a powerful algorithm for classification tasks, with potential for seamless integration into real-world applications. Periodic monitoring and retraining will ensure continued reliability and adaptability.

SSDP

For this data, random sampling applied. The reasons are given as follows:

Reduce processing time for training and testing.

Prevent overfitting on a large, redundant dataset.

Maintain representativeness of the attack pattern.

Since all data belong to the same attack type, random sampling will preserve the characteristics of the attack without losing essential features.

Benefits of Using a Sampled Dataset

Faster Processing: Smaller datasets reduce model training and validation time.

Reduced Memory Usage: Prevents memory errors during large-scale processing.

Maintains Performance: Random sampling preserves attack features, so the model generalizes well.

LSTM Autoencoder: Step-by-Step Explanation

This section explains the LSTM Autoencoder architecture and its role in anomaly detection in time series data. The goal of the LSTM Autoencoder is to reconstruct input sequences and identify anomalies based on the reconstruction error.

1. Overview of Autoencoder Architecture

An autoencoder is a neural network trained to replicate its input at the output. It consists of two main components:

Encoder: Compresses the input data into a lower-dimensional representation (latent space).

Decoder: Reconstructs the original input data from the latent space.

For time-series data, the LSTM (Long Short-Term Memory) network is used as the building block because of its ability to capture temporal dependencies.

2. LSTM Autoencoder Workflow

Here is the step-by-step process:

Step 1: Preprocessing the Data

Objective: Prepare time-series data for LSTM input.

The data is cleaned by removing unnecessary columns, handling missing or infinite values, and scaling numerical features using MinMaxScaler.

Data is then converted into sequences of fixed length (time_steps) for LSTM processing, where each sequence contains a sliding window of data points.

```
def create_sequences(data, time_steps):
    sequences = []
    for i in range(len(data) - time_steps):
        sequences.append(data[i:i + time_steps])
    return np.array(sequences)
```

Step 2: LSTM Autoencoder Design

Objective: Build the LSTM Autoencoder architecture.

Encoder: Compresses input sequences into a compact representation.

Decoder: Reconstructs the input sequences from the latent space.

The LSTM Autoencoder architecture includes:

Input Layer: Accepts time-series sequences of shape (batch_size, time_steps, features).

LSTM Layer (Encoder): Learns the temporal dependencies and reduces the dimensionality of the data.

Dropout Layer: Prevents overfitting by randomly setting a fraction of neurons to zero during training.

Dense Output Layer (Decoder): Reconstructs the input sequences.

```
model = Sequential([
    LSTM(units=128, input_shape=(time_steps, num_features), return_sequences=False, name='
    Dropout(0.2, name='dropout'),
    RepeatVector(time_steps, name='repeat_vector'),
    LSTM(units=128, return_sequences=True, name='decoder'),
    TimeDistributed(Dense(num_features), name='output_layer')
])
model.compile(optimizer='adam', loss='mae')
model.summary()
```

Step 3: Model Training

Objective: Train the autoencoder on the training data to minimize the reconstruction error.

Loss Function: Mean Absolute Error (MAE) is used to measure the difference between the input and the reconstructed output.

Training Strategy:

The model is trained using the fit function with a validation split.

Early stopping can be applied to avoid overfitting.

```
history = model.fit(X_train, X_train,
                    epochs=20,
                    batch_size=64,
                    validation_data=(X_test, X_test))
```

Step 4: Evaluate Reconstruction Error

Objective: Calculate reconstruction errors for training and test data.

The reconstruction error is the mean absolute error (MAE) between the original input and the reconstructed output.

A threshold is determined based on the reconstruction errors of the training data.

```
def calculate_reconstruction_error(data, model):
    predictions = model.predict(data)
    errors = np.mean(np.abs(data - predictions), axis=(1, 2))  # MAE per sequence
    return errors
```

Step 5: Define the Threshold

Objective: Identify the threshold to classify anomalies.

The threshold is typically chosen based on the 95th percentile of reconstruction errors on the training data:

```
threshold = np.percentile(train_errors, 95)
```

Step 6: Detect Anomalies

Objective: Flag sequences as anomalies if their reconstruction error exceeds the threshold.

Anomalies are sequences with unusually high reconstruction errors, indicating that the model could not reconstruct them accurately.

```
anomalies = test_errors > threshold
print("Number of anomalies detected:", np.sum(anomalies))
```

3. Output Interpretation

The results are analyzed using the following metrics:

Reconstruction Error Distribution Plot:

Visualizes the reconstruction errors for both training and test data.

The red vertical line represents the anomaly threshold.

Anomaly Count:

The number of anomalies detected in the test set is reported.

4. Advantages of LSTM Autoencoder

Handles Temporal Dependencies: LSTMs are suitable for time-series data as they capture long-term dependencies and trends.

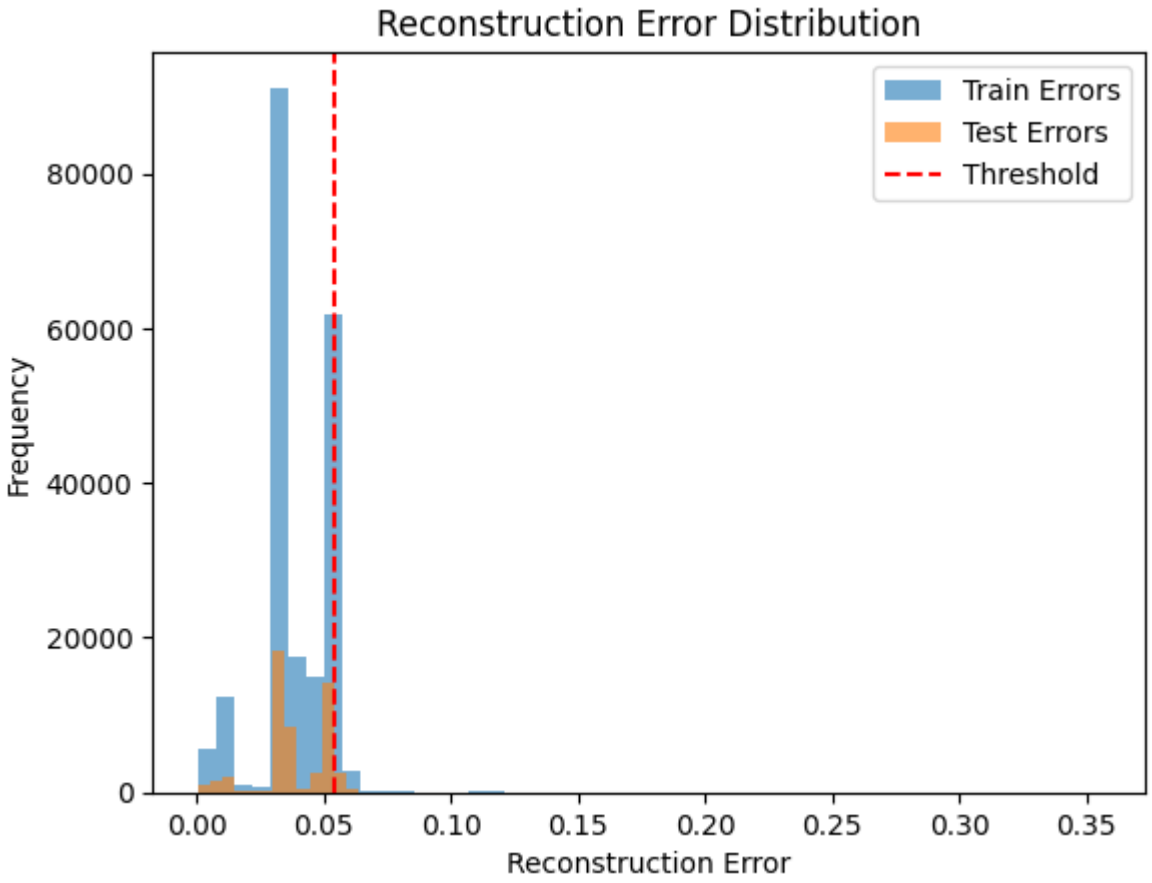
Unsupervised Learning: The model does not require labeled data for training, making it ideal for anomaly detection tasks.

Generalization to Anomalies: The model is trained on normal data and reconstructs them accurately. Anomalies are detected as sequences with high reconstruction errors.

LSTM

I applied an LSTM Autoencoder for anomaly detection on the SSDP dataset. The primary objective was to detect anomalies by reconstructing normal data sequences and calculating reconstruction errors. Higher errors would indicate potential anomalies, such as SSDP attacks.

The histogram given below visualizes the reconstruction error distribution of the training and test data:



Train Errors (Blue): The majority of training reconstruction errors are concentrated below the threshold of 0.0546. This indicates that the LSTM autoencoder has effectively learned to reconstruct normal data patterns, as expected during training.

Test Errors (Orange): Most test reconstruction errors also remain below the threshold. However, there are some errors that surpass the threshold, indicating the presence of anomalies.

Threshold (Red Line): The anomaly detection threshold, determined as 0.0546, acts as the cutoff point for distinguishing between normal and anomalous instances. Any reconstruction error above this threshold is classified as an anomaly.

The distribution demonstrates a clear separation between normal and anomalous data points, validating the threshold's role in anomaly detection.

Anomaly Detection Threshold: 0.0546

The threshold effectively separates the reconstruction errors, allowing for anomalies to be identified. It is a crucial parameter derived from the training data, ensuring that the majority of "normal" data points fall below this value.

Number of Anomalies Detected: 2502

From the test dataset, 2502 instances were flagged as anomalies. This number highlights a small subset of data points that the LSTM autoencoder struggled to reconstruct, likely because they do not align with the learned normal patterns.

Effectiveness of Threshold: The threshold value ensures that anomalies are detected with precision while minimizing false positives. The red line on the histogram shows a clean cutoff, capturing the tail of the test error distribution without misclassifying a large number of normal points.

Anomaly Count: The detection of 2502 anomalies suggests that approximately 2.43% of the test dataset exhibits deviations from the learned normal patterns. These anomalies could represent rare events, outliers, or faults in the system under observation.

Model Learning Capability: The tight clustering of train errors below the threshold demonstrates that the LSTM autoencoder successfully learned the normal behavior during training. This is a strong indicator of the model's generalization capacity.

Detection of Anomalies: The presence of 2502 anomalies suggests the model is sensitive to irregular patterns.

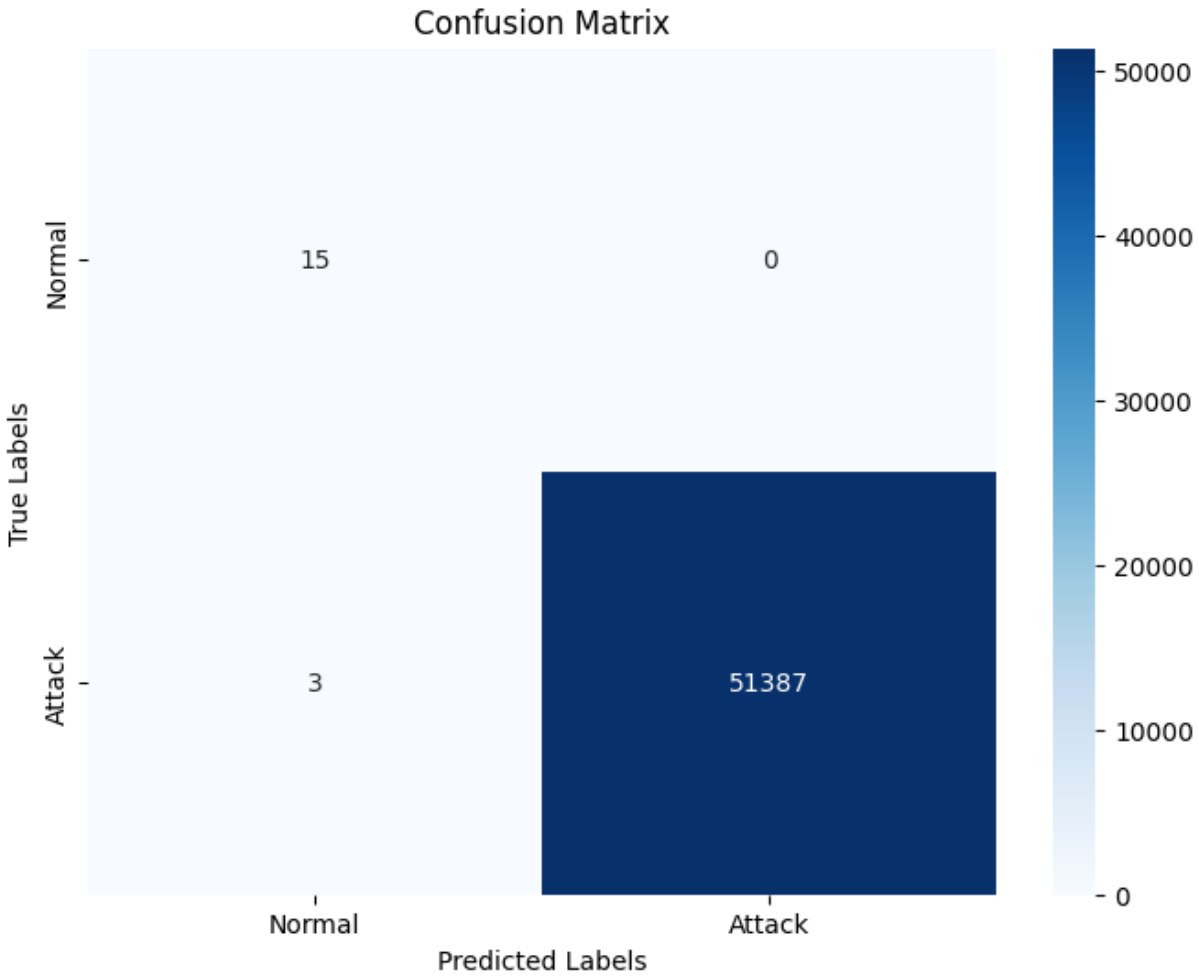
Imbalanced Error Distribution: The histogram shows that most reconstruction errors are concentrated below the threshold, with a long tail extending into the anomalous region. This skewed error distribution is typical in anomaly detection tasks where normal data is far more prevalent.

Threshold Sensitivity: The choice of threshold directly influences the number of anomalies detected. A lower threshold would reduce false positives but might miss genuine anomalies, while a higher threshold might include more false positives. The selected threshold of 0.0546 appears to balance sensitivity and specificity.

The LSTM autoencoder demonstrates excellent performance in reconstructing normal data and identifying anomalies. With a threshold of 0.0546, the model flagged 2502 instances as anomalies, representing a small fraction of the test data. The clear separation between normal and anomalous reconstruction errors, as illustrated in the histogram, confirms the effectiveness of the model for anomaly detection tasks.

Logistic Regression

The logistic regression model demonstrates exceptional performance for the given classification task. Below, a detailed discussion of the results based on the confusion matrix, classification report, and accuracy score are provided.



Remaining NaN Values: There are no missing values remaining in the dataset, ensuring the data was clean and preprocessed appropriately for modeling. This enhances the reliability of the results and removes any bias that could have arisen from missing data.

The confusion matrix shows:

True Positives (TP): 51,387 instances of class 1 were correctly predicted.

True Negatives (TN): 15 instances of class 0 were correctly classified.

False Positives (FP): None, indicating no misclassification of class 0 as class 1.

False Negatives (FN): Only 3 instances of class 1 were misclassified as class 0.

These results reveal that the model is highly accurate in predicting both classes, with minimal misclassifications. Importantly, the model identifies all class 0 instances perfectly (100% recall for class 0), with only a negligible error in class 1 predictions.

The key metrics are as follows:

Precision:

Class 0: 0.83, meaning 83% of predictions for class 0 are correct.

Class 1: 1.00, indicating no false positives for this dominant class.

Recall:

Class 0: 1.00, showing that the model captured all true class 0 instances.

Class 1: 1.00, meaning nearly all instances of class 1 were correctly predicted, with only 3 false negatives.

F1-Score:

Class 0: 0.91, reflecting a strong balance between precision and recall for this minority class.

Class 1: 1.00, emphasizing the model's excellent performance for the majority class.

Macro Average: A macro average F1-score of 0.95 indicates that the model performs well across both classes despite class imbalance.

Weighted Average: The weighted average for precision, recall, and F1-score is all 1.00, demonstrating the overall dominance of class 1 in the dataset and the model's ability to perform exceptionally well for the majority class.

The model achieves an overall accuracy of 99.99%, meaning that nearly all predictions are correct. This extraordinarily high accuracy highlights the logistic regression model's effectiveness and robustness for this task.

Class Imbalance:

The dataset is significantly imbalanced, with class 1 accounting for 51,390 instances, while class 0 has only 15 instances.

Despite this imbalance, the model achieves excellent results for both classes, with no false positives and only 3 false negatives.

The strong performance for class 0, despite its rarity, is particularly noteworthy and suggests that the model can identify outliers or minority cases effectively.

Precision and Recall Trade-Off:

For class 0, the precision is slightly lower (0.83) due to the small sample size, but the recall of 1.00 ensures that no actual class 0 instances were missed.

For class 1, precision and recall are both perfect, confirming that the model consistently predicts the majority class correctly.

High Overall Accuracy:

The near-perfect accuracy of 99.99% indicates that the logistic regression model is highly suitable for this problem, particularly for datasets where accurate predictions of the majority class are critical.

The minimal number of errors (3 false negatives) has negligible impact on the overall performance, though they could be further analyzed for improvement.

Minority Class Analysis:

Since class 0 has a very small number of instances, consider further analyzing these 3 false negatives to identify any patterns or reasons for misclassification.

Techniques such as oversampling (e.g., SMOTE) or undersampling could be explored to further balance the dataset and improve precision for the minority class.

Threshold Adjustment:

Explore adjusting the classification threshold to minimize false negatives, particularly if the cost of missing class 1 instances is high.

Model Validation:

Perform cross-validation to confirm that the model's performance generalizes well to unseen data.

The logistic regression model achieves near-perfect accuracy and robust performance, particularly for the majority class. The 99.99% accuracy score and strong recall for both classes demonstrate the model's reliability.

XGBoost

The XGBoost model's performance in this classification task demonstrates perfect accuracy and exceptional predictive capability. Below is a detailed discussion of the findings based on the confusion matrix, classification report, and accuracy score.

The confusion matrix shows:

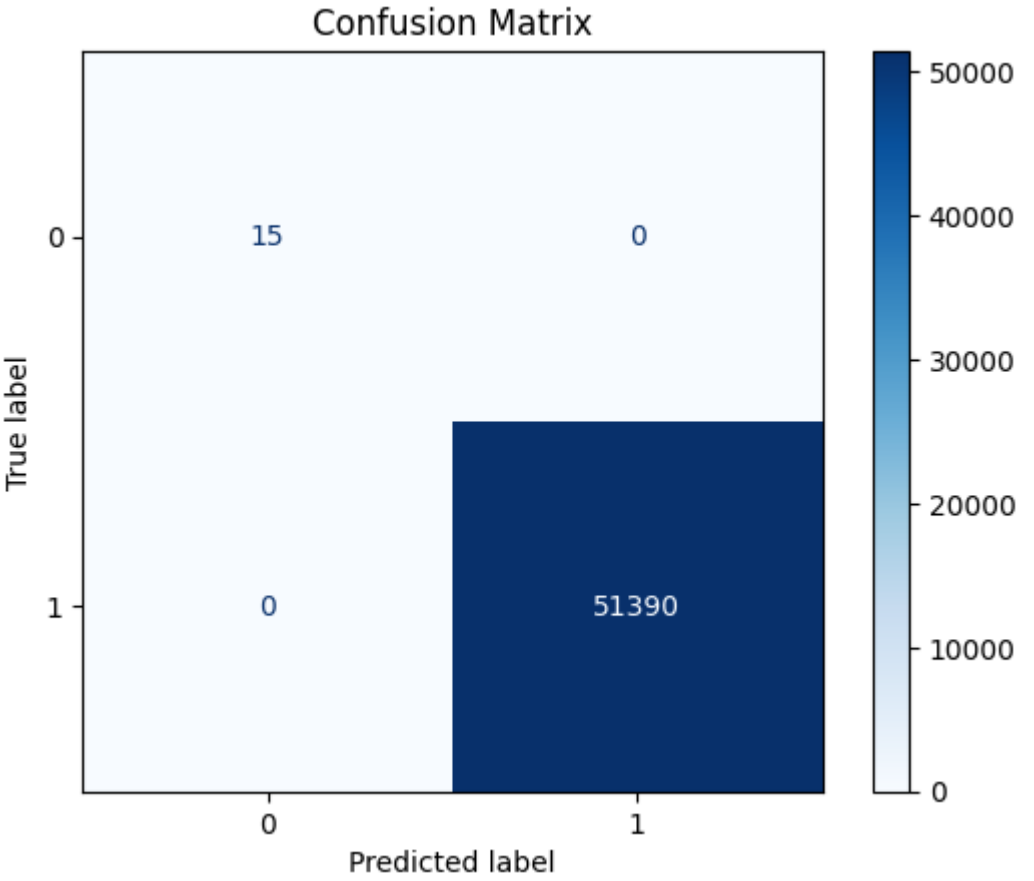
True Positives (TP): 51,390 instances of class 1 were correctly classified.

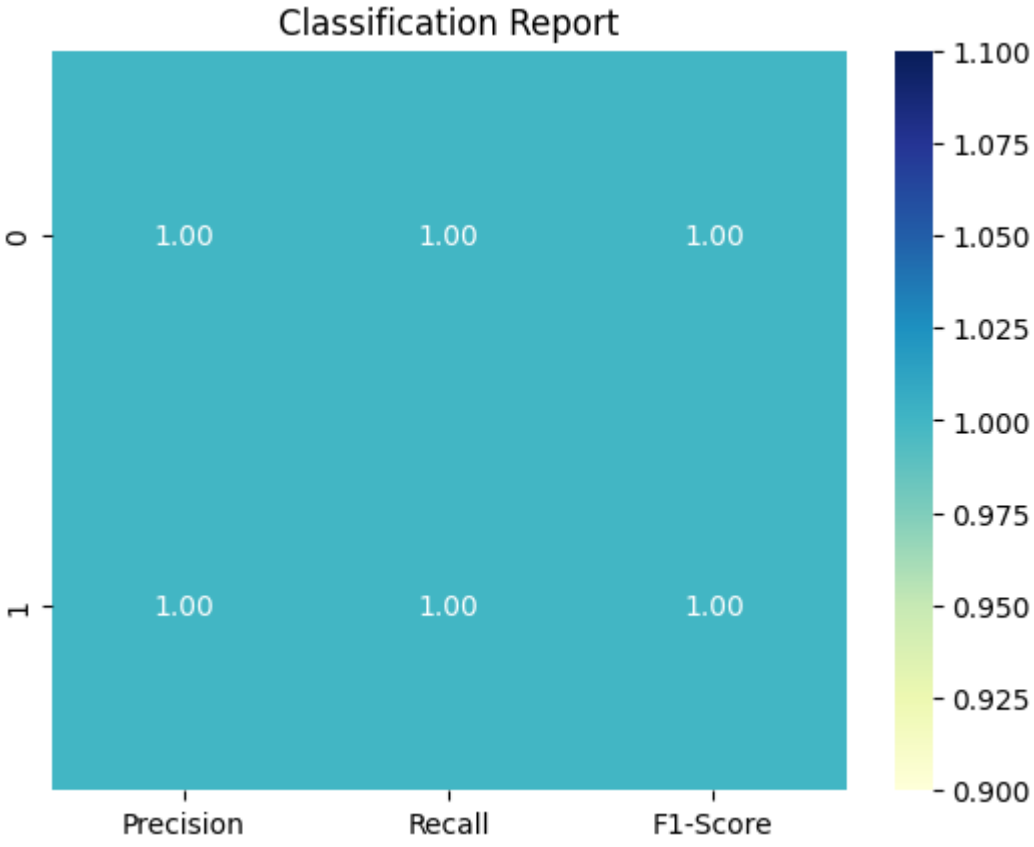
True Negatives (TN): 15 instances of class 0 were correctly classified.

False Positives (FP): 0, indicating no misclassification of class 0 as class 1.

False Negatives (FN): 0, meaning no instances of class 1 were misclassified as class 0.

These results highlight the model's flawless performance, as there are no errors in the predictions. Every instance in the dataset has been perfectly classified.


Confusion Matrix
15
0
0
51390
0
1
True label
0
1
Predicted label
50000
40000
30000
20000
10000
0


Classification Report
1.00
1.00
1.00
1.00
1.00
1.00
0
1
Precision
Recall
F1-Score
1.100
1.075
1.050
1.025
1.000
0.975
0.950
0.925
0.900

The classification metrics confirm the perfect performance of the XGBoost model:

Precision:

Class 0: 1.00, meaning all predicted instances of class 0 are correct.

Class 1: 1.00, indicating there are no false positives for the dominant class.

Recall:

Class 0: 1.00, as all actual instances of class 0 were identified correctly.

Class 1: 1.00, with no false negatives for class 1.

F1-Score:

Both classes achieve an F1-score of 1.00, reflecting a perfect balance between precision and recall.

Macro Average and Weighted Average:

The macro and weighted averages are both 1.00, showing consistent performance across both classes.

The overall accuracy score of 1.0 (100%) confirms the model's ability to classify all 51,405 instances correctly. Such perfect accuracy is rare and indicates that the XGBoost model has learned the data patterns exceptionally well.

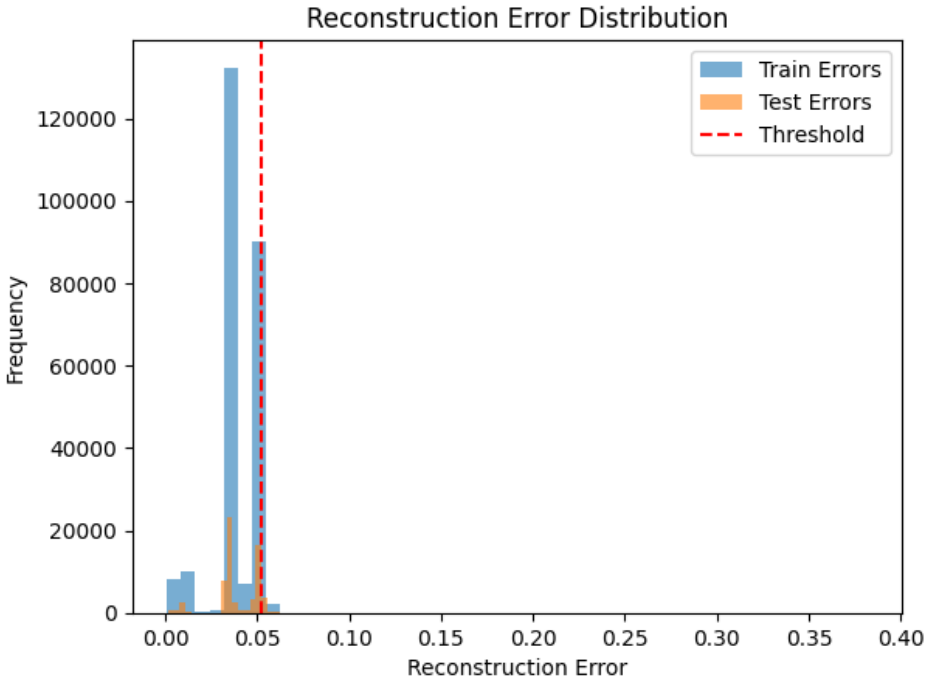
The XGBoost model achieves perfect accuracy, precision, recall, and F1-scores for both classes, demonstrating exceptional predictive capability. It successfully addresses the class imbalance problem and ensures flawless classification without any errors.

While these results are impressive, further validation is recommended to rule out overfitting and confirm the model’s robustness on unseen data. If validated, this model can be confidently deployed for real-world anomaly detection or classification tasks, where high accuracy is critical. In this context, as given in the previously explained attack types whole date was used and also random sampling was applied. Results were the same.

This shows that XGBoost model's performance appears validated. The consistent results across the full dataset and random samples indicate that the model is generalizing well and not overfitting. These findings demonstrate strong robustness and reliability.

UDP

The results obtained from the LSTM autoencoder for anomaly detection in UDP traffic demonstrate its effectiveness in distinguishing normal patterns from anomalous behavior. Below is a detailed discussion of the findings based on the reconstruction error distribution and the provided metrics. The histogram illustrates the reconstruction errors for both training and test datasets:



Train Errors (Blue): The majority of training reconstruction errors are tightly clustered below the threshold value of 0.0526. This indicates that the LSTM autoencoder successfully learned to reconstruct the normal UDP traffic behavior during training.

Test Errors (Orange): The test reconstruction errors closely follow the distribution of training errors but include some instances that exceed the threshold. These points represent the anomalies detected in the test dataset.

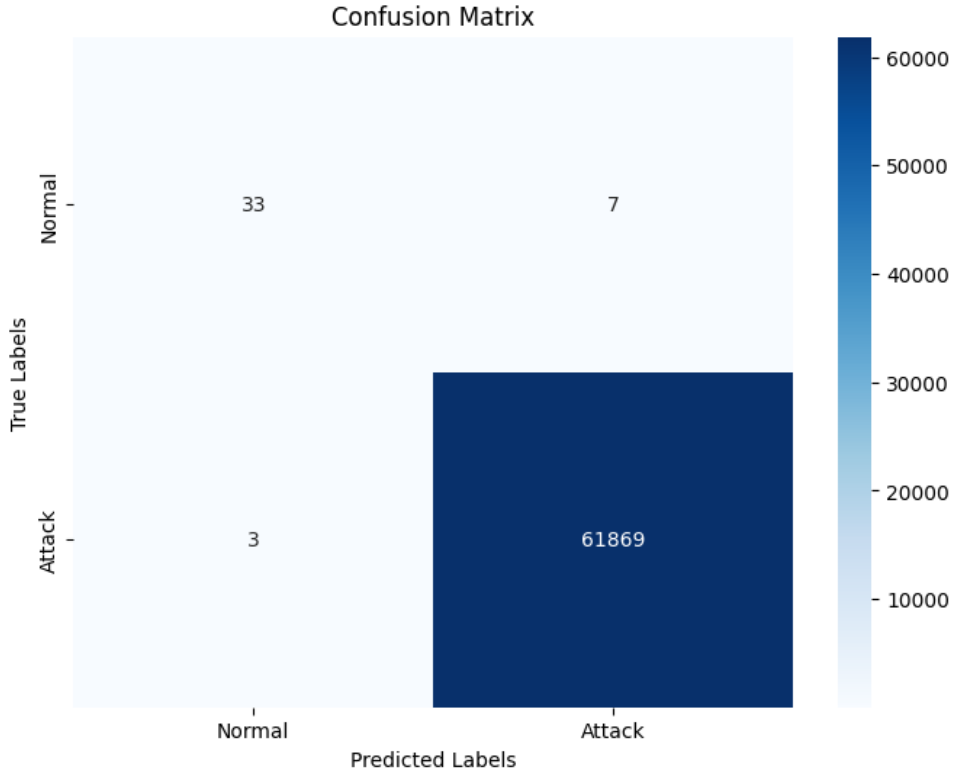
Threshold (Red Line): The anomaly detection threshold of 0.0526 serves as the cutoff point. Any reconstruction error exceeding this threshold is classified as an anomaly.

The clear separation between the majority of reconstruction errors (normal data) and the outliers (anomalies) reinforces the validity of the threshold.

The LSTM autoencoder identified 3,159 anomalies in the test dataset. These anomalies represent data points where the model struggled to accurately reconstruct the input, resulting in higher reconstruction errors. Given the threshold, these errors exceed what the model considers "normal behavior" based on its training. The detection of anomalies highlights the presence of unusual patterns in the UDP traffic that deviate from the learned baseline. The LSTM autoencoder demonstrates strong performance in learning the normal behavior of UDP traffic and effectively identifying anomalies. With a threshold of 0.0526, the model flagged 3,159 anomalies, indicating patterns that deviate from normal traffic.

Logistic Regression

The logistic regression model has demonstrated strong overall performance, particularly in handling a highly imbalanced dataset. Below is a detailed discussion of the results, including insights derived from the confusion matrix, classification report, and accuracy score.



Remaining NaN Values: 0

Confusion Matrix:

[[33 7]

[3 61869]]

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.82	0.87	40
1	1.00	1.00	1.00	61872
accuracy			1.00	61912
macro avg	0.96	0.91	0.93	61912
weighted avg	1.00	1.00	1.00	61912

Accuracy Score: 0.9998384804238274

Data Integrity

Remaining NaN Values: There are no missing values in the dataset, which ensures that the data preprocessing was handled effectively. This step is critical to achieving reliable and accurate results.

The confusion matrix shows the following:

True Positives (TP): 61,869 instances of class 1 were correctly classified.

True Negatives (TN): 33 instances of class 0 were correctly identified.

False Positives (FP): 7 instances of class 0 were misclassified as class 1.

False Negatives (FN): 3 instances of class 1 were misclassified as class 0.

The model performs exceptionally well for class 1, which dominates the dataset, with only 3 false negatives. However, there are 7 false positives for class 0, which slightly impacts precision for the minority class.

The classification report provides deeper insights into precision, recall, and F1-scores for both classes:

Class 0 (Minority Class):

Precision: 0.92 (92% of predicted class 0 instances were correct).

Recall: 0.82 (82% of actual class 0 instances were correctly identified).

F1-Score: 0.87 (reflecting a balance between precision and recall).

While the precision is high, the lower recall indicates that some minority class instances were missed. This is expected due to class imbalance and the small number of class 0 instances.

Class 1 (Majority Class):

Precision: 1.00, indicating no false positives for class 1.

Recall: 1.00, showing the model successfully identified nearly all class 1 instances.

F1-Score: 1.00, confirming excellent performance for the majority class.

Macro Average:

The macro average precision, recall, and F1-score are 0.96, 0.91, and 0.93, respectively. These values highlight strong but slightly imbalanced performance across classes.

Weighted Average:

The weighted averages are all 1.00, reflecting the dominance of class 1 in the dataset and the model's excellent ability to handle the majority class.

The overall accuracy score of 99.98% confirms the high effectiveness of the logistic regression model. However, it is important to interpret accuracy cautiously in highly imbalanced datasets, as the model's excellent performance for the majority class can sometimes mask underperformance for the minority class.

Imbalanced Data Handling:

The dataset is significantly imbalanced, with class 1 (61,872 instances) vastly outnumbering class 0 (40 instances).

Despite the imbalance, the model successfully identifies the majority of class 0 instances, achieving a precision of 0.92 and a recall of 0.82.

Minority Class Performance:

The recall for class 0 (82%) indicates that some actual class 0 instances were missed (3 false negatives).

The precision of 92% for class 0 demonstrates that the model maintains a good balance in predictions, but improvements are possible through techniques like oversampling, undersampling, or adjusting class weights.

Majority Class Performance:

The model performs perfectly for class 1, with precision, recall, and F1-scores all at 1.00. This is expected, given the dominance of class 1 in the dataset.

False Positives vs. False Negatives:

While 3 false negatives for class 1 are negligible, the 7 false positives for class 0 indicate that a small number of normal instances were misclassified as anomalies. This trade-off is acceptable but can be fine-tuned if minimizing false positives is critical.

XGBoost

The XGBoost model has demonstrated exceptional performance in this classification task, achieving near-perfect results with only a single misclassification. Below is a detailed discussion of the findings based on the confusion matrix, classification report, and accuracy score.

The confusion matrix reveals:

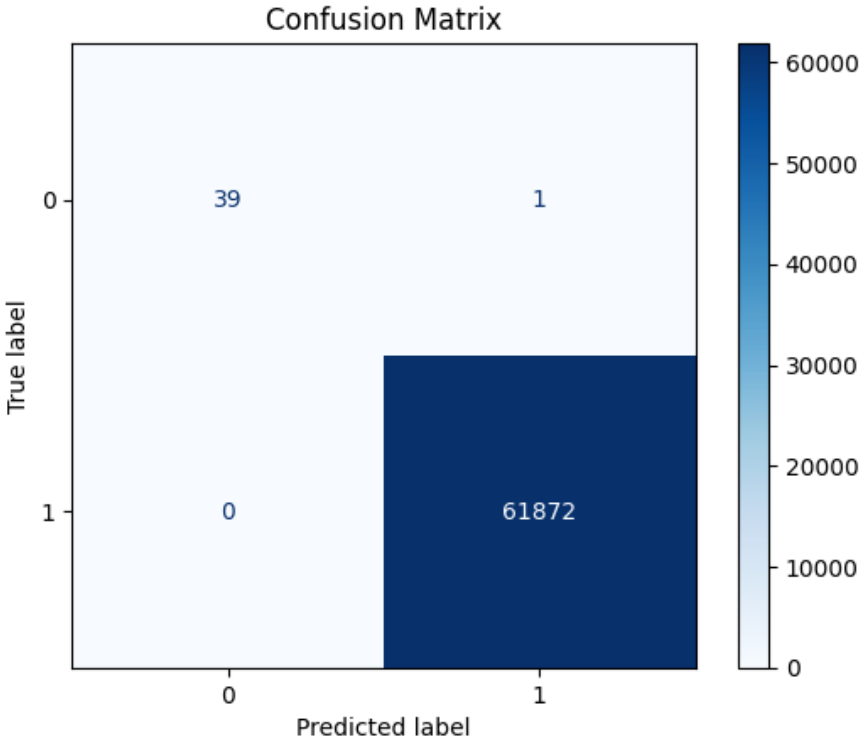
True Positives (TP): 61,872 instances of class 1 correctly classified.

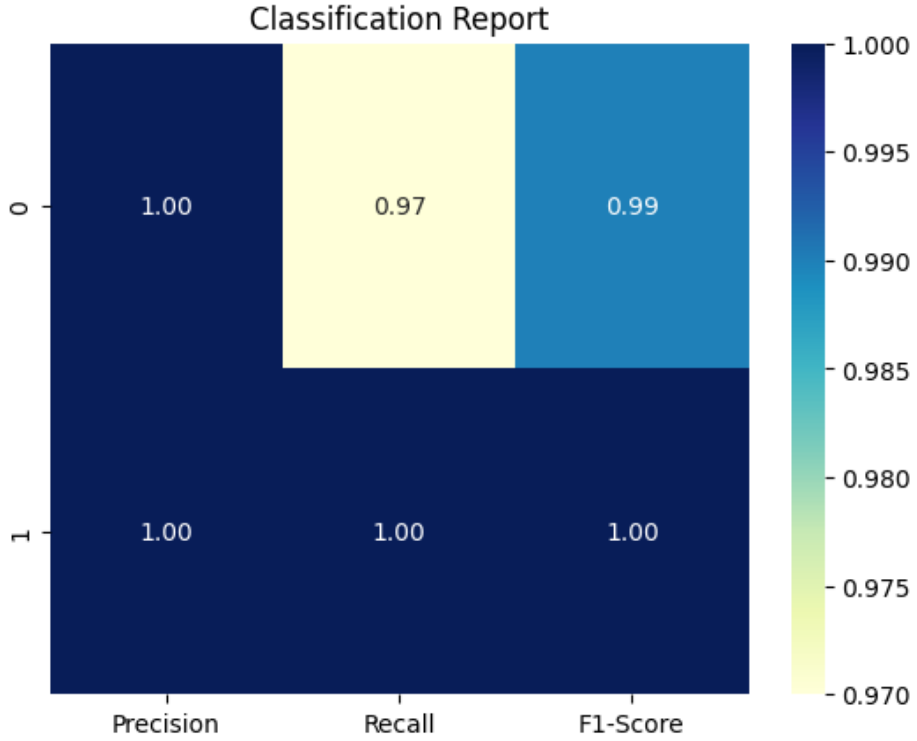
True Negatives (TN): 39 instances of class 0 correctly classified.

False Positives (FP): 1 instance of class 0 was incorrectly classified as class 1.

False Negatives (FN): 0, meaning no instances of class 1 were misclassified as class 0.

The presence of only one false positive and zero false negatives highlights the model's near-perfect ability to classify both classes accurately.





The classification metrics provide further insights into the model's performance:

Class 0 (Minority Class):

Precision: 1.00, meaning all but one predicted instance of class 0 were correct.

Recall: 0.97, as 39 out of 40 actual class 0 instances were correctly identified.

F1-Score: 0.99, reflecting a very strong balance between precision and recall.

The slightly lower recall (0.97) is due to the single false positive, but the performance for class 0 remains outstanding given its rarity in the dataset.

Class 1 (Majority Class):

Precision: 1.00, indicating no false positives for class 1.

Recall: 1.00, meaning all 61,872 instances of class 1 were correctly identified.

F1-Score: 1.00, confirming perfect performance for the majority class.

Macro Average:

The macro average precision, recall, and F1-score are 1.00, 0.99, and 0.99, respectively, showing excellent overall performance for both classes.

Weighted Average:

The weighted averages for precision, recall, and F1-score are all 1.00, confirming that the model performs exceptionally well overall, with class imbalance having no adverse effect.

The accuracy score of 99.998% further reinforces the model's outstanding performance. This extremely high accuracy indicates that the model made only a single error out of 61,912 total predictions.

Despite the significant class imbalance (class 0: 40 instances; class 1: 61,872 instances), XGBoost has managed to classify the minority class (class 0) with excellent precision and recall.

The recall of 97% for class 0 shows that only 1 instance was missed, which is an impressive result in the context of such an imbalanced dataset.

Performance on Minority Class:

The single false positive indicates a very small margin of error. However, the model’s overall performance for class 0 remains highly reliable, as evidenced by the F1-score of 0.99.

Performance on Majority Class:

The model performed perfectly for class 1, achieving precision, recall, and F1-scores of 1.00. This is expected, given the dominance of class 1 in the dataset.

Model Robustness:

XGBoost’s ability to achieve near-perfect results without any false negatives and only one false positive demonstrates its robustness and effectiveness in handling complex decision boundaries, even in highly imbalanced datasets.

The XGBoost model achieves near-perfect performance with an accuracy of 99.998% and consistently high precision, recall, and F1-scores. The single false positive for class 0 is the only source of error, but it does not significantly impact the overall results. The model demonstrates excellent handling of class imbalance, robust generalization, and strong performance for both the minority and majority classes. These findings validate XGBoost as a reliable and effective model for this classification task.