National College of Ireland

Configuration Manual

MSc Research Project
MSc in Cybersecurity

Mustafa Maveeya Maaz Mohammad
Student ID: x23213027

School of Computing
National College of Ireland

Supervisor: Kamil Mahajan

National College of

Ireland MSc Project

Submission Sheet

School of Computing

Student Name:	Mustafa Maveeya Maaz Mohammad		
Student ID:	x23213027		
Programme:	MSc in Cybersecurity	**Year:**	2024-25
Module:	Practicum Part 2		
Supervisor:	Kamil Mahajan		
Submission Due Date:	29/01/25		
Project Title:	Ensuring Network Security in Remote Work Environment		
Word Count:	**963**	**Page Count: 12**	

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Mustafa Maveeya Maaz Mohammad

Date: 29/01/25

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Mustafa Maveeya Maaz Mohammad
x23213027

1 Introduction

This Configuration Manual provides detailed instructions on reproducing the experiments and analyses conducted in the project titled *Ensuring Network Security in Remote Work Environments*. It describes the environment setup, data collection, preprocessing, modelling, and evaluation processes. The manual is organised as follows:

- Section 2: Environment Configuration
- Section 3: Importing Necessary Libraries
- Section 4: Data Collection and Cleaning
- Section 5: Exploratory Data Analysis
- Section 6: Preprocessing
- Section 7: Machine Learning and Deep Learning Models
- Section 8: Evaluation and Results

2 System Requirements

2.1 Hardware Requirements

The project was executed using the following hardware specifications:

- **RAM:** 8 GB DDR4
- **Storage:** 256 GB SSD
- **Processor:** Intel Core i5, 9th Generation
- **GPU:** NVIDIA GTX 1650

2.2 Software Requirements

The following software and libraries were used:

- **Python 3.9**: Programming language
- **Jupyter Notebook**: Integrated development environment
- Libraries:
    - Data Cleaning
    - Data Handling: NumPy, Pandas
    - Visualisation: Matplotlib, Seaborn
    - ML and DL: Scikit-learn, TensorFlow, Keras, PyTorch
    - Oversampling: imblearn
    - Clustering: DBSCAN, KMeans

2.3 Code Execution

1. Launch **Jupyter Notebook** from the Anaconda environment.
2. Navigate to the project folder and open the code file.
3. Run all cells sequentially to execute the entire workflow.

3 Importing Necessary Libraries

```
#import the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve, confusion_matrix
from keras.models import Sequential
from keras.utils import to_categorical
from imblearn.over_sampling import SMOTE
from sklearn.cluster import DBSCAN
import tensorflow as tf
from tensorflow.keras.layers import LSTM, Dense, Dropout, Input, Layer, Activation
from tensorflow.keras.models import Model
import tensorflow as tf
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from tensorflow.keras import layers, models
from sklearn.impute import SimpleImputer
import warnings
warnings.filterwarnings("ignore")
```

Figure 1: Importing Necessary Libraries into Workspace

4 Data Collection

The datasets were sourced from CICIDS2017. The combined dataset contains traffic data from multiple scenarios including DDoS, Port Scans, and benign traffic.

4.1 Dataset Files

- Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv
- Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv
- Friday-WorkingHours-Morning.pcap_ISCX.csv
- Monday-WorkingHours.pcap_ISCX.csv
- Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv
- Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv
- Tuesday-WorkingHours.pcap_ISCX.csv
- Wednesday-workingHours.pcap_ISCX.csv

4.2 Data Importing

Pandas library for Python is used to read the CSV files of the dataset. Importing for the same is given in figure 2 below.

```
#Read the dataset
data1 = pd.read_csv('Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv')
data2 = pd.read_csv('Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv')
data3 = pd.read_csv('Friday-WorkingHours-Morning.pcap_ISCX.csv')
data4 = pd.read_csv('Monday-WorkingHours.pcap_ISCX.csv')
data5 = pd.read_csv('Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv')
data6 = pd.read_csv('Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv')
data7 = pd.read_csv('Tuesday-WorkingHours.pcap_ISCX.csv')
data8 = pd.read_csv('Wednesday-workingHours.pcap_ISCX.csv')
```

Figure 2: Importing the Dataset Files

Columns across datasets were standardised before concatenation. The final dataset contained **common_columns** shared by all datasets.

```
# List of datasets
datasets = [data1, data2, data3, data4, data5, data6, data7, data8]

# Ensure all datasets have the same columns
common_columns = list(set.intersection(*(set(df.columns) for df in datasets)))
datasets = [df[common_columns] for df in datasets]

# Combine all datasets into one DataFrame
df_final = pd.concat(datasets, ignore_index=True)
```

Figure 3: Standardisation of Columns and Concatenation

4.3 Data Cleaning

Getting the count of each attack type in the dataset is performed using the value_counts() for Pandas dataframe as shown in Figure 4.

```
#Count for label encoder
df_final[' Label'].value_counts()
```

```
 Label
BENIGN                        2273097
DoS Hulk                       231073
PortScan                       158930
DDoS                           128027
DoS GoldenEye                   10293
FTP-Patator                      7938
SSH-Patator                      5897
DoS slowloris                    5796
DoS Slowhttptest                 5499
Bot                              1966
Web Attack ? Brute Force         1507
Web Attack ? XSS                  652
Infiltration                       36
Web Attack ? Sql Injection         21
Heartbleed                         11
Name: count, dtype: int64
```

Figure 4: Getting Unique Labels

Statistical description of the dataset can be obtained as shown in Figure 5 below.

```
#Statistical Description of dataset
df_final.describe()
```

Figure 5: Getting Statistical Description of the Dataset

Information of the column data types and the length of the dataset is obtained as below.

```
#Checking the columns information
df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2830743 entries, 0 to 2830742
Data columns (total 79 columns):
 #   Column                       Dtype
---  ------                       -----
 0    PSH Flag Count              int64
 1    Bwd URG Flags               int64
 2    Fwd IAT Mean                float64
 3    Idle Min                    int64
 4   Bwd Packet Length Max        int64
 5    Bwd Packet Length Std       float64
 6   Init_Win_bytes_forward       int64
 7   Bwd IAT Total                int64
 8    Packet Length Std           float64
 9    Min Packet Length           int64
 10   Active Max                  int64
 11  Idle Mean                    float64
 12   Down/Up Ratio               int64
 13   Bwd Packets/s               float64
 14  Flow Bytes/s                 float64
 15   Fwd URG Flags               int64
 16   Packet Length Mean          float64
 17   Packet Length Variance      float64
 18   Fwd Header Length.1         int64
 19   Idle Std                    float64
...
 77   Label                       object
 78   Max Packet Length           int64
dtypes: float64(24), int64(54), object(1)
memory usage: 1.7+ GB
```

Figure 6: Getting the column meta data

Count of the null data for the dataset is obtained using the isnull().sum() operation on the dataframe.

```
#Checking the null value
df_final.isnull().sum()
```

```
 PSH Flag Count           0
 Bwd URG Flags            0
 Fwd IAT Mean             0
 Idle Min                 0
Bwd Packet Length Max     0
                         ..
 CWE Flag Count           0
Bwd Avg Bulk Rate         0
 Bwd Packet Length Min    0
 Label                    0
 Max Packet Length        0
Length: 79, dtype: int64
```

Figure 7: Getting nulls from the dataset

After this, the duplicated rows across the dataset are identified and dropped from the dataset. This is shown in figure 8.

```
#Checking the duplicates
df_final.duplicated().sum()
```

```
308381
```

```
# Drop duplicate rows
df_final = df_final.drop_duplicates()
```

Figure 8: Identifying and removing duplicates

The columns in the dataset are identified as shown below.

```
#Dataset columns
df_final.columns
```

```
Index([' PSH Flag Count', ' Bwd URG Flags', ' Fwd IAT Mean', ' Idle Min',
       'Bwd Packet Length Max', ' Bwd Packet Length Std',
       'Init_Win_bytes_forward', 'Bwd IAT Total', ' Packet Length Std',
       ' Min Packet Length', ' Active Max', 'Idle Mean', ' Down/Up Ratio',
       ' Bwd Packets/s', 'Flow Bytes/s', ' Fwd URG Flags',
       ' Packet Length Mean', ' Packet Length Variance',
       ' Fwd Header Length.1', ' Idle Std', ' Subflow Bwd Bytes',
       'Fwd IAT Total', ' Bwd PSH Flags', ' Bwd IAT Min', ' Fwd IAT Min',
       ' Fwd Avg Packets/Bulk', ' Init_Win_bytes_backward',
       ' Total Length of Bwd Packets', ' Total Fwd Packets',
       ' Average Packet Size', ' Bwd Avg Bytes/Bulk', 'Fwd PSH Flags',
       'Fwd Packets/s', ' Active Min', ' Flow Packets/s', ' SYN Flag Count',
       'Active Mean', ' Flow Duration', ' Bwd IAT Max', ' Bwd IAT Mean',
       ' Fwd Header Length', 'FIN Flag Count', ' Bwd Header Length',
       ' RST Flag Count', ' Total Backward Packets', ' Flow IAT Min',
       ' Fwd IAT Std', ' Fwd Packet Length Max', ' act_data_pkt_fwd',
       ' Active Std', ' Avg Fwd Segment Size', ' ECE Flag Count',
       ' Subflow Fwd Bytes', ' URG Flag Count', ' Flow IAT Std',
       ' Fwd IAT Max', ' Avg Bwd Segment Size', ' Fwd Packet Length Std',
       'Total Length of Fwd Packets', ' Bwd Packet Length Mean',
       ' Destination Port', ' min_seg_size_forward', 'Fwd Avg Bytes/Bulk',
       ' Bwd Avg Packets/Bulk', ' Flow IAT Max', 'Subflow Fwd Packets',
       ' ACK Flag Count', ' Subflow Bwd Packets', ' Flow IAT Mean',
       ' Bwd IAT Std', ' Idle Max', ' Fwd Packet Length Mean',
       ' Fwd Packet Length Min', ' Fwd Avg Bulk Rate', ' CWE Flag Count',
       'Bwd Avg Bulk Rate', ' Bwd Packet Length Min', ' Label',
       ' Max Packet Length'],
      dtype='object')
```

Figure 9: Column names in the dataset

The labels names are changed through a mapping function (.map()) as shown below.

```
#Label Mapping
label_mapping = {
    'BENIGN': 'Normal',
    'Bot': 'Botnet',
    'FTP-Patator': 'Brute Force',
    'SSH-Patator': 'Brute Force',
    'DoS Hulk': 'DoS/DDoS',
    'DoS GoldenEye': 'DoS/DDoS',
    'DoS slowloris': 'DoS/DDoS',
    'DoS Slowhttptest': 'DoS/DDoS',
    'DDoS': 'DoS/DDoS',
    'Heartbleed': 'DoS/DDoS',
    'Infiltration': 'Infiltration',
    'PortScan': 'Port Scan',
    'Web Attack \x96 Brute Force': 'Web Attack',
    'Web Attack \x96 XSS': 'Web Attack',
    'Web Attack \x96 Sql Injection': 'Web Attack'
}

df_final[' Label'] = df_final[' Label'].map(label_mapping)
```

Figure 10: Label mapping

5 Exploratory Data Analysis

Several plots are plotted for the exploratory data analysis. Implementation for the same are discussed below.

```
# Violinplot of Packet Length Variance by Label
sns.violinplot(x=' Label', y=' Packet Length Variance', data=df_final)
plt.title('Violinplot of Packet Length Variance by Label')
plt.show()
```

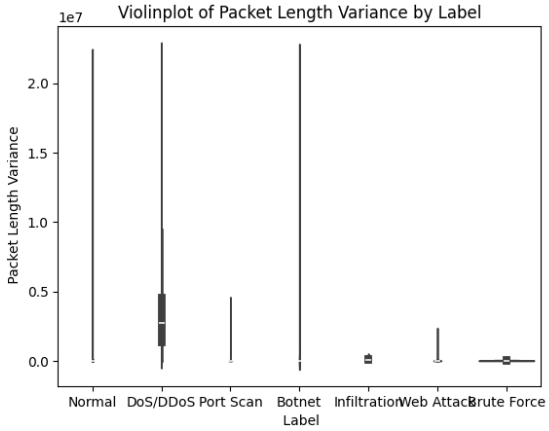

Figure 11: Violin Plot for Packet Length Variance by Label

```
#  Scatter plot - Fwd Packet Length Mean vs. Bwd Packet Length Mean, by Label
sns.scatterplot(data=df_final, x=' Fwd Packet Length Mean', y=' Bwd Packet Length Mean', hue=' Label')
plt.title('Fwd Packet Length Mean vs. Bwd Packet Length Mean by Label')
plt.show()
```

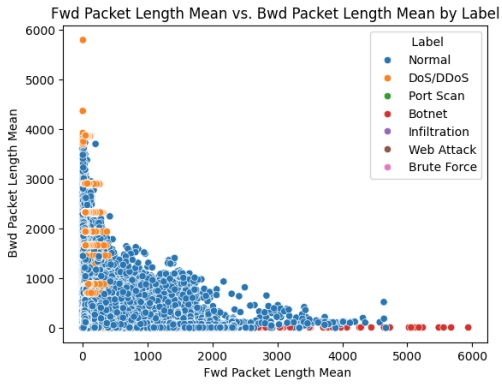

Figure 12: Scatter Plot for Fwd Packet Length Mean vs. Bwd Packet Length Mean by label

The distribution of the class labels in the dataset is obtained as shown in Figure 13.

```
#Countplot for Label
sns.countplot(data=df_final, x=' Label',palette='hls')
plt.title('Distribution of Label')
plt.xticks(rotation=90)
```

```
([0, 1, 2, 3, 4, 5, 6],
 [Text(0, 0, 'Normal'),
  Text(1, 0, 'DoS/DDoS'),
  Text(2, 0, 'Port Scan'),
  Text(3, 0, 'Botnet'),
  Text(4, 0, 'Infiltration'),
  Text(5, 0, 'Web Attack'),
  Text(6, 0, 'Brute Force')])
```

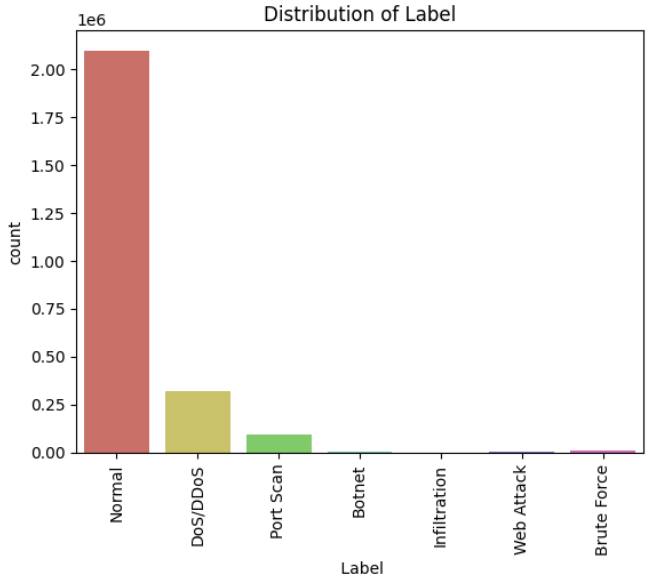

Figure 13: Getting Label Distribution

The KDE Plot for the Average Packet Size is plotted using the Seaborn library as shown in Figure 14.

```
#  Distribution of Average Packet Size - KDE plot
sns.kdeplot(df_final[' Average Packet Size'], fill=True)
plt.title('Distribution of Average Packet Size')
plt.show()
```

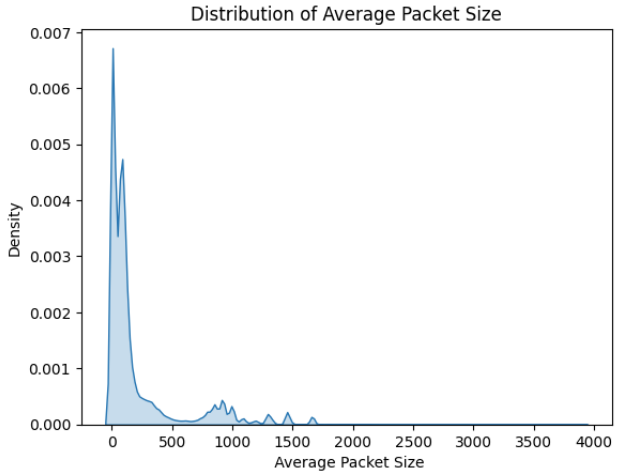

Figure 14: KDE Plot for Average Packet Size

6 Preprocessing

Labels in the dataset are encoded for Supervised Learning through LSTM, Attention LSTM and Transformer using the Label Encoder in Sklearn. Its implementation is shown in Figure 15 below.

```
# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the 'Label' column
df_final[' Label'] = label_encoder.fit_transform(df_final[' Label'])
```

Figure 15: Implementation of Label Encoding

Correlation matrix is plotted to identify multicollinearity in the dataset. This is implemented by generating heatmap of the correlation coefficients for the features.

```
# Divide the dataset into chunks of 20 columns
chunks = [df_final.iloc[:, i:i+20] for i in range(0, df_final.shape[1], 20)]

# Loop through each chunk and plot the correlation heatmap individually
for i, chunk in enumerate(chunks):
    corr = chunk.corr()
    plt.figure(figsize=(20, 8))

    # Plot the heatmap
    sns.heatmap(corr, annot=True, cmap='hsv', fmt='.2f', vmin=-1, vmax=1)

    # Adjust the title to reflect the correct number of columns
    start_col = i * 20 + 1
    end_col = (i + 1) * 20
    # If the last chunk has fewer than 20 columns
    if end_col > df_final.shape[1]:
        end_col = df_final.shape[1]

    plt.title(f'Correlation Heatmap - Columns {start_col} to {end_col}', fontsize=14)

    plt.xticks(rotation=90)
    plt.tight_layout()
    plt.show()  # Show each plot individually
```

Figure 16: Correlation matrix generation

To ensure faster processing with low resource requirements 50000 samples of the total dataset are used for further processing.

```
#taking 50000 rows
df_final_subset = df_final.sample(n=50000, random_state=42)
```

Figure 17: Getting subset of the data

The dependent and independent variables from the data are separated by dropping the label column from the dataset.

```
#Seperating dependent and independent variable
X = df_final_subset.drop(columns=[' Label'])
y = df_final_subset[' Label']
```

Figure 18: Separating Dependent and Independent Columns

The correlation coefficients of the dataset features are thresholded at 0.9 to remove multicollinearity.

```
# Identifying highly correlated features
correlation_matrix = X.corr()
upper_tri = correlation_matrix.where(np.triu(np.ones(correlation_matrix.shape), k=1).astype(bool))
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > 0.9)]

# Drop the highly correlated features
X_clean = X.drop(columns=to_drop)
```

Figure 19: Thresholding the Correlation Matrix

The data is then split into training and testing set using the Sklearn library's train_test_split.

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 20: Splitting the Dataset

Large or infinite values in the dataset are replaced with 0.

```
# Replace infinite values with 0 or any other suitable value
X_train[np.isinf(X_train)] = 0
X_test[np.isinf(X_test)] = 0
```

Figure 21: Zeroing the large values

SimpleImputer from the Sklearn is then used to replace the missing values in the dataset by their corresponding column means.

```
# Impute missing values
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)
```

Figure 22: Replacing nulls in the dataset by column means

Features are then normalized through standardization.

```
# Scale the features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Figure 23: Standardisation of the Features

The labels are balanced using the SMOTE technique as shown in Figure 24.

```
# Apply SMOTE to handle class imbalance
smote = SMOTE(random_state=42, k_neighbors=3)
X_train_res, y_train_res = smote.fit_resample(X_train_scaled, y_train)
```

Figure 24: Application of SMOTE

Balanced class counts are then plotted.

```
sns.countplot(x=y_train_res, palette='pastel')
```

```
<Axes: xlabel=' Label', ylabel='count'>
```

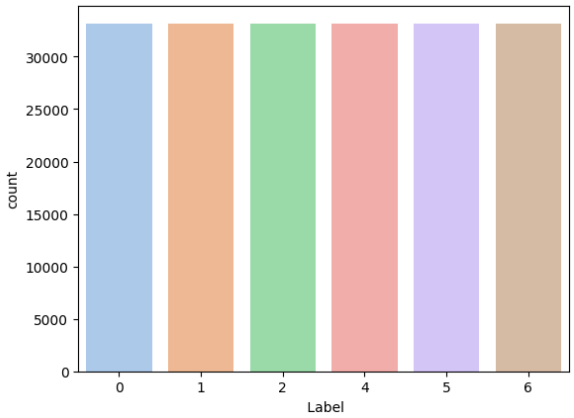

Figure 25: Balanced Classes after SMOTE

The feature set is then reshaped for applicability for LSTM and Attention LSTM and models

```
#  Reshape data for LSTM model
X_train_res = X_train_res.reshape((X_train_res.shape[0], 1, X_train_res.shape[1]))
X_test_scaled = X_test_scaled.reshape((X_test_scaled.shape[0], 1, X_test_scaled.shape[1]))
```

Figure 26: Reshaping the feature set

The classes are then one-hot encoded using the to_categorical function of the keras utils module.

```
y_train_cat = to_categorical(y_train_res)
#y_test_cat = to_categorical(y_test)
```

Figure 27: One-hot encoding the labels

7 Modelling

Five different models are used in the study. These are:

1. LSTM
2. Attention LSTM
3. Transformer
4. Kmeans
5. DBSCAN

Their implementations are discussed below.

```python
# Build the LSTM model
model = Sequential()

# LSTM layer with 50 units
model.add(LSTM(units=50, activation='relu', input_shape=(X_train_res.shape[1], X_train_res.shape[2])))
model.add(Dropout(0.2))

# Fully connected layer
model.add(Dense(units=y_train_cat.shape[1], activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train_res, y_train_cat, epochs=10, batch_size=128, validation_data=(X_test_scaled, y_test_cat))

#  Make predictions
y_pred = model.predict(X_test_scaled)
y_pred = np.argmax(y_pred, axis=1)

# Calculate metrics
accuracy_1 = accuracy_score(y_test, y_pred)
precision_1 = precision_score(y_test, y_pred, average='macro')  # For multi-class, use average='macro'
recall_1 = recall_score(y_test, y_pred, average='macro')
f1_1 = f1_score(y_test, y_pred, average='macro')
conf_matrix_1 = confusion_matrix(y_test, y_pred)

# Print metrics
print(f"Accuracy: {accuracy_1}")
print(f"Precision: {precision_1}")
print(f"Recall: {recall_1}")
print(f"F1 Score: {f1_1}")
print(f"Confusion Matrix: \n{conf_matrix_1}")
```

Figure 28: LSTM Model Implementation

```python
# Define custom Attention Layer
class Attention(Layer):
    def __init__(self):
        super(Attention, self).__init__()

    def build(self, input_shape):
        self.W = self.add_weight(name='attention_weight', shape=(input_shape[-1], 1), initializer='random_normal', trainable=True)
        self.b = self.add_weight(name='attention_bias', shape=(input_shape[1],), initializer='zeros', trainable=True)
        super(Attention, self).build(input_shape)

    def call(self, inputs):
        # Compute attention scores
        score = tf.matmul(inputs, self.W) + self.b
        attention_weights = tf.nn.softmax(score, axis=1)
        # Apply attention weights to the input features
        output = inputs * attention_weights
        return tf.reduce_sum(output, axis=1)
```

```python
# Build the model
inputs = Input(shape=(X_train_res.shape[1], X_train_res.shape[2]))

# LSTM layer with 50 units
lstm_out = LSTM(units=50, activation='relu', return_sequences=True)(inputs)

# Attention layer
attention_out = Attention()(lstm_out)

# Dropout for regularization
dropout_out = Dropout(0.2)(attention_out)

# Fully connected output layer
output = Dense(units=y_train_cat.shape[1], activation='sigmoid')(dropout_out)

# Compile the model
model = Model(inputs=inputs, outputs=output)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train_res, y_train_cat, epochs=10, batch_size=128, validation_data=(X_test_scaled, y_test_cat))

# Make predictions
y_pred = model.predict(X_test_scaled)
y_pred = np.argmax(y_pred, axis=1)

# Calculate metrics
accuracy_2 = accuracy_score(y_test, y_pred)
precision_2 = precision_score(y_test, y_pred, average='macro')
recall_2 = recall_score(y_test, y_pred, average='macro')
f1_2 = f1_score(y_test, y_pred, average='macro')
conf_matrix_2 = confusion_matrix(y_test, y_pred)

# Print metrics
print(f"Accuracy: {accuracy_2}")
print(f"Precision: {precision_2}")
print(f"Recall: {recall_2}")
print(f"F1 Score: {f1_2}")
print(f"Confusion Matrix: \n{conf_matrix_2}")
```

Figure 29: Attention LSTM Model Implementation

```python
# Model Building
def create_transformer_model(input_shape):
    inputs = layers.Input(shape=input_shape)

    # Transformer Encoder Layer
    x = layers.MultiHeadAttention(num_heads=8, key_dim=64)(inputs, inputs)
    x = layers.Dropout(0.1)(x)
    x = layers.LayerNormalization(epsilon=1e-6)(x)
    x = layers.GlobalAveragePooling1D()(x)

    # Fully connected layer
    x = layers.Dense(64, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(y_train_cat.shape[1], activation='sigmoid')(x)

    model = models.Model(inputs, outputs)
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    return model

# Create and compile the model
model = create_transformer_model((X_train_res.shape[1], X_train_res.shape[2]))

# Train the model
history = model.fit(X_train_res, y_train_cat, epochs=10, batch_size=64, validation_data=(X_test_scaled, y_test_cat))

# Evaluate the model
y_pred = model.predict(X_test_scaled)
y_pred = np.argmax(y_pred, axis=1)

# Metrics calculation for multiclass
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Output the metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

Figure 30: Transformer Model Implementation

```python
# Set the number of clusters
num_clusters = len(np.unique(y_train))

#  Initialize and fit K-Means
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
kmeans.fit(X_train_scaled)

# Predict the cluster labels
y_kmeans_pred = kmeans.predict(X_train_scaled)

# Evaluate the clustering performance

# Accuracy
accuracy_4 = accuracy_score(y_train, y_kmeans_pred)
print(f"Accuracy: {accuracy:.4f}")

# Precision, Recall, F1 Score
precision_4 = precision_score(y_train, y_kmeans_pred, average='weighted', zero_division=0)
recall_4 = recall_score(y_train, y_kmeans_pred, average='weighted', zero_division=0)
f1_4 = f1_score(y_train, y_kmeans_pred, average='weighted', zero_division=0)

print(f"Precision: {precision_4:.4f}")
print(f"Recall: {recall_4:.4f}")
print(f"F1 Score: {f1_4:.4f}")

# Generate and display the confusion matrix
cm = confusion_matrix(y_train, y_kmeans_pred)
cm
```

Figure 31: Kmeans Clustering Implementation

```
# Initialize and fit DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan.fit(X_train_scaled)

#  Predict the cluster labels
y_dbscan_pred = dbscan.labels_

#  Evaluate the clustering performance

# Convert noise points (-1) to a unique label for evaluation purposes
y_dbscan_pred_no_noise = np.where(y_dbscan_pred == -1, -1, y_dbscan_pred)

# Accuracy
accuracy_5 = accuracy_score(y_train, y_dbscan_pred_no_noise)
print(f"Accuracy: {accuracy_5:.4f}")

# Precision, Recall, F1 Score
precision_5 = precision_score(y_train, y_dbscan_pred_no_noise, average='weighted', zero_division=0)
recall_5 = recall_score(y_train, y_dbscan_pred_no_noise, average='weighted', zero_division=0)
f1_5 = f1_score(y_train, y_dbscan_pred_no_noise, average='weighted', zero_division=0)

print(f"Precision: {precision_5:.4f}")
print(f"Recall: {recall_5:.4f}")
print(f"F1 Score: {f1_5:.4f}")

# Generate and display the confusion matrix
cm = confusion_matrix(y_train, y_dbscan_pred_no_noise)
cm
```

Figure 32: DBSCAN Implementation

8 Evaluation

Models are evaluated and compared based on the accuracy, precision, recall, f1-score and confusion matrix. Its comparison is given below.

```
# Model Evaluation Metrics
metrics = {
    'Model': ['Attention LSTM', 'LSTM', 'Transformer', 'DBSCAN', 'KMeans'],
    'Accuracy': [accuracy_2, accuracy_1, accuracy, accuracy_5, accuracy_4],
    'Precision (weighted)': [precision_2, precision_1, precision, precision_5, precision_4],
    'Recall (weighted)': [recall_2, recall_1, recall, recall_5, recall_4],
    'F1-Score (weighted)': [f1_2, f1_1, f1, f1_5, f1_4]
}

# Create DataFrame
comparison_df = pd.DataFrame(metrics)

# Print the comparison table
print(comparison_df)
```

Figure 33: Model Evaluation

9 References

- *Guide* (no date). https://www.tensorflow.org/guide.
- *scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation* (no date). https://scikit-learn.org/.
- *IDS 2017 | Datasets | Research | Canadian Institute for Cybersecurity | UNB* (no date). https://www.unb.ca/cic/datasets/ids-2017.html.
- *Welcome to imbalanced-learn documentation!* (no date). https://imbalanced-learn.org/.