from google.colab import drive

drive.mount('/content/drive')

# This Python 3 environment comes with many helpful analytics libraries installed

# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python

# For example, here's several helpful packages to load

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

import seaborn as sns

import time

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.cluster import KMeans

from sklearn.metrics import accuracy\_score, confusion\_matrix, ConfusionMatrixDisplay, RocCurveDisplay, auc

from sklearn.metrics import precision\_score, recall\_score, f1\_score, roc\_curve

from sklearn.feature\_selection import RFE

from sklearn.tree import DecisionTreeClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.feature\_selection import mutual\_info\_classif

import xgboost as xgb

from lightgbm import LGBMClassifier

import keras

from numpy import array

# Input data files are available in the read-only "../input/" directory

# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os

\*\*Concatinate train and test to make some data precessing and cleaning before splitting\*\*

df\_train = pd.read\_csv("/content/drive/MyDrive/code/UNSW\_NB15\_testing-set.csv")

df\_test = pd.read\_csv("/content/drive/MyDrive/code/UNSW\_NB15\_training-set.csv")

df = pd.concat([df\_train, df\_test])

# information about the dataset

pd.set\_option('display.max\_columns', None)

df

\*\*Get all Data with "Backdoor Attack"

and get 2329 Normal data as the same number as the Backdoor Attack\*\*

# Filter the DataFrame to include only "Backdoor" and "Normal" rows

backdoor\_data = df[df['attack\_cat'] == 'Backdoor']

normal\_data = df[df['attack\_cat'] == 'Normal']

# Randomly sample 2329 rows from the "Normal" category

sampled\_normal\_data = normal\_data.sample(n=2329, random\_state=42) # You can adjust the random\_state if needed

# Combine the "Backdoor" and sampled "Normal" data

balanced\_data = pd.concat([backdoor\_data, sampled\_normal\_data])

# Shuffle the balanced data

balanced\_data\_shuffled = balanced\_data.sample(frac=1, random\_state=42) # Shuffle with random\_state for reproducibility

unique\_sttl\_values = balanced\_data\_shuffled['state'].unique()

unique\_sttl\_values

\*\*Check Any missing Value\*\*

# Check for missing values in the dataset

missing\_values = balanced\_data\_shuffled.isnull().sum()

print("Missing values in the dataset:")

print(missing\_values)

target\_variable = balanced\_data\_shuffled['label']

plt.figure(figsize=(12, 8))

sns.boxplot(x=target\_variable, y=balanced\_data\_shuffled['sttl'], data=balanced\_data\_shuffled)

plt.title('Box Plot for sttl')

plt.show()

plt.hist(backdoor\_data['sttl'], bins=20, color='red', alpha=0.7, label='Backdoor', edgecolor='black')

plt.hist(normal\_data['sttl'], bins=20, color='blue', alpha=0.5, label='Normal', edgecolor='black')

plt.title('Histogram of sttl')

plt.xlabel('Total TTL')

plt.ylabel('Frequency')

plt.legend()

plt.tight\_layout()

plt.show()

plt.hist(backdoor\_data['dttl'], bins=20, color='red', alpha=0.7, label='Backdoor', edgecolor='black')

plt.hist(normal\_data['dttl'], bins=20, color='blue', alpha=0.5, label='Normal', edgecolor='black')

plt.title('Histogram of dttl')

plt.xlabel('Total TTL')

plt.ylabel('Frequency')

plt.legend()

plt.tight\_layout()

plt.show()

normal\_data = balanced\_data\_shuffled[balanced\_data\_shuffled['label'] == 0]

backdoor\_data = balanced\_data\_shuffled[balanced\_data\_shuffled['label'] == 1]

# Scatter plot for Duration vs Packet Count

plt.figure(figsize=(10, 6))

# Backdoor traffic

plt.scatter(backdoor\_data['dur'], backdoor\_data['sbytes'], color='red', label='Backdoor', alpha=0.7)

# Normal traffic

plt.scatter(normal\_data['dur'], normal\_data['sbytes'], color='blue', label='Normal', alpha=0.5)

plt.title('Scatter Plot - Duration vs sbytes')

plt.xlabel('Duration')

plt.ylabel('Packet Count')

plt.legend()

plt.grid(True)

plt.show()

normal\_data = balanced\_data\_shuffled[balanced\_data\_shuffled['label'] == 0]['proto'].value\_counts()

print ( balanced\_data\_shuffled[balanced\_data\_shuffled['label'] == 1]['proto'].unique())

backdoor\_data = balanced\_data\_shuffled[balanced\_data\_shuffled['label'] == 1]['proto'].value\_counts()

# Calculate percentages

total\_normal = normal\_data.sum()

normal\_percentages = [(count / total\_normal) \* 100 for count in normal\_data]

total\_backdoor = backdoor\_data.sum()

backdoor\_percentages = [(count / total\_backdoor) \* 100 for count in backdoor\_data]

# Create Pie Charts

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))

# Pie chart for Normal Traffic

axes[0].pie(normal\_percentages, labels=normal\_data.index, autopct='%1.1f%%', startangle=90)

axes[0].set\_title('Normal Traffic Distribution')

# Pie chart for Backdoor Traffic

axes[1].pie(backdoor\_percentages, labels=backdoor\_data.index, autopct='%1.1f%%', startangle=90)

axes[1].set\_title('Backdoor Traffic Distribution')

plt.show()

target\_variable = balanced\_data\_shuffled['label']

plt.figure(figsize=(12, 8))

sns.violinplot(x=target\_variable, y=balanced\_data\_shuffled['sttl'], data=balanced\_data\_shuffled)

plt.title('Box Plot for sttl')

plt.show()

unique\_sttl\_values = balanced\_data\_shuffled['ackdat'].unique()

unique\_sttl\_values

target\_variable = balanced\_data\_shuffled['label']

plt.figure(figsize=(12, 8))

sns.boxplot(x=target\_variable, y=balanced\_data\_shuffled['ct\_state\_ttl'], data=balanced\_data\_shuffled)

plt.title('Box Plot for ct\_state\_ttl')

plt.show()

unique\_sttl\_values = balanced\_data\_shuffled['state'].unique()

unique\_sttl\_values

df.columns

import pandas as pd

import statsmodels.api as sm

# Assuming your DataFrame is named 'df'

# Replace 'YourColumnName' with the actual column names in your DataFrame

# Select the columns of interest

selected\_columns = ['state', 'sttl', 'dttl', 'ct\_state\_ttl']

# Create a new DataFrame with only the selected columns

selected\_df = balanced\_data\_shuffled[selected\_columns]

# Convert 'state' column to numeric

selected\_df['state'] = pd.to\_numeric(selected\_df['state'], errors='coerce')

# Encode categorical variable 'state' into numerical values

selected\_df = pd.get\_dummies(selected\_df, columns=['state'], prefix='state', drop\_first=True)

# Convert all columns to numeric, handle any non-numeric values

selected\_df = selected\_df.apply(pd.to\_numeric, errors='coerce')

# Drop rows with missing values

selected\_df = selected\_df.dropna()

# Add a constant term for the intercept

selected\_df = sm.add\_constant(selected\_df)

# Separate independent variables (X) and dependent variable (y)

X = selected\_df.drop('ct\_state\_ttl', axis=1) # Exclude the dependent variable

y = selected\_df['ct\_state\_ttl']

# Fit the linear regression model

model = sm.OLS(y, X).fit()

# Print the model summary

print(model.summary())

\*\*Drop unecessary Features\*\*

# # Drop features that don't belong to the "Backdoor" attack category

# balanced\_data\_shuffled = balanced\_data\_shuffled.drop(['swin', 'dwin', 'stcpb', 'dtcpb',

# 'trans\_depth', 'tcprtt', 'synack', 'ackdat', 'is\_sm\_ips\_ports',

# 'ct\_flw\_http\_mthd', 'is\_ftp\_login', 'ct\_ftp\_cmd',

# 'attack\_cat','id'], axis=1)

# balanced\_data\_shuffled.info()

# Drop features that don't belong to the "Backdoor" attack category

balanced\_data\_shuffled = balanced\_data\_shuffled.drop(['attack\_cat','id'], axis=1)

balanced\_data\_shuffled.info()

Shuffel

balanced\_data\_shuffled['service'].unique() #Here, we'll deal with the type of service that is '-'

balanced\_data\_shuffled['service']= np.where(balanced\_data\_shuffled['service'] == '-', 'None', balanced\_data\_shuffled['service'])

print(balanced\_data\_shuffled['service'].unique())

balanced\_data\_shuffled['proto'].unique() #This is definitely a categorical feature.

balanced\_data\_shuffled['state'].unique() #Keep it.

def Remove\_dump\_values(data, cols):

for col in cols:

data[col] = np.where(data[col] == '-', 'None', data[col])

return data

cols = balanced\_data\_shuffled.columns

data\_bin = Remove\_dump\_values(balanced\_data\_shuffled, cols)

\*\*I need OneHotEncoding Proto, Service, state\*\*

df\_cat = data\_bin.select\_dtypes(exclude=[np.number])

from sklearn.preprocessing import LabelEncoder

import joblib

# Assuming df\_cat is your DataFrame with categorical columns

for feature in df\_cat.columns:

le = LabelEncoder()

data\_bin[feature] = le.fit\_transform(data\_bin[feature])

# Save the LabelEncoder

encoder\_filename = f'{feature}\_encoder.joblib'

joblib.dump(le, encoder\_filename)

# Now you can use encoder\_filename to load the encoder later

data\_bin.head()

\*\*Corrolation Matrix\*\*

sns.heatmap(data\_bin.corr())

plt.show()

columns = data\_bin.columns.tolist()

corr = data\_bin.corr()

correlated\_vars = []

for i in range(len(columns) - 1):

for j in range(i+1, len(columns)):

if corr[columns[i]][columns[j]] > 0.95:

print(columns[i], columns[j], corr[columns[i]][columns[j]])

correlated\_vars.append(columns[j])

data\_bin = data\_bin.drop(columns=correlated\_vars)

data\_bin.info()

\*\*Mutual Information\*\*

x = data\_bin.drop(columns=['label'])

y = data\_bin['label']

mi\_scores = mutual\_info\_classif(x, y)

mi\_df = pd.DataFrame(mi\_scores, index=x.columns, columns=['MI Score'])

mi\_df = mi\_df.sort\_values(by='MI Score', ascending=False)

print(mi\_df)

mi\_df\_sorted = mi\_df.sort\_values(by='MI Score', ascending=True)

plt.figure(figsize=(10, 8))

sns.barplot(x='MI Score', y=mi\_df\_sorted.index, data=mi\_df\_sorted, palette='mako')

plt.xlabel('Mutual Information Score')

plt.ylabel('Features')

plt.title('Mutual Information Scores per Feature')

plt.tight\_layout()

plt.show()

data\_bin = data\_bin.drop(['trans\_depth','ct\_src\_ltm','tcprtt','rate','ct\_state\_ttl'], axis=1)

data\_bin.info()

\*\*Splitting training and testing sets\*\*

selected\_columns = ['dur', 'proto', 'spkts', 'dpkts', 'sbytes', 'dbytes',

'dload', 'sloss', 'dinpkt', 'ackdat', 'smean', 'dmean', 'ct\_srv\_src',

'ct\_dst\_src\_ltm', 'ct\_srv\_dst','label']

dtt = data\_bin[selected\_columns]

X = dtt.drop(columns=['label'])

feature\_list = list(X.columns)

X = np.array(X)

y = dtt['label']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25)

print("Training set:", len(X\_train))

print("Testing set:", len(X\_test))

# \*\*Algorithms\*\*

from sklearn.ensemble import RandomForestClassifier

params = {

'max\_depth': 10,

'objective': 'multi:softmax', # error evaluation for multiclass training

'num\_class': 2, # Number of classes

'n\_gpus': 4

}

models = {}

models['Naive Bayes'] = DecisionTreeClassifier()

#models['LGM'] =random\_forest= LGBMClassifier()

models['Random Forest '] = RandomForestClassifier()

models['xgboost'] = xgb\_classifier = xgb.XGBClassifier(\*\*params)

train\_score, accuracy, precision, recall, training\_time, y\_pred = {}, {}, {}, {}, {}, {}

for key in models.keys():

start\_time = time.time()

models[key].fit(X\_train, y\_train)

training\_time[key] = time.time() - start\_time

y\_pred[key] = models[key].predict(X\_test)

train\_score[key] = models[key].score(X\_train, y\_train)

accuracy[key] = models[key].score(X\_test, y\_test)

precision[key] = precision\_score(y\_test, y\_pred[key])

recall[key] = recall\_score(y\_test, y\_pred[key])

df\_models = pd.DataFrame(index=models.keys(), columns=['Training score', 'Accuracy', 'Precision', 'Recall', 'Training time'])

df\_models['Training score'] = train\_score.values()

df\_models['Accuracy'] = accuracy.values()

df\_models['Precision'] = precision.values()

df\_models['Recall'] = recall.values()

df\_models['Training time'] = training\_time.values()

df\_models

scaler = StandardScaler().fit(X\_train)

X\_train = scaler.transform(X\_train)

X\_test = scaler.transform(X\_test)

from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential

from keras.layers import LSTM, Dense, Dropout

from keras.optimizers import Adam

from keras.callbacks import EarlyStopping

# Assuming X\_train, y\_train, X\_val, and y\_val are defined elsewhere in your code

# Preprocess data

scaler = MinMaxScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

joblib.dump(scaler, 'scaler.joblib')

# Reshape data for LSTM

X\_train\_reshaped = X\_train\_scaled.reshape(X\_train\_scaled.shape[0], 1, X\_train\_scaled.shape[1])

# Build LSTM model

lstm\_model=Sequential()

lstm\_model.add(LSTM(units=100, return\_sequences=True, input\_shape=(X\_train\_reshaped.shape[1], X\_train\_reshaped.shape[2])))

lstm\_model.add(LSTM(units=100))

lstm\_model.add(Dropout(0.2)) # Adding dropout for regularization

lstm\_model.add(Dense(1, activation='sigmoid'))

# Compile the model with an adaptive learning rate

optimizer = Adam(learning\_rate=0.001)

lstm\_model.compile(optimizer=optimizer, loss='binary\_crossentropy', metrics=['accuracy'])

# Implement early stopping

early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)

# Train the LSTM model

start\_time = time.time()

history = lstm\_model.fit(X\_train\_reshaped, y\_train, epochs=100, batch\_size=32, callbacks=[early\_stopping])

end\_time = time.time()

# Evaluate the model on the test set

X\_test\_scaled = scaler.transform(X\_test)

X\_test\_reshaped = X\_test\_scaled.reshape(X\_test\_scaled.shape[0], 1, X\_test\_scaled.shape[1])

test\_loss, test\_accuracy = lstm\_model.evaluate(X\_test\_reshaped, y\_test)

# Calculate additional metrics

y\_pred = lstm\_model.predict(X\_test\_reshaped)

y\_pred\_binary = (y\_pred > 0.5).astype(int) # Assuming binary

precision = precision\_score(y\_test, y\_pred\_binary)

recall = recall\_score(y\_test, y\_pred\_binary)

# Save the trained LSTM model to a file

model\_filename = 'lstm\_model.h5'

lstm\_model.save(model\_filename)

print(f"Trained LSTM model saved to {model\_filename}")

from keras.models import load\_model

# Load the saved LSTM model

loaded\_model = load\_model('lstm\_model.h5')

loaded\_scaler = joblib.load('scaler.joblib')

# Confusion Matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_binary)

metrics\_df = pd.DataFrame({

'Metric': ['Accuracy', 'Precision', 'Recall'],

'Score': [accuracy, precision, recall]

})

# Display Confusion Matrix

plt.figure(figsize=(6, 6))

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=["0", "1"], yticklabels=["0", "1"])

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("True")

plt.show()

print("Metrics:")

print(metrics\_df)