**### GBC with an enhancement with RNN-FCNN**

# Importing required libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import metrics

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

from sklearn.ensemble import GradientBoostingClassifier

import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dense, LSTM, Dropout, concatenate, BatchNormalization

from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

import warnings

warnings.filterwarnings('ignore')

# Loading data into dataframe

data = pd.read\_csv("phishing.csv")

# Drop the index column

data = data.drop(['Index'], axis=1)

# Split the dataset into dependent and independent features

X = data.drop(["class"], axis=1)

y = data["class"]

# Ensure that the target labels are correctly encoded

y = y.apply(lambda x: 1 if x == 1 else 0)

# Split the dataset into train and test sets: 80-20 split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Reshape data for LSTM input (samples, time steps, features)

X\_train\_seq = np.expand\_dims(X\_train, axis=1)

X\_test\_seq = np.expand\_dims(X\_test, axis=1)

# Input layer for FCNN

input\_fcnn = Input(shape=(X\_train.shape[1],))

# FCNN layers

fcnn = Dense(256, activation='relu')(input\_fcnn)

fcnn = BatchNormalization()(fcnn)

fcnn = Dropout(0.5)(fcnn)

fcnn = Dense(128, activation='relu')(fcnn)

fcnn = BatchNormalization()(fcnn)

fcnn = Dropout(0.5)(fcnn)

fcnn = Dense(64, activation='relu')(fcnn)

fcnn = BatchNormalization()(fcnn)

fcnn = Dropout(0.5)(fcnn)

# Input layer for RNN

input\_rnn = Input(shape=(X\_train\_seq.shape[1], X\_train\_seq.shape[2]))

# RNN layers

rnn = LSTM(100, activation='relu', return\_sequences=True)(input\_rnn)

rnn = Dropout(0.2)(rnn)

rnn = LSTM(50, activation='relu')(rnn)

rnn = Dropout(0.2)(rnn)

# Combine FCNN and RNN outputs

combined = concatenate([fcnn, rnn])

# Final dense layer

output = Dense(1, activation='sigmoid')(combined)

# Define the model

model = Model(inputs=[input\_fcnn, input\_rnn], outputs=output)

# Compile the model

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

# Callbacks for learning rate reduction and early stopping

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.2, patience=5, min\_lr=0.0001)

early\_stopping = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)

# Train the model

history = model.fit([X\_train, X\_train\_seq], y\_train, epochs=100, batch\_size=32, validation\_data=([X\_test, X\_test\_seq], y\_test), callbacks=[reduce\_lr, early\_stopping])

# Evaluate the RNN-FCNN hybrid model

acc\_train\_hybrid = model.evaluate([X\_train, X\_train\_seq], y\_train, verbose=0)

acc\_test\_hybrid = model.evaluate([X\_test, X\_test\_seq], y\_test, verbose=0)

print("Hybrid Model : Accuracy on training Data: {:.3f}".format(acc\_train\_hybrid[1]))

print("Hybrid Model : Accuracy on test Data: {:.3f}".format(acc\_test\_hybrid[1]))

# Get hybrid model predictions

train\_pred\_prob\_hybrid = model.predict([X\_train, X\_train\_seq])

test\_pred\_prob\_hybrid = model.predict([X\_test, X\_test\_seq])

# Add hybrid model predictions as new features to the dataset

X\_train\_enhanced = np.hstack((X\_train, train\_pred\_prob\_hybrid))

X\_test\_enhanced = np.hstack((X\_test, test\_pred\_prob\_hybrid))

# Gradient Boosting Classifier Model

gbc = GradientBoostingClassifier(max\_depth=4, learning\_rate=0.7)

# Fit the Gradient Boosting Classifier on the enhanced dataset

gbc.fit(X\_train\_enhanced, y\_train)

# Evaluate the enhanced Gradient Boosting Classifier model

acc\_train\_gbc\_enhanced = gbc.score(X\_train\_enhanced, y\_train)

acc\_test\_gbc\_enhanced = gbc.score(X\_test\_enhanced, y\_test)

print("Enhanced Gradient Boosting Classifier : Accuracy on training Data: {:.3f}".format(acc\_train\_gbc\_enhanced))

print("Enhanced Gradient Boosting Classifier : Accuracy on test Data: {:.3f}".format(acc\_test\_gbc\_enhanced))

# Plotting the confusion matrix for the enhanced model

y\_test\_gbc\_enhanced = gbc.predict(X\_test\_enhanced)

cm = metrics.confusion\_matrix(y\_test, y\_test\_gbc\_enhanced)

sns.heatmap(cm, annot=True, fmt="d", cmap="viridis", xticklabels=["Legitimate", "Phishing"], yticklabels=["Legitimate", "Phishing"])

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.title('Confusion Matrix for Enhanced Gradient Boosting Classifier')

plt.show()