**SAMPLE CODE**

from tkinter import \*

import tkinter

from tkinter import filedialog

from tkinter.filedialog import askopenfilename

import seaborn as sns

from sklearn.metrics import accuracy\_score

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

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import confusion\_matrix

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier

import os

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import MinMaxScaler

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

import matplotlib.pyplot as plt

from ABC import ABC

from SwarmPackagePy import testFunctions as tf

from sklearn.svm import SVC

from keras.models import Sequential

from keras.layers.core import Dense,Activation,Dropout, Flatten

from keras.utils.np\_utils import to\_categorical

from keras.callbacks import ModelCheckpoint

import os

import pickle

main = tkinter.Tk()

main.title("Robust and Secure Data Transmission Using Artificial Intelligence Techniques in Ad-Hoc Networks")

main.geometry("1200x1200")

global filename

global X, Y

global X\_train, X\_test, y\_train, y\_test

global throughput

global pdr

global delay

global classifier, class\_labels, dataset, label\_encoder, scaler

def uploadDataset():

global filename, class\_labels, dataset

filename = filedialog.askopenfilename(initialdir="AODVDataset")

pathlabel.config(text=filename)

text.delete('1.0', END)

text.insert(END,filename+" loaded\n\n")

dataset = pd.read\_csv(filename)

text.insert(END,str(dataset))

class\_labels = np.unique(dataset['Label'])

label = dataset.groupby('Label').size()

label.plot(kind="bar")

plt.title("Different Attacks Found in Dataset Graph")

plt.xlabel("Attack Name")

plt.ylabel("Count")

plt.show()

def preprocessDataset():

global dataset, label\_encoder, X, Y, X\_train, X\_test, y\_train, y\_test, scaler

text.delete('1.0', END)

dataset.fillna(0, inplace = True)

label\_encoder = []

columns = dataset.columns

types = dataset.dtypes.values

for i in range(len(types)):

name = types[i]

if name == 'object': #finding column with object type

le = LabelEncoder()

print(columns[i])

dataset[columns[i]] = pd.Series(le.fit\_transform(dataset[columns[i]].astype(str)))#encode all str columns to numeric

label\_encoder.append(le)

text.insert(END,str(dataset)+"\n\n")

dataset = dataset.values

X = dataset[:,0:dataset.shape[1]-1]

Y = dataset[:,dataset.shape[1]-1]

scaler = MinMaxScaler(feature\_range = (0, 1)) #use to normalize training features

X = scaler.fit\_transform(X)

#function which will calculate all metrics and plot confusion matrix

def calculateMetrics(predict, y\_test, algorithm):

global class\_labels

p = precision\_score(y\_test, predict,average='macro') \* 100

r = recall\_score(y\_test, predict,average='macro') \* 100

a = accuracy\_score(y\_test,predict)\*100

conf\_matrix = confusion\_matrix(y\_test, predict)

throughput.append(a)

pdr.append(p)

delay.append(100 - r)

text.insert(END,algorithm+' Throughput : '+str(a)+"\n")

text.insert(END,algorithm+' PDR : '+str(p)+"\n")

text.insert(END,algorithm+' Delay : '+str(100 - r)+"\n\n")

plt.figure(figsize =(6, 4))

ax = sns.heatmap(conf\_matrix, xticklabels = class\_labels, yticklabels = class\_labels, annot = True, cmap="viridis" ,fmt ="g");

ax.set\_ylim([0,len(class\_labels)])

plt.title(algorithm+" Confusion matrix")

plt.ylabel('True class')

plt.xlabel('Predicted class')

plt.show()

def runPropose():

text.delete('1.0', END)

global X, Y, throughput, pdr, delay

delay = []

throughput = []

pdr = []

alh = ABC(X, tf.easom\_function, -10, 10, 2, 20)

Gbest = np.asarray(alh.get\_Gbest())

in\_mask = [True if i > 0 else False for i in Gbest]

in\_mask = np.asarray(in\_mask)

X\_selected\_features = X[:,in\_mask==1]

svm\_cls = SVC(probability=True)

svm\_cls.fit(X\_selected\_features, Y)

Y1 = to\_categorical(Y)

X\_selected\_features = np.reshape(X\_selected\_features, (X\_selected\_features.shape[0], X\_selected\_features.shape[1], 1))

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_selected\_features, Y1, test\_size=0.2)

ann\_model = Sequential()

ann\_model.add(Flatten(input\_shape=[X\_train.shape[1],X\_train.shape[2]]))

ann\_model.add(Dense(300, activation="relu"))

ann\_model.add(Dense(100, activation="relu"))

ann\_model.add(Dense(y\_train.shape[1], activation="softmax"))

ann\_model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

if os.path.exists("model/model\_weights.hdf5") == False:

model\_check\_point = ModelCheckpoint(filepath='model/model\_weights.hdf5', verbose = 1, save\_best\_only = True)

hist = ann\_model.fit(X\_train, y\_train, batch\_size = 32, epochs = 350, validation\_data=(X\_test, y\_test), callbacks=[model\_check\_point], verbose=1)

f = open('model/history.pckl', 'wb')

pickle.dump(hist.history, f)

f.close()

else:

ann\_model.load\_weights("model/model\_weights.hdf5")

predict = ann\_model.predict(X\_test)

predict = np.argmax(predict, axis=1)

testY = np.argmax(y\_test, axis=1)

calculateMetrics(predict, testY, "Propose AODV with ABC, SVM & ANN")

def runRF():

global X, Y, classifier

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2)

rf = RandomForestClassifier()

rf.fit(X\_train, y\_train)

predict = rf.predict(X\_test)

classifier = rf

calculateMetrics(predict, y\_test, "Random Forest")

def runDT():

global X, Y, classifier

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2)

dt = DecisionTreeClassifier()

dt.fit(X\_train, y\_train)

predict = dt.predict(X\_test)

calculateMetrics(predict, y\_test, "DecisionTreeClassifier")

def graph():

#now plot accuracy and other metrics comparison graph

df = pd.DataFrame([['Propose ABC, SVM & ANN','Throughput',throughput[0]],['Propose ABC, SVM & ANN','PDR',pdr[0]],['Propose ABC, SVM & ANN','Delay',delay[0]],

['Random Forest','Throughput',throughput[1]],['Random Forest','PDR',pdr[1]],['Random Forest','Delay',delay[1]],

['Decision Tree','Throughput',throughput[2]],['Decision Tree','PDR',pdr[2]],['Decision Tree','Delay',delay[2]],

],columns=['Parameters','Algorithms','Value'])

df.pivot("Parameters", "Algorithms", "Value").plot(kind='bar')

plt.title("All Algorithms Performance Graph")

plt.show()

def predict():

global scaler, classifier, label\_encoder, class\_labels

text.delete('1.0', END)

filename = filedialog.askopenfilename(initialdir="AODVDataset")

pathlabel.config(text=filename)

dataset = pd.read\_csv(filename)

dataset.fillna(0, inplace = True)

columns = dataset.columns

types = dataset.dtypes.values

index = 0

for i in range(len(types)):

name = types[i]

if name == 'object': #finding column with object type

dataset[columns[i]] = pd.Series(label\_encoder[index].transform(dataset[columns[i]].astype(str)))#encode all str columns to numeric

index = index + 1

dataset = dataset.values

X = scaler.transform(dataset)

predict = classifier.predict(X)

print(predict)

for i in range(len(predict)):

print(predict[i])

text.insert(END,str(dataset[i])+" Predicted Attack =====> "+class\_labels[predict[i]]+"\n\n")

font = ('times', 15, 'bold')

title = Label(main, text='Robust and Secure Data Transmission Using Artificial Intelligence Techniques in Ad-Hoc Networks')

title.config(bg='brown', fg='white')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=5,y=5)

font1 = ('times', 13, 'bold')

uploadButton = Button(main, text="Upload AODV Dataset", command=uploadDataset)

uploadButton.place(x=50,y=100)

uploadButton.config(font=font1)

pathlabel = Label(main)

pathlabel.config(bg='brown', fg='white')

pathlabel.config(font=font1)

pathlabel.place(x=700,y=100)

processButton = Button(main, text="Preprocess Dataset", command=preprocessDataset)

processButton.place(x=400,y=100)

processButton.config(font=font1)

proposeButton = Button(main, text="Run Propose ABC, SVM & ANN Model", command=runPropose)

proposeButton.place(x=50,y=150)

proposeButton.config(font=font1)

rfButton = Button(main, text="Run Random Forest Algorithm", command=runRF)

rfButton.place(x=400,y=150)

rfButton.config(font=font1)

dtButton = Button(main, text="Run Decision Tree Algorithm", command=runDT)

dtButton.place(x=50,y=200)

dtButton.config(font=font1)

graphButton = Button(main, text="Comparison Graph", command=graph)

graphButton.place(x=400,y=200)

graphButton.config(font=font1)

predictButton = Button(main, text="Attack Detection from Test Data", command=predict)

predictButton.place(x=600,y=200)

predictButton.config(font=font1)

font1 = ('times', 12, 'bold')

text=Text(main,height=20,width=130)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=10,y=250)

text.config(font=font1)

main.config(bg='brown')

main.mainloop()