**AUTOMATIC ASSESSMENT OF COMMUNICATION SKILL IN NON-CONVENTIONAL INTERVIEW SETTINGS**

**SOURCE CODE**

from tkinter import \*

import tkinter

from tkinter import filedialog

from tkinter.filedialog import askopenfilename

from PIL import Image

import tensorflow as tf

from tensorflow import keras

import numpy as np

import cv2

import os

from tkinter import messagebox

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

import seaborn as sns

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

from string import punctuation

from nltk.corpus import stopwords

import nltk

from nltk.stem import WordNetLemmatizer

from sklearn.feature\_extraction.text import TfidfVectorizer

import pandas as pd

import pickle

from nltk.stem import PorterStemmer

from sklearn.preprocessing import MinMaxScaler

from xgboost import XGBClassifier

import soundfile

import librosa

from keras.models import model\_from\_json

main = tkinter.Tk()

main.title("Automatic Assessment of communication skills in non-conventional interview settings")

main.geometry("1200x1200")

facial\_expression= ['Anger', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']

speech\_emotion = ['neutral', 'calm', 'happy', 'sad', 'angry', 'fearful', 'disgust', 'surprised']

exp\_model = keras.models.load\_model("model/model\_35\_91\_61.h5")

font\_cv = cv2.FONT\_HERSHEY\_SIMPLEX

face\_cas = cv2.CascadeClassifier('model/haarcascade\_frontalface\_default.xml')

global video, vectorizer, normalize, xgb

stop\_words = set(stopwords.words('english'))

lemmatizer = WordNetLemmatizer()

ps = PorterStemmer()

with open('model/speechmodel.json', "r") as json\_file:

loaded\_model\_json = json\_file.read()

speech\_classifier = model\_from\_json(loaded\_model\_json)

json\_file.close()

speech\_classifier.load\_weights("model/speech\_weights.h5")

speech\_classifier.\_make\_predict\_function()

def cleanPost(doc):

tokens = doc.split()

table = str.maketrans('', '', punctuation)

tokens = [w.translate(table) for w in tokens]

tokens = [word for word in tokens if word.isalpha()]

tokens = [w for w in tokens if not w instop\_words]

tokens = [word for word in tokens if len(word) > 1]

tokens = [ps.stem(token) for token in tokens]

tokens = [lemmatizer.lemmatize(token) for token in tokens]

tokens = ' '.join(tokens)

return tokens

def loadModels():

global vectorizer, normalize, xgb

text.delete('1.0', END)

textdata = np.load("model/X.npy")

Y = np.load("model/Y.npy")

vectorizer = TfidfVectorizer(stop\_words=stop\_words, use\_idf=True, smooth\_idf=False, norm=None, decode\_error='replace', max\_features=3000)

X = vectorizer.fit\_transform(textdata).toarray()

indices = np.arange(X.shape[0])

np.random.shuffle(indices)

X = X[indices]

Y = Y[indices]

normalize = MinMaxScaler()

X = normalize.fit\_transform(X)

print(X.shape)

print(Y)

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

if os.path.exists("model/xgb.txt"):

with open('model/xgb.txt', 'rb') as file:

xgb = pickle.load(file)

file.close()

else:

xgb = XGBClassifier()

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

with open('model/xgb.txt', 'wb') as file:

pickle.dump(xgb, file)

file.close()

predict = xgb.predict(X\_test)

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

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

f = f1\_score(y\_test, predict,average='macro') \* 100

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

text.insert(END,"XGBoost Accuracy : "+str(a)+"\n")

text.insert(END,"XGBoost Precision : "+str(p)+"\n")

text.insert(END,"XGBoost Recall : "+str(r)+"\n")

text.insert(END,"XGBoost FSCORE : "+str(f)+"\n\n")

labels = np.unique(y\_test)

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

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

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

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

plt.title("XGBoost Communication SKills Score Prediction Confusion Matrix Graph")

plt.ylabel('True class')

plt.xlabel('Predicted class')

plt.show()

def visualAssessment():

text.delete('1.0', END)

counter = 0

confident = 0

confuse = 0

video = cv2.VideoCapture(0)

while(counter < 20):

ret, frame = video.read()

if ret == True:

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

faces = face\_cas.detectMultiScale(gray, 1.3,5)

for (x, y, w, h) in faces:

face\_component = gray[y:y+h, x:x+w]

fc = cv2.resize(face\_component, (48, 48))

inp = np.reshape(fc,(1,48,48,1)).astype(np.float32)

inp = inp/255.

prediction = exp\_model.predict\_proba(inp)

expression = facial\_expression[np.argmax(prediction)]

cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2)

if expression == 'Neutral' or expression == 'Happy':

confident = confident + 1

else:

confuse = confuse + 1

counter = counter + 1

print(counter)

cv2.putText(frame, "Confident Count : "+str(confident), (30, 40), font\_cv, 1, (0, 255, 0), 2)

cv2.putText(frame, "Confuse Count : "+str(confuse), (30, 120), font\_cv, 1, (0, 255, 0), 2)

cv2.imshow("image", frame)

if cv2.waitKey(250) & 0xFF == ord('q'):

break

else:

break

video.release()

cv2.destroyAllWindows()

if confident > 0:

confident = confident / 20.0

if confuse > 0:

confuse = confuse / 20.0

text.insert(END,"Your Visual Interview Confidence% : "+str(confident)+"\n")

text.insert(END,"Your Visual Interview Confusion% : "+str(confuse)+"\n")

text.update\_idletasks()

def essayAssessment():

global vectorizer, normalize, xgb

essay = text.get(1.0, "end-1c")

print(essay)

state = essay.strip().lower()

state = cleanPost(state)

temp = []

temp.append(state)

temp = vectorizer.transform(temp).toarray()

temp = normalize.transform(temp)

predict = xgb.predict(temp)

predict = predict[0]

messagebox.showinfo("Your Essay Prediction Score : "+str(predict), "Your Essay Prediction Score : "+str(predict))

def extract\_feature(file\_name, mfcc, chroma, mel):

with soundfile.SoundFile(file\_name) as sound\_file:

X = sound\_file.read(dtype="float32")

sample\_rate=sound\_file.samplerate

if chroma:

stft=np.abs(librosa.stft(X))

result=np.array([])

if mfcc:

mfccs=np.mean(librosa.feature.mfcc(y=X, sr=sample\_rate, n\_mfcc=40).T, axis=0)

result=np.hstack((result, mfccs))

if chroma:

chroma=np.mean(librosa.feature.chroma\_stft(S=stft, sr=sample\_rate).T,axis=0)

result=np.hstack((result, chroma))

if mel:

mel=np.mean(librosa.feature.melspectrogram(X, sr=sample\_rate).T,axis=0)

result=np.hstack((result, mel))

sound\_file.close()

return result

def spokenAssessment():

global speech\_classifier

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

fname = os.path.basename(filename)

test = []

mfcc = extract\_feature(filename, mfcc=True, chroma=True, mel=True)

test.append(mfcc)

test = np.asarray(test)

test = test.astype('float32')

test = test/255

test = test.reshape((test.shape[0],test.shape[1],1,1))

predict = speech\_classifier.predict(test)

predict = np.argmax(predict)

predict = speech\_emotion[predict-1]

if predict == 'neutral' or predict == 'calm' or predict == 'happy':

messagebox.showinfo("Your Speaking Verbal Audio Predicted as : Confident","Your Speaking Verbal Audio Predicted as : Confident")

else:

messagebox.showinfo("Your Speaking Verbal Audio Predicted as : Confuse","Your Speaking Verbal Audio Predicted as : Confuse")

font = ('times', 14, 'bold')

title = Label(main, text='Active Online Learning for Social media Analysis to Support Crisis Management')

title.config(bg='DarkGoldenrod1', fg='black')

title.config(font=font)

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

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

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

loadButton = Button(main, text="Generate & Load Assessment Model", command=loadModels)

loadButton.place(x=50,y=250)

loadButton.config(font=font1)

videoButton = Button(main, text="Visual Interview Assessment", command=visualAssessment)

videoButton.place(x=400,y=250)

videoButton.config(font=font1)

spokenButton = Button(main, text="Spoken Interview Assessment", command=spokenAssessment)

spokenButton.place(x=690,y=250)

spokenButton.config(font=font1)

essayButton = Button(main, text="Written & Short Essay Assessment", command=essayAssessment)

essayButton.place(x=990,y=250)

essayButton.config(font=font1)

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

text=Text(main,height=13,width=150)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=10,y=400)

text.config(font=font1)

main.config(bg='LightSteelBlue1')

main.mainloop()