APPENDIX

import nltk

from n1tk.stem import WordNetLemmatizer # from nltl‹.corpus import stopwords

from numpy.lib.twodim base import triu indices from

lemmatizer = WordNetLemmatizer() import json

import pickle

impor‘t numpy as np

from keras.models import Sequential

from keras.layers import Dense, Activation, Dropout, LSTM from keras.optimizers import SGD

import random ’

nltk .down1oad( ’ punkt ' ) nltk .download( ’ wordnet ‘ )

# nick.download('stopwords’)

words=[] classes = [] documents = [ J

ignore words - [ ’ ? ’ ' ! ‘ j

data file = open(’futbot.json', encoding=’utf-8’).read()

intents - json.loads(data\_file)

print( 'Intent file read success-Fully ' }

# setting stop words

# stop words = set(stopwords.words(’english’))

for intent in intents['intents']:

-for pattern in intent[ ’ patterns ' ):

*4tol‹enize* each word

w = nltk.word\_tokenize(pattern)

words . extend(w)

tfadd document s in the corpus documents . append( (w, intent[ ' tag’ ]))

# add’to our classes list

if intent['tag'] not in classes:

classes .append(intent [ ' tag‘ ])

# # lemmaztize and lower each word and remove duplicates

words - [lemmatizer.lemmatize(w.lower()) for w in words if w not in

ignore words J

word s = sorted(list(set(words ) ) ) pnknt ( ’ word sorted .. . ' )

p mint ( ' cla ss es are: ‘ , cla sses )

# sort classes

print('classes length: ', len(classes))

classes — sorted(set(c 1as *ses)* ) print( 'class sorted' )

1 doc uments = corb Anat bon bet een paIt ein s and Enlents

print (1en(documents) , ”document s" ) # clas ses = ! ntents

print (len(cla sses) , "classes", cla sses )

# words = all words, vocabulary

print (ten(wo rds ) , “unique lemmaI i zed bonds ", wo *rds )*

pickle.dump(words,open('texts.pkl','wb')) print('texts.pkl dusped successfully') pickle.dump(classes,open('labels.pkl', 'wb')) print('lebel.pkl dumped successfully')

# create our I r ai ni ng data

training - I

ñ cneate an eripty amay ‹on oun output

output empty = [9] \* ten ( eta s ses )

# training set, bag of words fo’ ee:' sertence for doc in documents:

# initialize our bag of worus

bag — [ ]

# list of tokenized words for thr pattern pattern words = doc[0]

# lemmatize each word - create base word, in attempt to represent related wcrds

pattern words = [lemmatizer.lemmatize(word.lower()) for word in

pattern wood s ]

# create our bag of words array 'Cth 1, if word latch fnund in current pattern

-on w in wond s :

bag. append(1) if w in patterri «ords else bag . append(0)

.. output is a ’ 0 ' for each ad end ’ I ’ Tor‘ *cu : ent* tag (for each pattern)

output row = list(output empty) output row[classes.index(doc[1])] = 1

training.append([bag, output row])

# shuffle our features and turn into np.array

random.shuff1e(training) training = np.array(training)

# create train and test lists. X - patterns, Y - intents . train x = list(training[:,B])

trains = list(training[:,l])

print("Training data created”)

# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output layer contains number of neurons

# equal to number of intents to predict output intent with sof5max model = Sequential()

model.add(Dense(128, input shape=(1en(train\_x[0J),), activation='relu')) model.add(Dropout(0.9))

model.add(Dense(64, activation=’relu'))

# model.add(Dense(64, activation-'relu')) model.add(Deuse(len(train [0]), activation=’softmax'))

# Compile model. Stochastic gradient descent with Nesterov accelerated gradient gives good results for this model

sgd - SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True) model.compile(loss='categorical crossentropy’, optimizer='adam', metrics-['accuracy'])

# sgd

#fitting and saving the model

hist = model.fit(np.array(train\_x), np.array(trains), epochs=200, batch size=S, verbose=l)

model.save(’model.h5', hist)

# # Define the LSTN model architecture # seq\_length len(train x[0])

# model = Sequential()

*#* model. add(LSTH(un1ts=64, 1nput\_shape= (seq length, 1),

return sequences=True))

# model.add(Dropout(0.2)) # model.add(LSTN(units=64)) # model.add(Dropout(0.2))

# model.add(Dense(units-seq length, activation=’softmax'))

# model.compile(optimizer=’adam', loss='categorical crossentropy', metrics-[’accuracy’j)

# # Train the model

# hist = model.fit(np.array(train\_x), np.array(trains), epochs=200,

batch size=5, verbose=1)

# model . save( ’ mode1. h5 ' , hist)

print("model created”)

from flask import Flask, render\_template, request, Response import random

import json

from keras.models import load model import numpy as np

import pickle

from nltk.stem import WordNetLemmatizer import nltk

from nltk.corpus import stopwords

import speech recognition as sr

import pytt*s x 3*

*mom* txt2speech impont speak\_te xt, mdc

nltk.download('punkt') nltk.download('wordnet') nltk.download('Stopwords')

nltk.download('popular') lemmatizer = WordNetLemmatizer()

model = load model('model.h5')

intents - json.loads(open('futbot.json', encoding='utf-8').read()) words - pickle.load(open('texts.pkl', 'rb'))

classes - pickle.load(open('labels.pkl', ’rb'))

# setting stop words

stop words - set(stopwords.words('english’))

def clean up\_sentence(sentence):

# tokenize the pattern - split words into array

sentence woi°ds = nltk .word toI‹enize(sentence)

# stem each word - create short form for word

sentence words = [lemmatizer.lemmatize(word.lower()) for word in sentence words]

# words - [lemmatizer.lemmatize(w.lower()) for w in words if iv not in stop words]if word not in s:op wor’ds

return sentence words

t return bag of words array: 0 or 1 for each worJ in the bag that exists in the sentence

def bow(sentence, words, show details=True): # tokenize the pattern

sentence words - clean up\_sentence(sentence)

# bag *o(* words *-* matrix of N words, vocabula ny matrix bag — [0] \*ten(words)

Ton s in sentence word s :

for i, w in enumerate(words):

if w == s:

i assign 1 if current word is in the vocabulary position bag[i] = 1

if show\_details:

print("found in bag: ,s” % w) return(np.array(bag))

def predict class(sentence, model):

# filter out predictions below a threshold

p = bow(sentence, words, show details=False) res = model.predict(np.array([pJ))[0]

ERROR THRESHOLD = 0.25

results = [[i, r) for i, r in enumerate(res) if r > ERROR THRESHOLD] f sort by strength of prodability

results.sort(key=lambda x: x[1], reverse=True) return list = []

for r in results:

return list.append(("intent”: classes[r[O]], ”probability”:

return return list

def getResponse(ints, intents json): if (len(ints) ‹ 1):

return "Can you please rephrase your question?, I'm having trouble understanding"

tag = ints[dJ['intent'J

list of intents = intents json['intents'] for i in list of intents:

if(i['tag'] == tag):

result - random.choice(i 'responses'j) break

return result

def chatbot\_response(msg):

ints - predict class(msg, model) res = get Response ( ints, intent s) print(res)

netopn re s

app = F1ask ( named)

app . static folder = ' st ati c '

(§app . poute( ”/ " )

def home():

return render template(”index.html”)

wlcm\_txt =“Hey dear welcome, I am your virtual assistant what can i do for you? Note,You can either speek or write your querry! and also, You must confirm what ever solution is given to you Gs i am only a bot.”

speak text(wlcm txt)

@app.route(”/get")

def get bot response():

us erText = request . a igs . get ( ‘ rnsg ’ ) response - chatbot response( us enText) s peak text(response)

# retunn Response ( j son. dumps(( ’ res ’ : mes pon z<e}) ,

mïi›ietype= ’ appU caII on/j son ' ) ; return *response*

@app.route(”/bot/api/v1/prompt”, methods=['kOST']) def get api response():

userText = request.grgs.get('msg')

response = chatòot res ponse( userText) ;

peturn Response(j son. dumps (( ' les ' : res pons e)) ,

miretype= ' appli cation/j son ' ) ;

if ndme == ” main ": app. run(debug-True)

irrport speech recognit ion as sn iupor‘t pyttsx3

# Initialize the recognizer r = sr.Recognizer()

S Function to convert ?ext to speech

deb speal‹ Text (c onoand) :

# Initialize the engine

engine = pyttsx3.init()

”””VOICE”””

voices = engine . getProperty( ' voices ’ ) #¢etting details ot current

voice

#engine.setProperty('voice', voices 0].id) #changing index, changes voices. o for male

engine.setPropenty('voice', voices[1].id) #changing index, changes voices. l for female

eng:ine . say(command)

engine . nunAndWait ( )

# Loop infinitely for user to speak def mic(duration=S.2):

while True:

# use the sicrophone as source for input.

with sr .Micnophone( ) as source :

4 wait hon a sewer d ?o let I he ncc ognizen # cdj ust the en\*rqh th ne shoed ba sed on

# the surrounding noise levcl

r.adjust for ambient noise(source, duration)

# listens for tre user's ingut audio = r.listen(source)

# Using google -o recognize eudio MyText = n. recogn ze google (audio ) MyText = MyText . fo\ver ( )

print("Did you say : ” , MyT ext) speak text(MyText )

except sr .Request Enror as e:

potnt( "Could not new ue st r esuIt s; (9}" .I-oreat(e) )

except sn.UnknounVa1ueEnror:

print("unknown error occurred“)

if enamel == ” main ”: # Start speaking

speak text(”Hello! my name is Jenifa... Go ahead and say your querry or

type it. You t‹now what? just do as you wish”)

# Start listening