Chatbot Assignment- 4

Dated – 19/6/2024

**New.py : loads the intents in the json file and, builds and tokenizes them to fit into the pre-loaded AI model.**

import random

import json

import pickle

import numpy as np

import tensorflow as tf

import nltk

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

intents = json.loads(open('intents.json').read())

words = []

classes = []

documents = []

ignoreLetters = ['?', '!', '.', ',']

for intent in intents['intents']:

for pattern in intent['patterns']:

wordList = nltk.word\_tokenize(pattern)

words.extend(wordList)

documents.append((wordList, intent['tag']))

if intent['tag'] not in classes:

classes.append(intent['tag'])

words = [lemmatizer.lemmatize(word) for word in words if word not in ignoreLetters]

words = sorted(set(words))

classes = sorted(set(classes))

pickle.dump(words, open('words.pkl', 'wb'))

pickle.dump(classes, open('classes.pkl', 'wb'))

training = []

outputEmpty = [0] \* len(classes)

for document in documents:

bag = []

wordPatterns = document[0]

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

for word in words:

bag.append(1) if word in wordPatterns else bag.append(0)

outputRow = list(outputEmpty)

outputRow[classes.index(document[1])] = 1

training.append(bag + outputRow)

random.shuffle(training)

training = np.array(training)

trainX = training[:, :len(words)]

trainY = training[:, len(words):]

model = tf.keras.Sequential()

model.add(tf.keras.layers.Dense(128, input\_shape=(len(trainX[0]),), activation = 'relu'))

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(64, activation = 'relu'))

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(len(trainY[0]), activation='softmax'))

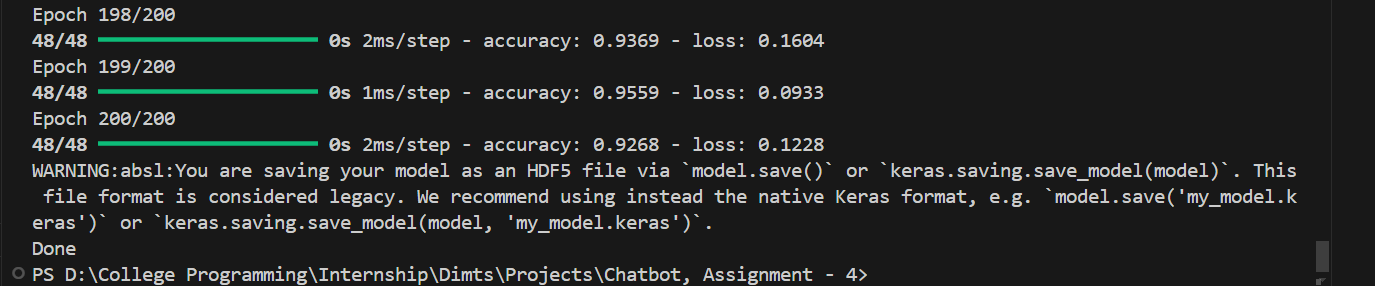
sgd = tf.keras.optimizers.SGD(learning\_rate=0.01, momentum=0.9, nesterov=True)

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

hist = model.fit(np.array(trainX), np.array(trainY), epochs=200, batch\_size=5, verbose=1)

model.save('chatbot\_model.h5', hist)

print('Done')



**Chatbot.py : Acts as a discriminative ai model which converses with the user to answer their queries using related intents stored in Json file and their responses.**

import random

import json

import pickle

import numpy as np

import nltk

from nltk.stem import WordNetLemmatizer

from keras.models import load\_model

lemmatizer = WordNetLemmatizer()

intents = json.loads(open('intents.json').read())

words = pickle.load(open('words.pkl', 'rb'))

classes = pickle.load(open('classes.pkl', 'rb'))

model = load\_model('chatbot\_model.h5')

def clean\_up\_sentence(sentence):

sentence\_words = nltk.word\_tokenize(sentence)

sentence\_words = [lemmatizer.lemmatize(word) for word in sentence\_words]

return sentence\_words

def bag\_of\_words (sentence):

sentence\_words = clean\_up\_sentence(sentence)

bag = [0] \* len(words)

for w in sentence\_words:

for i, word in enumerate(words):

if word == w:

bag[i] = 1

return np.array(bag)

def predict\_class (sentence):

bow = bag\_of\_words (sentence)

res = model.predict(np.array([bow]))[0]

ERROR\_THRESHOLD = 0.25

results = [[i, r] for i, r in enumerate(res) if r > ERROR\_THRESHOLD]

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

return\_list = []

for r in results:

return\_list.append({'intent': classes [r[0]], 'probability': str(r[1])})

return return\_list

def get\_response(intents\_list, intents\_json):

tag = intents\_list[0]['intent']

list\_of\_intents = intents\_json['intents']

for i in list\_of\_intents:

if i['tag'] == tag:

result = random.choice (i['responses'])

break

return result

print("GO! Bot is running!")

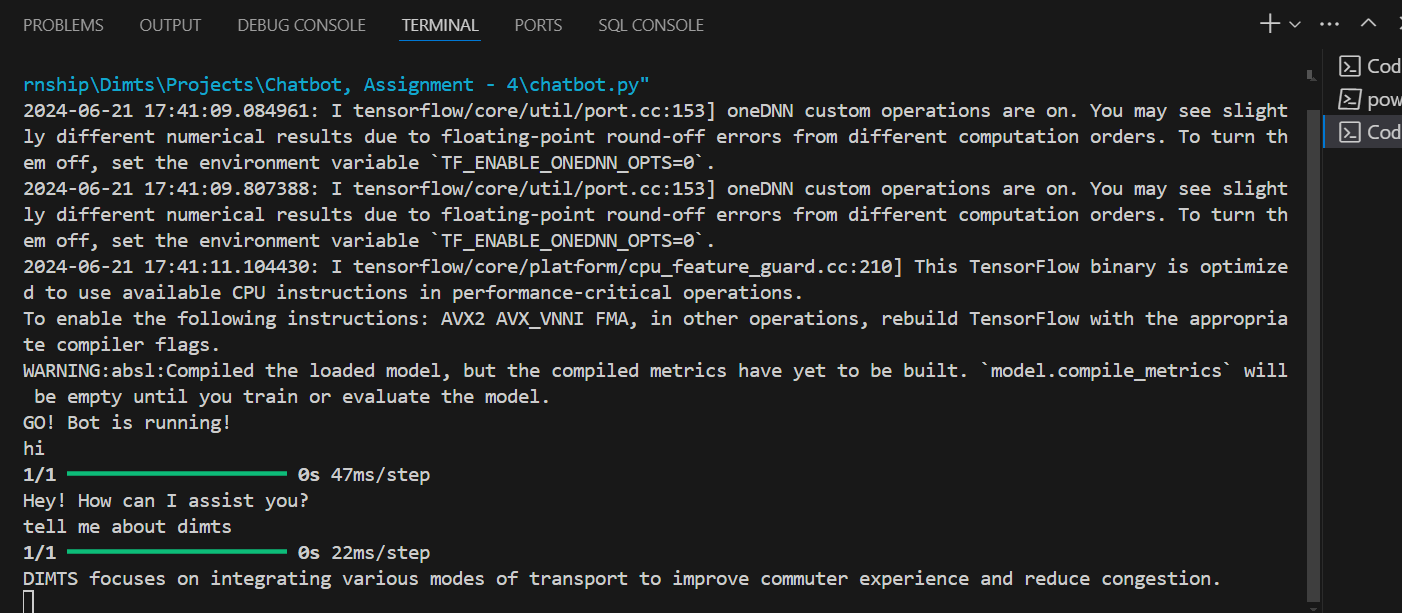
while True:

message = input("")

ints = predict\_class (message)

res = get\_response (ints, intents)

print (res)



**parse.py : Used to analyze a pdf file and create intents from that data then store those intents in a csv file.**

import fitz as PyMuPDF

import json

import re

import nltk

from nltk.tokenize import sent\_tokenize

from nltk.stem import WordNetLemmatizer

# Initialize WordNet lemmatizer

lemmatizer = WordNetLemmatizer()

# Function to clean and lemmatize text

def clean\_up\_text(text):

text = re.sub(r'\s+', ' ', text) # Remove extra whitespace

tokens = nltk.word\_tokenize(text.lower()) # Tokenize and lowercase

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

return ' '.join(tokens)

# Function to check if an intent already exists

def intent\_exists(intents, tag, patterns):

for intent in intents:

if intent['tag'] == tag and any(pattern in intent['patterns'] for pattern in patterns):

return True

return False

# Function to parse PDF and extract intents from meaningful sections

def parse\_pdf\_and\_extract\_intents(pdf\_file, intents\_json\_file):

try:

with open(intents\_json\_file, 'r') as file:

intents = json.load(file) # Load existing intents

except FileNotFoundError:

intents = [] # Initialize as empty list if file doesn't exist

except json.JSONDecodeError:

intents = [] # Initialize as empty list if file is empty or corrupted

# Initialize empty lists for new intents

new\_intents = []

# Parse PDF

pdf\_document = PyMuPDF.open(pdf\_file)

# Iterate through pages

for page\_num in range(len(pdf\_document)):

page = pdf\_document.load\_page(page\_num)

blocks = page.get\_text("dict")['blocks']

for block in blocks:

if block['type'] == 0: # Text block

lines = block['lines']

for line in lines:

if line['spans']:

text = line['spans'][0]['text'].strip()

if len(text) > 20: # Example condition for meaningful content

# Clean and tokenize line

cleaned\_line = clean\_up\_text(text)

# Create intent

tag = f'heading\_{page\_num}\_{lines.index(line)}' # Unique tag for each heading

patterns = [text]

responses = ["Example response for " + text[:50]] # Example response based on content

context = ["Page " + str(page\_num + 1)] # Optional context information

# Check if intent already exists

if not intent\_exists(intents, tag, patterns):

new\_intents.append({"tag": tag, "patterns": patterns, "responses": responses, "context": context})

# Add new intents to existing intents

intents.extend(new\_intents)

# Save to intents2.json

with open(intents\_json\_file, 'w') as file:

json.dump(intents, file, indent=4)

print(f"Extracted {len(new\_intents)} new intents. Saved to intents2.json.")

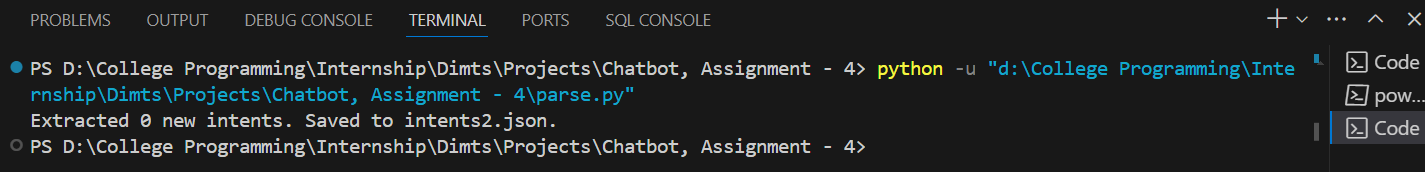
# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

pdf\_file = 'path.pdf' # Replace with your PDF file path

intents\_json\_file = 'intents2.json' # Existing intents file

parse\_pdf\_and\_extract\_intents(pdf\_file, intents\_json\_file)



**intents.json: Acts as a backend for storing intents used to create chatbot responses for the users' queries.**

