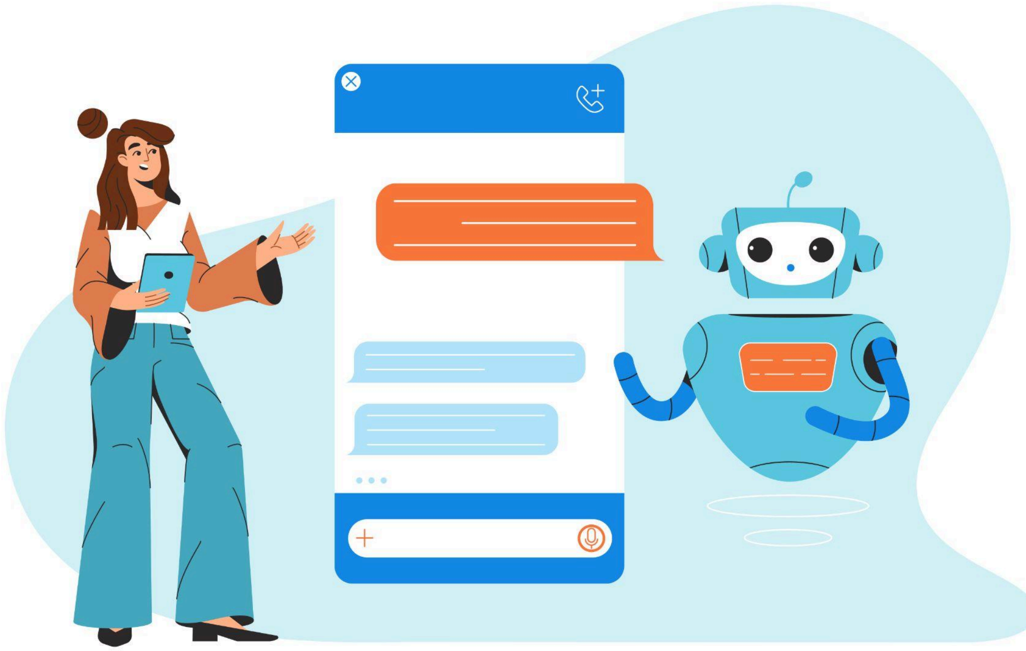

CHATBOT FOR MENTAL HEALTH

27.07.2024

TEAM MEMBERS

PRAGADESHWARAN S

Dataset Collection JSON

RASHEEN FAHMI M

Voice-Voice integration and Dataset

ITHIKASH R

CNN model development

ARUN KUMAR S

Documentation

PROBLEM STATEMENT

Development of a Chatbot for Providing Remedies for Stress, Depression, and General Health Conditions

Dataset and Source (don t change this)

1. **Dataset Overview:** The dataset is a JSON file that contains a collection of intents. Each intent includes various components essential for training the chatbot to understand and respond to user queries effectively.
2. **Source:** We compiled and managed multiple JSON and CSV files to create this final dataset. This dataset can be further refined and expanded to enhance its accuracy and effectiveness in chatbot training.

DATASET PREPARATION

Modules Used:

We import the necessary packages for our chatbot and initialize the variables we will use in our Python project. The data file is in JSON format so we used the json package to parse the JSON file into Python.

```
import nltk

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

import json

import pickle
```

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import SGD
import random
from IPython.display import HTML, Audio, display, Javascript
from google.colab.output import eval_js
from base64 import b64decode
from pydub import AudioSegment
import io
import time
import speech_recognition as sr
from gtts import gTTS
```

LOADING DATASET:

```
words=[]
classes = []
documents = []
ignore_words = ['?', '!']
data_file = open("/content/deda.json").read() # read json file
intents = json.loads(data_file) # load json file
```

When working with text data, we need to perform various preprocessing on the data before we make a machine learning or a deep learning model. Based on the requirements we need to apply various operations to preprocess the data.

- Tokenizing is the most basic and first thing you can do on text data.
- Tokenizing is the process of breaking the whole text into small parts like words.
- Here we iterate through the patterns and tokenize the sentence using nltk.word_tokenize() function and append each word in the words list. We also create a list of classes for our tags.

Tokenization:

```
for intent in intents['intents']:
    for pattern in intent['patterns']:
        #tokenize each word
        w = nltk.word_tokenize(pattern)
        words.extend(w)# add each elements into list
        #combination between patterns and intents
        documents.append((w, intent['tag']))#add single element into end of list
        # add to tag in our classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
```

Lemmatization:

Now we will lemmatize each word and remove duplicate words from the list.

- Lemmatizing is the process of converting a word into its lemma form and then creating a pickle file to store the Python objects which we will use while predicting.

```
# lemmatize, lower each word and remove duplicates

words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]

words = sorted(list(set(words)))

# sort classes

classes = sorted(list(set(classes)))

# documents = combination between patterns and intents

print (len(documents), "documents\n", documents, "\n")

# classes = intents[tag]

print (len(classes), "classes\n", classes, "\n")

# words = all words, vocabulary

print (len(words), "unique lemmatized words\n", words, "\n")

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

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

Training the Model:

Now, we will create the training data in which we will provide the input and the output.

- Our input will be the pattern and output will be the class our input pattern belongs to. But the computer doesn't understand text so we will convert text into numbers

```
import numpy as np

import random

# Create empty lists for bags and output_rows

bags = []

output_rows = []


# Iterate through documents to create bags and output_rows

for doc in documents:

    bag = []

    pattern_words = doc[0]

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


    # Create a bag for the current document

    for w in words:

        bag.append(1) if w in pattern_words else bag.append(0)
```

```
    # Create the output row for the current document
    output_row = [0] * len(classes)
    output_row[classes.index(doc[1])] = 1

    # Append the bag and output_row to their respective lists
    bags.append(bag)
    output_rows.append(output_row)
# Convert bags and output_rows to numpy arrays
bags = np.array(bags)
output_rows = np.array(output_rows)
# Combine bags and output_rows into a list of tuples
training = list(zip(bags, output_rows))
# Shuffle training
random.shuffle(training)
# Convert bags and output_rows back to lists for easier indexing
bags, output_rows = zip(*training)
# Convert bags and output_rows to numpy arrays
bags = np.array(bags)
output_rows = np.array(output_rows)
# Split into train and test sets
train_x = bags
train_y = output_rows
```

```
# Use the same data for the test for now (you might want to split differently)

test_x = train_x

test_y = train_y

print("Training data created")


from tensorflow.python.framework import ops

ops.reset_default_graph()
```

MODEL ARCHITECTURE

Building the CNN model: -

We have our training data ready, now we will build a deep neural network that has 3 layers. We use the Keras sequential API for this. After training the model for 200 epochs, we achieved 100% accuracy on our model. Let us save the model as 'chatbot_model.h5'.

```
# Create the model

model = Sequential()

model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))

model.add(Dropout(0.5))

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

model.add(Dropout(0.5))

model.add(Dense(len(train_y[0]), activation='softmax'))


# Compile the model using Adam optimizer without weight_decay

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Train the Model:

```
# Train the model
hist = model.fit(np.array(train_x), np.array(train_y), epochs=75, batch_size=5, verbose=1)
model.save('medical_chatbot_model.h5')
from keras.models import load_model
model1 = load_model('medical_chatbot_model.h5')
from sklearn.model_selection import train_test_split
train_x, val_x, train_y, val_y = train_test_split(train_x, train_y, test_size=0.1, random_state=42)
# Evaluate the model on the test set
loss, accuracy = model1.evaluate(np.array(test_x), np.array(test_y), verbose=1)
print("Test loss:", loss)
print("Test accuracy:", accuracy)
```

10/10 [==============================] - 0s 2ms/step - loss: 0.0643 - accuracy: 0.9830

Test loss: 0.06432799994945526

Test accuracy: 0.982993185520172

FOR PREDICTING RESPONSE:

We load intents from a JSON file and vocabulary and class labels from pickled files and we do Text Preprocessing which includes functions to clean and preprocess input sentences, such as tokenization and lemmatization.

```
intents = json.loads(open(r"/content/deda.json").read())
words = pickle.load(open('words.pkl','rb'))
```

```
classes = pickle.load(open('classes.pkl','rb'))
def clean_up_sentence(sentence):
# tokenize the pattern - split words into array
  sentence_words = nltk.word_tokenize(sentence)
#print(sentence_words)
# stem each word - create short form for word
  sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words]
#print(sentence_words)
  return sentence_words
def bow(sentence, words, show_details=True):
# tokenize the pattern
  sentence_words = clean_up_sentence(sentence)
  #print(sentence_words)
  # bag of words - matrix of N words, vocabulary matrix
```

The code converts input sentences into a bag-of-words representation to match the words with the predefined vocabulary.

```
  bag = [0]*len(words)
  #print(bag)
  for s in sentence_words:
      for i,w in enumerate(words):
          if w == s:
              # assign 1 if current word is in the vocabulary position
              bag[i] = 1
              if show_details:
```

```
            print ("found in bag: %s" % w)
          #print ("found in bag: %s" % w)
  #print(bag)
  return(np.array(bag))
```

It predicts the class of the input sentence using a pre-trained machine learning model and filters predictions based on a defined threshold.

```
def predict_class(sentence, model):
  # filter out predictions below a threshold
  p = bow(sentence, words,show_details=False)
  #print(p)
  res = model.predict(np.array([p]))[0]
  #print(res)
  ERROR_THRESHOLD = 0.25

  results = [[i,r] for i,r in enumerate(res) if r>ERROR_THRESHOLD]
  #print(results)
  # sort by strength of probability

  results.sort(key=lambda x: x[1], reverse=True)
  #print(results)
  return_list = []
  for r in results:
    return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
  return return_list
```

```
 #print(return_list)
```

Based on the predicted class, it selects an appropriate response from a set of predefined responses in the intents JSON file.

```
def getResponse(ints, intents_json):
 tag = ints[0]['intent']
 #print(tag)
 list_of_intents = intents_json['intents']
 #print(list_of_intents)
 for i in list_of_intents:
     if(i['tag']== tag):
         result = random.choice(i['responses'])
         break
 return result
```

The main function integrates these steps to take user input, predict the intent, and generate a corresponding response, enabling the chatbot to interact with users effectively.

```
def chatbot_response(text):
 ints = predict_class(text, model)
 #print(ints)
 res = getResponse(ints, intents)
 #print(res)
 return res
```

PREDICTING RESULTS:

Text-text:

- The provided code sets up an interactive chatbot session and terminate when we give quit , exit, bye as input and if error throws asking rephrase the question

```
start = True
while start:
 query = input('Enter Message:')
 if query in ['quit','exit','bye']:
     start = False
     continue
 try:
     res = chatbot_response(query)
     print(res)
 except:
     print('You may need to rephrase your question.')
```

```
Enter Message:hi
Code cell output actions =============] - 0s 268ms/step
Hello! How can I assist you with Medical Assistance today?
Enter Message:im feeling low
1/1 [==============================] - 0s 20ms/step
Mood swings and irritability can be symptoms of bipolar disorder. Treatments often involve mood stabilizers and psychotherapy.
Enter Message:body is hot
1/1 [==============================] - 0s 20ms/step
Stay hydrated and rest. Over-the-counter medications like acetaminophen or ibuprofen can help reduce fever. If the fever is very high or lasts more than a couple of
Enter Message:im having coughj
1/1 [==============================] - 0s 22ms/step
Hello! How can I assist you with Medical Assistance today?
Enter Message:im having cough
1/1 [==============================] - 0s 21ms/step
Persistent coughing and shortness of breath, especially at night, could indicate Chronic Obstructive Pulmonary Disease (COPD). Inhaled bronchodilators and quitting s
```

Voice Recognition:

- It generates a response and converts the response to speech using the Google Text-to-Speech (gTTS) library. ends if a termination keyword is entered, and plays the spoken response

```
from gtts import gTTS
from IPython.display import Audio, display

for _ in range(5):
    query = input('Enter Message: ')
    print("f")
    if query.lower() in ['quit', 'exit', 'bye']:
        start = False
        continue
    res = chatbot_response(query)
    print(res)

    # Convert the response to speech
    tts = gTTS(text=res, lang='en')
    tts.save("response.mp3")
    while True:
      # Play the response without autoplay
      audio = Audio("response.mp3")
```

display(audio)

break

```
Enter Message:hi
1/1 [==============================] - 0s 44ms/step
Hello! How can I assist you with Medical Assistance today?
0:04 / 0:04
Enter Message:im feeling low
1/1 [==============================] - 0s 24ms/step
Mood swings and irritability can be symptoms of bipolar disorder. Treatments often involve mood stabilizers and psychotherapy.
0:02 / 0:09
```

Voice-Voice:

- This section captures audio from the user using a web interface, processes it into a suitable format, and converts the speech to text using Google's Speech Recognition API
- It handles the processing of the transcribed text input, generates a response using a chatbot model, and converts the chatbot's response back to speech.

```
from IPython.display import HTML, Audio, display, Javascript

from google.colab.output import eval_js

from base64 import b64decode

from pydub import AudioSegment

import io

import time

import speech_recognition as sr

from gtts import gTTS
```

```
def get_audio():
    display(HTML(AUDIO_HTML))
    data = eval_js("data")
    if not data:
        return None
    binary = b64decode(data.split(',')[1])
    # Convert audio data to pydub AudioSegment
    audio_segment = AudioSegment.from_file(io.BytesIO(binary), format="webm")
    # Convert to mono channel
    audio_segment = audio_segment.set_channels(1)
    # Convert to 16-bit PCM format
    audio_segment = audio_segment.set_sample_width(2)
    # Convert to 16 kHz sample rate
    audio_segment = audio_segment.set_frame_rate(16000)
    return audio_segment


def speech_to_text(audio_segment):
    recognizer = sr.Recognizer()
    audio_data = io.BytesIO()
    audio_segment.export(audio_data, format="wav")
    audio_data.seek(0)
```

```
    with sr.AudioFile(audio_data) as source:
        audio = recognizer.record(source)
    try:
        text = recognizer.recognize_google(audio)
        return text
    except sr.UnknownValueError:
        print("Could not understand audio.")
        return None
    except sr.RequestError as e:
        print(f"Error connecting to Google API: {e}")
        return None

def chatbot_response(text):
  ints = predict_class(text, model)
  #print(ints)
  res = getResponse(ints, intents)
  #print(res)
  return res
start = True
while start:
    audio_segment = get_audio()
    if audio_segment:
```

```
    try:
        audio_text = speech_to_text(audio_segment)
        if audio_text:
            print(f"You said: {audio_text}")
            if audio_text.lower() in ["bye", "exit", "quit"]:
                print("Conversation ended.")
                break
            res = chatbot_response(audio_text)
            print(res)
            # Convert the response to speech
            tts = gTTS(text=res, lang='en')
            tts.save("response.mp3")
            # Play the response
            display(Audio(filename="response.mp3", autoplay=True))
    except Exception as e:
        print(f'Error: {e}')
# Add a delay after processing to prevent continuous recording
time.sleep(5)  # You can adjust the delay as needed
```

```
You said: hello
1/1 [==============================] - 0s 36ms/step
Hello! How can I assist you with Medical Assistance today?
0:01 / 0:04
You said: I am having shortness of breath during physical activities
1/1 [==============================] - 0s 20ms/step
Asthma is managed with inhaled corticosteroids and avoiding known allergens. Using a short-acting beta-agonist can
0:04 / 0:11
```

FUTURE WORKS:

1. Multi-Language Support:

- **Language Detection:** Automatically detect the user's language and switch to the appropriate one for a seamless user experience.
- **Localization:** Customize responses to align with cultural nuances and local health practices.
- **Translation Services:** Utilize advanced translation services to ensure accurate and contextually appropriate translations.

2.Emotion Detection:

- **Sentiment Analysis:** Implement sentiment analysis to gauge the user's emotional tone and adjust responses accordingly.
- **Facial Recognition:** Integrate with facial recognition technologies to detect emotional cues from user expressions.
- **Voice Tone Analysis:** Analyze the tone and pitch of the user's voice to detect stress or anxiety levels.

3.Mental Health Resources:

- **Resource Database:** Develop a comprehensive database of mental health resources, including articles, videos, and professional contacts.

- **Resource Matching:** Match users with relevant resources based on their specific issues and needs.

4.Crisis Management:

- **Crisis Detection:** Implement algorithms to detect signs of crisis or severe distress in user inputs.
- **Emergency Contacts:** Provide immediate access to emergency contacts and hotlines during critical moments.
- **Professional Referral:** Automatically refer users to mental health professionals if a crisis is detected.