Trabajo de Fin de Grado

UNIVERSIDAD DE GRANADA

Desarrollo de un Chatbot para Servicios Sociales Dirigido a Personas en Situación de Dependencia

Sprint 2

Claudia Salado Méndez

1. Introducción

En este sprint el objetivo era tener conectado el chatbot al localhost y tener una página web donde comunicarse con él de forma visual, además de entrenarle para que conteste de una determinada forma a unas preguntas concretas.

2. Implementación

- **Primer Objetivo**

El primer objetivo es conectar el bot con el localhost y poder conversar con él desde allí. Para ello hay que instalar un framework web llamado Flask que es el que se utiliza en python dentro del environment.

```
pip install flask
```

En el archivo python del chatbot hay que importarlo para poder usarlo, además de hacerle más cambios para que detecte el html.

```
from flask import Flask, render_template, request
```

Se crea el archivo html, con un contenido básico para el funcionamiento. Ahora ejecutando el programa se debería poder hablar con el chatbot en el localhost.

```
python app.py
```

Esto da un problema al ir al localhost dice que no hay conexión,este mensaje indica que el servidor no está respondiendo al puerto solicitado y como se puede observar al ejecutar el programa no aparece ningún mensaje en la terminal de que flash se esté ejecutando, este error es debido a que el código de python le falta algo, hay que realizar un par de cambios en el código, incluirle dos decoradores.

- @app.route('/') para vincular la función home() a la ruta raíz del servidor.
- @app.route('/chat', methods=['POST']) para vincular la función chat() a la ruta '/chat' y limitarla a los métodos POST.

Asi se soluciono este problema pero ahora aparece otro que hace que no se pueda escribir al bot, que nos indica que no encuentra el archivo html, para ello se tiene que crear una carpeta para meter dentro de ella el html quedando la estructura del proyecto de esta forma:

```
- chatbot.py
- templates
  - index.html
```

Además hay que cambiar la función home del archivo python:

```
@app.route('/')
def home():
    return render_template('templates/index.html').
```

Gracias a esto ya se le puede escribir una pregunta al bot pero al pulsar en enviar no envia la pregunta, este problema se sabe que al ser al pulsar el botón tiene que ver con el JavaScript que hay en el html, por eso hay que modificar la parte del script un poco para que funcione. Tras realizar un par de cambios ya funcionaba correctamente.

Ya se ha cumplido el primer objetivo pero el localhost era muy básico y para nada visual por eso se propuso como siguiente objetivo crear un css a parte donde especificar el estilo del html un poco para que fuese más claro y así ya tener un pequeño ejemplo de funcionamiento simple que durante los próximos sprints se irá haciendo más complejo.

Así se crea el archivo styles.css que servirá para estilizar la interfaz de usuario del chatbot. Para que el html lo reconozca y lo use hay que añadirle la siguiente línea de código en el head.

```
<link rel="stylesheet" type="text/css" href="styles.css">
```

Aun así esto dio problemas, no reconoció el archivo css, dando el error 404 indicando que el archivo css no se encuentra en el servidor, como flask es el servidor utilizado hay que agregar la siguiente línea de código al archivo python:

```
app = Flask(__name__, static_folder='static')
```

Esto le indica a Flask que los archivos estáticos se encuentran en una carpeta llamada static. Quedando el esquema del proyecto de esta forma ahora. También hay que cambiar la dirección de la línea de código del html donde se indica donde está el css.

```
- chatbot.py
- static
  - styles.css
- templates
  - index.html
```

```
<link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename='styles.css') }}">
```

Ya funcionan los estilos, ahora surge otro tipo de problema cuando se le escribe algo al chatbot este te responde dos veces lo mismo por ejemplo.

```
Human: hola AI:Hola, ¿en qué puedo ayudarte?
AI: Hola, ¿en qué puedo ayudarte?
```

Una posible solución era mover la línea donde se añade la respuesta a la conversación justo antes de enviar la conversación al html. Pero esto no funcionó.

Otra causa que pudo ocasionar esto eran los valores del modelo, se cambiaron pero no soluciono el problema, así que se restablecieron a los originales

Otra prueba que se hizo fue cambiar la variable conversation, al principio fue una cadena de caracteres y se cambió por un vector de caracteres, cambiando el código del archivo python para que este funcionara guardándose en los distintos espacios del vector. Pero esto ocasionó más errores.

Ya que los errores indicaban que el parámetro prompt que se le pasaba al modelo no tenía un formato válido porque conversation era un vector, para ello una idea para solucionarlo era formatear la variable conversation. Y utilizar esta nueva variable como el valor del parámetro prompt.

```
formatted_conversation = "\n".join([f"{turn['role']}: {turn['text']}"
for turn in conversation])
```

Cambiar la variable conversación por un vector solo causó errores por eso se volvió a dejar como un string. Tras corregir el código python se pensó que el error sería del html y se añadió código en el div de conversación.

```
<div id="conversation">
        {% for turn in conversation %}
            {% if turn.role == 'Human' %}
                <p class="human">{{ turn.text }}</p>
            {% elif turn.role == 'AI' %}
                <p class="ai">{{ turn.text }}</p>
            {% endif %}
        {% endfor %}
    </div>
```

Esto tampoco soluciono nada y dio más errores ya que no reconoció bien el proyecto estas líneas de código. Así se estuvieron toqueteando ambos archivos hasta encontrar otra posible solución y se remodeló casi toda la estructura para solucionar el problema pero no resultó y se restableció todo como estaba al inicio.

También se probó creando varios if en el archivo python para que si reconocía la palabra AI sin un salto de línea anterior lo eliminase hasta el siguiente AI, para que así solo apareciese uno y en una línea distinta a la pregunta del humano, pero tampoco resultó.

Otra solución fue crear dos variables conversation_input y conversation_output, para formatear la de entrada para ponerle el formato requerido por el OpenAI, luego agregarle la pregunta actual, unirla a una sola cadena de texto, realizar la solicitud al modelo, agregar la pregunta y la respuesta actual a la conversación de salida y enviarla al html. Esto tampoco lo soluciono, daba un resultado que no era el esperado de pregunta salto de línea y respuesta.

Finalmente mirando el codigo poco a poco del html y python y ejecutando línea a línea con el modo debug y viendo lo que se guardaba en las variables se cambió un poco el código dejando solo una conversación cambiando la forma en la que se introducen las preguntas y respuestas escribiendo en ellas como se ponen los saltos de línea en html para que éste los reconozca y enviando solo al html la conversación ya que se enviaba también la respuesta, asi funciono correctamente con un formato pregunta dos saltos de línea y respuesta.

- **Segundo Objetivo**

Ahora que se tiene el chatbot funcionando en el localhost y con la conversación apareciendo correctamente, se quiere entrenar al bot para que se pueda utilizar para ayudar a personas con dependencia.

Para ello hay que crear un conjunto de entrenamiento, utilizar datos anotados para crear un conjunto en el formato requerido por la plataforma de entrenamiento, para ello se usa un formato de entrada y salida de preguntas y respuestas donde cada pregunta se empareja con su respuesta correspondiente.

Para realizar esto se crea un archivo JSON, el cual contendrá una lista de ejemplos de entrenamiento, pregunta-respuesta. Cada uno tiene que tener dos propiedades, la propiedad question contendrá la pregunta formulada por el usuario y la propiedad answer contendrá la respuesta generada por el bot.

Para que el archivo python pueda utilizar las preguntas y respuestas entrenadas de un archivo JSON , hay que añadir código al inicio del archivo python para leer el contenido del JSON. Se puede importar la biblioteca json para así leer el archivo JSON:

```
import json

# Cargar el archivo JSON
with open('entrenamiento.json', 'r') as file:
    data = json.load(file)

# Obtener las preguntas y respuestas
preguntas = data['preguntas']
respuestas = data['respuestas']
```

Además hay que modificar la función chat para buscar la respuesta correspondiente en el archivo JSON en lugar de llamar a la API de OpenAi. Se puede comparar la pregunta del usuario con las preguntas almacenadas en preguntas y obtener la correspondiente desde respuestas.

```
 # Buscar la pregunta en el archivo JSON y obtener la respuesta
correspondiente
    answer = None
    for i, pregunta in enumerate(preguntas):
        if pregunta == question:
            answer = respuestas[i]
            break

    if answer is None:
        answer = "Lo siento, no tengo una respuesta para esa pregunta."
```

De esta forma el chatbot solo responderá si se le pregunta una de las preguntas que tiene entrenadas, con el resto dirá que no tiene respuesta y no se quiere eso. Si se desea que el chatbot responda normalmente cuando se hace una pregunta que no está en el archivo JSON, se puede combinar la búsqueda en el archivo con la llamada a la API de OpenAI, para ello hay que volver a modificar la función chat.

```
# Buscar la pregunta en el archivo JSON y obtener la respuesta
correspondiente
    answer = None
    for i, pregunta in enumerate(preguntas):
        if pregunta == question:
            answer = respuestas[i]
            break

    if answer is None:
        # Si la pregunta no está en el archivo JSON, llamar a la API de
OpenAI
```

De esta manera el chatbot primero buscará la pregunta en el archivo JSON y si no la encuentra proporcionará la respuesta correspondiente. Si la pregunta no está en el archivo, el chatbot llamara a la API de OPenAI para obtener una respuesta utilizando el código que ya había implementado.

Esto le dará flexibilidad al chatbot para responder a una amplia variedad de preguntas.

Ahora aparece un error que indica que no se encuentra el archivo de entrenamiento es decir el JSON, para ello hay que meterlo en otra carpeta en la raíz del proyecto quedando el árbol de la estructura de esta forma:

```
- MiProyecto/
  - chatbot.py
  - data/
    - entrenamiento.json
  - templates/
    - index.html
  - static/
    - styles.css
```

Y cambiado la siguiente línea en python:

```
with open('/ruta/completa/al/directorio/data/entrenamiento.json', 'r')
as file:
    data = json.load(file)
```

Aun cambiando todo esto daba error pero se soluciono cambiando esto en el python:

```
# Obtener la ruta absoluta del archivo
file_path = os.path.abspath('data/entrenamiento.json')

# Abrir el archivo utilizando la ruta absoluta
with open(file_path, 'r') as file:
```

Tras solucionar esto aparece otro error que indicaba que la clave preguntas no se encuentra en el archivo JSON cargado, la estructura del JSON no coincide con la esperada por el código, para ello tenemos que modificar un par de líneas de la obtención de las preguntas y respuestas del archivo.

```
# Obtener las preguntas y respuestas
examples = data['examples']
preguntas = [example['question'] for example in examples]
respuestas = [example['answer'] for example in examples]
```

Esto hace que el error desaparezca pero hace que solo responda cuando la pregunta está exactamente igual, y pensándolo bien una persona puede formular una pregunta de muchas formas distintas mejorando así la experiencia del usuario si esto se puede implementar, aquí se origina el siguiente objetivo de la aplicación.

Este código calcula la similitud de coseno entre la pregunta del usuario y las preguntas almacenadas.

```
question_vector = vectorizer.transform([question_preprocesada])
similarities = cosine_similarity(question_vector, vectorizer)[0]
max_similarity = max(similarities)
index = similarities.tolist().index(max_similarity)
```

Esto da un error que indica que la biblioteca NLTK no está instalada en el entorno, hay que instalarla con el siguiente comando en el entorno :

```
pip install nltk
```

Después de la instalación aparece otro error similar que indica que la biblioteca scikit-learn no está instalada en el entorno. Para solucionarlo se instala con el siguiente comando:

```
pip install scikit-learn
```

A continuación se produce otro error que indica que no se encontró el recurso stopwords de NLTK , esto son palabras comunes que se consideran irrelevantes en el procesamiento del lenguaje natural y generalmente se eliminan del texto.

Para solucionar este problema, se debe descargar el recurso utilizando el NLTK downloader. Se ejecuta el siguiente código en el entorno.

```
import nltk
nltk.download('stopwords')
```

Esto no funcionó, no se pudo descargar el recurso debido a un problema de conexión, una solución alternativa es descargar manualmente el archivo de stopwords y guardarlo en la ubicación adecuada. Se descarga stopwords.zip, se extrae el contenido y se mueve a /home/claudia/nltk_data/corpora/

Se ejecuta nuevamente el código tras realizar estos pasos, esto soluciona el problema anterior pero produce uno nuevo ya que falta el recurso punkt que se utiliza para el tokenizado de oraciones. Para solucionarlo descargamos el recurso en el entorno.

```
python -m nltk.downloader punkt
```

Vuelve a dar un error al descargar como el recurso anterior, para ello se va a hacer manualmente, descargando el .zip, guardandolo en una ubicación accesible para el sistema, descomprimiendolo para obtener la carpeta punkt y moviéndola a la ruta /home/claudia/nltk_data/tokenizers/.

Se vuelve a ejecutar el código y aparece un error que el objeto vectorizer no tiene atributo transformer indica, para solucionarlo se verifica si se ha importado el vectorizador adecuado en el archivo python .

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
```

También hay que reemplazar la línea donde se inicializa el vectorizer

```
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(preguntas_preprocesadas)
transformer = TfidfTransformer()
X_train_transformed = transformer.fit_transform(X_train)
```

En este código, primero se crea una instancia de TfidfVectorizer y se ajusta el vectorizador a los datos del JSON. Luego, se crea una instancia de TfidfTransformer y se ajusta el transformer a los datos transformados por el vectorizer.

En la función chat se cambia la línea donde se calcula la question_vetor por :

```
question_vector = vectorizer.transform([question_preprocesada])
question_vector_transformed = transformer.transform(question_vector)
```

Todo esto soluciona el anterior problema pero genera otro que indica que se está pasando un objeto TfidVectorizer como argumento a una función que espera un número o una cadena como argumento float.

El problema radica en la línea similarities = cosine_similarity (question_vector, vectorizer)[0]. La función cosine_similarity espera matrices numéricas como entrada, pero se está pasando vectorizer, que es una instancia de TfidfVectorizer.

Para solucionar este problema, se debe pasar la matriz de características X_train_transformed en lugar del objeto vectorizer a la función cosine_similarity.

```
similarities = cosine_similarity(question_vector_transformed,
X_train_transformed)[0]
```

Con esto ya se solucionan todos los problemas, se cumplen todos los objetivos, funcionando así correctamente el chatbot.

3. Conclusión

Como ya se tiene una primera version de ejemplo funcionando, en los próximos sprints se completará el archivo JSON con mas preguntas y respuestas especializadas en el tema de ayuda a personas en situación de dependencia, además de hacer más visual la interfaz de usuario de la página web ya que es muy básica.

Además de añadir nuevas implementaciones como por ejemplo un registro de usuarios para que se identifiquen al comenzar la conversación con el chat además de crear nuevos botones para que haya rutas más fáciles y más accesibles para ciertas preguntas o peticiones más comunes.

4. Enlaces

- https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/packages/tokenizers/punkt.zip.

- stopwords.zip