Arabot-Question-Answering

November 19, 2023

Hello Everyone

0.1 Question-Answering System with FastAPI

I. Introduction * Overview of the project purpose and functionality. * Mention of key technologies: FastAPI, Hugging Face Transformers.

II. Components and Libraries Used

- FastAPI: Brief explanation of why FastAPI is chosen for building the web service.
- Hugging Face Transformers: Explanation of its role in loading and utilizing pre-trained question-answering models.

III. Project Structure

- FastAPI Setup:
  - Creation of endpoints (/answer and /batch-answer).
- Question-Answering Model:
  - Integration of a pre-trained model (deepset/roberta-base-squad2).
  - Utilization of a question-answering pipeline.
- IV. Endpoints
  - Single Question Answering (/answer):
    - Acceptance of a JSON payload with "context" and "question" fields.
    - Generation of an answer using the question-answering pipeline.
    - Return of the generated answer in JSON format.
  - Batch Question Answering (/batch-answer):
    - Acceptance of a JSON payload with a list of items, each containing "context" and "question" fields.
    - Generation of answers for each item in the batch.
    - Return of a CSV file (streaming response) with columns for the original question, original answer (if provided), and the generated answer.

V. Usage Examples

- Demonstrations of how to use both single and batch question-answering endpoints.

[8]:
```
from fastapi import FastAPI, HTTPException
from typing import Dict, List
from transformers import pipeline, AutoTokenizer, AutoModelForQuestionAnswering
```

```
from fastapi.responses import StreamingResponse
import datasets
```

Let's Create the apis for the models

[4]:
```
# Creating an instance of the FastAPI class
app = FastAPI()

# Load the pre-trained model and tokenizer
model_name = "deepset/roberta-base-squad2"
qa_pipeline = pipeline("question-answering", model=model_name,␣
 ↪tokenizer=model_name)

# Endpoint for answering a single question
@app.post("/answer")
def get_answer(data: Dict):
    # Extracting context and question from the request data
    context = data["context"]
    question = data["question"]

    # Using the question-answering pipeline to get the answer
    answer = qa_pipeline(question=question, context=context)

    # Returning the answer in JSON format
    return {"answer": answer["answer"]}

# Endpoint for batch question-answering
@app.post("/batch-answer")
def get_batch_answers(dataset: List[Dict]):
    # Initializing empty lists for answers and CSV data
    answers = []
    csv_data = "question,original_answer,generated_answer\n"

    # Iterating over each item in the dataset
    for item in dataset:
        # Extracting context, question, and original answer (if provided)
        context = item["context"]
        question = item["question"]
        original_answer = item.get("answer", "")

        # Using the question-answering pipeline to get the generated answer
        generated_answer = qa_pipeline(question=question,␣
 ↪context=context)["answer"]

        # Appending the results to the answers list
        answers.append({
            "question": question,
            "original_answer": original_answer,
```

```
            "generated_answer": generated_answer,
        })

        # Adding the data to the CSV string
        csv_data += f"{question},{original_answer},{generated_answer}\n"

    # Setting up response headers for streaming CSV file
    response = StreamingResponse(iter([csv_data]), media_type="text/csv")
    response.headers["Content-Disposition"] = "attachment;␣
↪filename=generated_answers.csv"

    # Returning the streaming response
    return response
```

0.2 Testing the single question answering

[5]:
```
import requests

url = "http://127.0.0.1:8000/answer"
data = {
    "context": "The blue whale is the largest mammal on Earth.",
    "question": "What is the largest mammal on Earth?"
}

response = requests.post(url, json=data)

print(response.status_code)
print(response.json())
```

```
200
{'answer': 'The blue whale'}
```

[6]:
```
import requests

url = "http://127.0.0.1:8000/answer"
data = {
    "context": "Camel is created for desert.",
    "question": "what camel is made for?"
}

response = requests.post(url, json=data)

print(response.status_code)
print(response.json())
```

```
200
{'answer': 'desert'}
```

0.3 Testing the batch question answering on SQuad dataset

[19]:
```
import requests
import csv
from io import StringIO


#load dataset

dataset=datasets.load_dataset("squad")

# Assuming you have separate lists for context, question, and answers
context = dataset["train"]["context"][:100]
question = dataset["train"]["question"][:100]
answers = dataset["train"]["answers"][:100]

# Create fake_data using a loop
fake_data = [
    {
        "context": ctx,
        "question": qst,
        "answer": ans["text"] if ans else ""  # Assuming "answers" is a list of␣
 ↪dictionaries
    }
    for ctx, qst, ans in zip(context, question, answers)
]

# Define the URL for the /batch-answer endpoint
url = "http://127.0.0.1:8000/batch-answer"

# Send a POST request to the /batch-answer endpoint
response = requests.post(url, json=fake_data)

# Assuming csv_data contains the CSV content from the response
csv_data = response.text

# Use StringIO to create a file-like object
csv_file = StringIO(csv_data)

# Read the CSV file
csv_reader = csv.reader(csv_file)
header = next(csv_reader)  # Read the header
data = list(csv_reader)    # Read the rest of the data

# Save the CSV data to a file
output_file_path = "generated_answers_from_api.csv"
with open(output_file_path, "w", newline="", encoding="utf-8") as output_file:
```

```
    # Create a CSV writer
    csv_writer = csv.writer(output_file)

    # Write the header
    csv_writer.writerow(header)

    # Write the data
    csv_writer.writerows(data)

print(f"CSV data saved to: {output_file_path}")
```

```
CSV data saved to: generated_answers_from_api.csv
```

[ ]:

[ ]:

we use this command to set the wheels for the project

- uvicorn: This is the ASGI server that you're using to run your FastAPI application.

With command:

- uvicorn QA_model:app –reload

0.3.1 END of the task ! ‘