BANGLA CHATBOT DOCUMENTATION

Group: 22

Members

20101581 Tahsin Zaman Jilan
20101459 Maisha Shabnam Chowdhury
20301204 Prima Sarker
20301471 Kazi Shahed Mamun
20141013 Swapnil Sarker

Introduction :

Natural Language Processing (NLP) has been witnessing an upsurge of development, especially with chatbots that engage in human-like conversations. The demand for chatbots will always rise with technology. Chatbots with regional languages are helping people get technological help more. This project involves a Bangla chatbot and Neural Network model, which is designed to comprehend and respond to queries of the user in Bangla. Two datasets were used for the project in order to train the chatbot, which includes a diverse range of conversational contexts. Moreover, to achieve enhanced performance and language understanding, we have used several models for ensuring Bangla language. BanglaBERT has been used for running the QA models, DistilBERT model for tokenizing our data and "Sequential" model class of Keras for defining the Neural Network architecture for our model. The bot responds in Bangla to any question asked to it in Bangla. Additionally, we encountered many challenges during developing the project, which were discussed as potential sectors for future research and improvement as well.

Purpose:

The project aimed to create a helpful chatbot called BanglaBot that understands and communicates in Bengali. It allows people to ask questions in Bengali and gives them the answers in the same language. This makes it easier for users to communicate in their native language and brings Bengali into the conversation with technology. Our goal is to make it easier and more accessible for Bengali-speaking users, to communicate and get information more comfortably.

Models Used:

We have used different models for different purposes. To tokenize our data we have used the Distillbert Model And for QA function we have used BanglaBert and Sequential Model.

First Model is DistilBERT Model:

DistilBERT is a transformer model that is quicker and smaller than BERT. It was pre trained using the BERT foundational model as a teacher on the identical corpus in a self-supervised manner. This means that it was pre trained using only the raw texts and an algorithmic process that used the BERT base model to produce inputs and labels derived from these texts. No human labeling was done, which allows it to leverage a large amount of publically available data. In this manner, the model is quicker for inference or downstream tasks while learning the same inner picture of the English language as its instructor model.

Code explanation :

```python
import json
from transformers import AutoModelForQuestionAnswering, AutoTokenizer
import torch
import random

# Load the pre-trained model and tokenizer
model_path = "shams/banglabert-finetuned-squad"
tokenizer_path = "distilbert-base-uncased"
model = AutoModelForQuestionAnswering.from_pretrained(model_path)
tokenizer = AutoTokenizer.from_pretrained(tokenizer_path)
```

The code snippet demonstrates the usage of various Python libraries and Hugging Face's transformers library for natural language processing tasks. It imports necessary modules like json, transformers, torch, and random. Specifically, it imports classes AutoModelForQuestionAnswering and AutoTokenizer from the transformers library, enabling access to pre-trained models and tokenizers for NLP tasks.

The code also defines variables model_path and tokenizer_path, representing paths or identifiers for a fine-tuned question-answering model (trained on SQuAD in Bengali) and a DistilBERT tokenizer for English text, respectively.

To load the pre-trained model and tokenizer, the from_pretrained method is used for both AutoModelForQuestionAnswering and AutoTokenizer, initializing them with the specified paths. This process loads the pre-trained model fine-tuned for answering questions in Bengali and the associated tokenizer designed for processing English text.

```python
tokenized_example = tokenizer(
    example['question'][0]["input_text"],
    example["story"],
    max_length=max_length,
    truncation="only_second",
    return_overflowing_tokens=True,
    return_offsets_mapping=True,
    stride=doc_stride
)
print(tokenized_example["offset_mapping"][0][:100])
```

In natural language processing, the process involves breaking text into tokens for analysis. The tokenizer helps map these tokens back to their positions in the original text. When answering questions or extracting

information, the model predicts potential answers within the text. By leveraging the offset_mapping from the tokenizer, the model identifies where these answers correspond in the tokenized text, providing start and end positions for precise extraction. This approach streamlines the process of finding answers within the text by using tokenization and offset_mapping to link the tokenized text back to its original context, enabling accurate information extraction.

```
10 print(tokenized_example["offset_mapping"][0][:100])
```

```
[(0, 0), (0, 6), (6, 11), (12, 19), (19, 22), (23, 24),
```

Features :

Using the Hugging Face transformers library, the code configures a script to import a pre-trained question-answering model and tokenizer. While the tokenizer appears to use DistilBERT to handle English text, the model appears to be optimized for answering queries in Bengali. For processing and using pre-trained models for different NLP applications, especially question answering, these tools are essential.

- Transformer-based pre-trained language model DistilBERT was trained on a vast corpus of textual data. Through tasks like next sentence prediction and masked language modeling, it learns word and sentence representations in an unsupervised way.
- Efficiency & Speed: Compared to the original BERT model, DistilBERT is intended to be faster and more computationally efficient during both training and inference. It is more suited for applications with constrained processing resources because of its efficiency.

Second Model Description(BANGLABERT):

Our own model, **BanglaBERT**, serves as evidence of our dedication to enhancing the capabilities of natural language processing for the Bengali language. BanglaBERT is designed with painstaking attention to detail, utilizing cutting-edge methods in deep learning and transfer learning, to effectively tackle the intrinsic difficulties present in Bengali text data.

Linguistic Nuances: BanglaBERT beyond fundamental language comprehension by exploring the intricacies of Bengali linguistics. The model has been taught to accurately identify and understand the complex subtleties, idiomatic phrases, and expressions that are inherent in Bengali language.
Contextual Sensitivity: One notable attribute of BanglaBERT is its exceptional ability to capture subtle contextual cues. The model demonstrates exceptional proficiency in comprehending the semantics of words and phrases in the context of the full sentence, facilitating more precise and contextually-sensitive natural language processing.

Domain Adaptability: BanglaBERT is specifically engineered to possess high adaptability, allowing for effortless adaptation to several domains and sectors. The model has strong performance across several applications, such as sentiment analysis in customer evaluations, classification of news items, and information extraction from Bengali documents.
Fine-tuning Capabilities: BanglaBERT offers the opportunity for academics and practitioners to customize the model according to certain tasks or domains. Its capacity to adapt makes it a significant resource for projects with distinct requirements or specialized datasets.
Multilingual Considerations: Although Bengali is its main focus, BanglaBERT demonstrates a certain degree of proficiency in understanding many languages. The architecture of the system allows it to comprehend both the similarities and contrasts between Bengali and other languages, giving it a flexible option for multilingual natural language processing applications.
Continuous Improvement: Our dedication to perfection extends to the continuing enhancement of our models. We conduct extensive research and development to improve the capabilities of BanglaBERT, ensuring that it remains at the forefront of breakthroughs in natural language processing in the Bengali language sector.

Code explanation :

```
1 import json
2 from transformers import AutoModelForQuestionAnswering, AutoTokenizer
3 import torch
4 import random
5
6 # Load the pre-trained model and tokenizer
7 model_path = "shams/banglabert-finetuned-squad"
```

Library Imports:

The code first imports the json module, allowing JSON data handling within Python scripts. Additionally, it utilizes specific classes (AutoModelForQuestionAnswering and AutoTokenizer) from the transformers library. These classes are crucial for interacting with and accessing pre-trained models and tokenizers used in natural language processing tasks.

Pre-trained Model and Tokenizer Loading:

model_path = "shams/banglabert-fine tuned-squad": Defines the path or identifier for a specific pre-trained question-answering model.

Missing Initialization:

The code lacks the specific lines to initialize the AutoModelForQuestionAnswering and AutoTokenizer using the model_path. Typically, this involves using

“AutoModelForQuestionAnswering.from_pretrained(model_path)” and AutoTokenizer.from_pretrained(model_path) to load the pre-trained model and tokenizer respectively.

Features:

Tokenization: The tokenizer is used to tokenize the input question and context, converting them into tensors suitable for the model.
Model Inference: The tokenized inputs are passed through the model to obtain start and end logits.
Answer Span Extraction: The indices corresponding to the start and end positions of the answer span are determined based on the maximum scores from the model's output.
Answer Decoding: The answer span is decoded using the tokenizer to obtain the final answer.
User Interaction Loop: The code includes a loop for user interaction, allowing the user to input questions. The loop can be terminated by entering "exit" or "quit."
Context and Bot Response: A predefined context is provided, and the bot's response to the user's input is obtained using the answer_question function.

Third Sequential Model :

Neural network topologies can be created with the Sequential model class in Keras, which depicts a linear stack of layers where each layer delivers its output sequentially to the next layer in the stack. In order Nature: Data travels across the network from input to output as layers are added one after the other. Simplicity: Simple linear designs are particularly easy to construct and comprehend. Cons: Less adaptability than a functional API when building complicated models with several shared layers or inputs/outputs.

Code explanation :

Loading Intents:
with open("intents.json", "r", encoding="utf-8") as file:: Opens the 'intents.json' file in read mode and loads its content using json.load(file)["intents"]. It seems to assume that the JSON file contains a structure where the intents are stored under the "intents" key.

answer_question Function:

def answer_question(user_input, intents):: Defines the function answer_question that takes user_input and intents as arguments.

Intent Matching:

It iterates through each intent's patterns in the loaded intents data and checks if any of these patterns are present in the user_input. If a match is found, it returns a random response associated with the matched intent and assigns a high confidence value of 1.0.

Question-Answering Model Fallback:

If no match is found among the predefined intents, it attempts to use a question-answering model (indicated by the tokenizer and model objects that should have been initialized earlier in the code). It tries to tokenize the user_input with the tokenizer object (tokenizer(user_input, return_tensors="pt")). If successful, it utilizes the previously initialized question-answering model to obtain logits (start_scores and end_scores) for potential answer spans within the user_input.

Error Handling:

Exception Handling: If there's an exception during tokenization, it catches the exception, prints an error message, and returns a default response indicating that it couldn't understand the input, setting the confidence for the response to

```python
def answer_question(question, context):
    # Tokenize the question and context
    inputs = tokenizer(question, context, return_tensors="pt")

    # Get model outputs for start and end positions of the answer
    with torch.no_grad():
        outputs = model(**inputs)
    start_scores = outputs.start_logits
    end_scores = outputs.end_logits

    # Find the best start and end position indices
    start_index = torch.argmax(start_scores)
    end_index = torch.argmax(end_scores) + 1
    # Get the answer using the indices and decode it
    answer = tokenizer.decode(inputs["input_ids"][0, start_index:end_index])

    return answer
```

Feature:

1. Intent-Based Response Handling:

- Intent Data: Loads intent data from a JSON file (intents.json) that presumably contains predefined intents, their associated patterns, and responses.
- Matching User Input to Intents: Compares the user's input with predefined intent patterns. If there's a match, it randomly selects a response associated with the matched intent and assigns a high confidence value (1.0) for that intent.

2. Fallback to Question-Answering Model:

- Question-Answering Model Fallback: If the user's input does not match any predefined intents, the code attempts to use a question-answering model as a fallback.

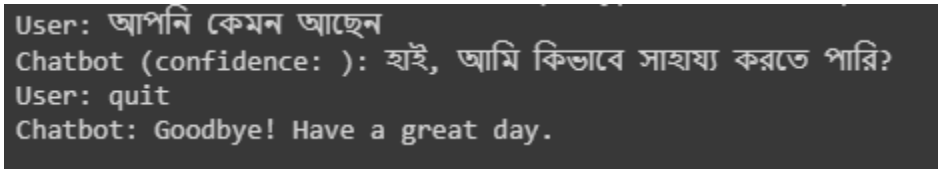
- Tokenization: Utilizes a tokenizer object (tokenizer) to tokenize the user's input, which is necessary for input formatting in the question-answering model.
- Question-Answering Model Inference: Attempts to generate answer logits (start_scores and end_scores) using a question-answering model based on the tokenized input.

3. Error Handling:

- Exception Handling: Catches exceptions that might occur during tokenization and provides an error message if there's an issue. In such cases, it returns a default response indicating that it couldn't understand the input and assigns a low confidence value (0.0) for that response.

4. Confidence Level:

- Confidence Level Assignment: Assigns confidence levels (1.0 for matched intents, 0.0 for errors) to indicate the reliability of the generated responses based on intent matching or the question-answering model.



Fourth Hybrid model (Sequential And BanglaBERT) :

Intent Matching:

The code checks if the user input matches any predefined intent patterns. If a match is found, a random response from the corresponding intent is selected, and a high confidence score (1.0) is returned.

Question-Answering Model Fallback:

If no intent is matched, the code attempts to use the DistilBERT-based question-answering model to generate a response. It tokenizes the user input and uses the model to predict the start and end positions of the answer span. If the confidence, calculated based on the softmax probabilities of start and end positions, is above a threshold (0.5), it considers the answer as valid and returns it with the calculated confidence. Otherwise, it returns a default response with low confidence.

Error Handling:

The code includes error handling for tokenization. If an error occurs during tokenization, it catches the exception, prints an error message, and returns a default response indicating that it couldn't understand the input, with low confidence.

```python
def answer_question_based_on_intent(user_input, intents):
    # Check if user input matches any intent patterns
    for data in intents:
        for pattern in data["patterns"]:
            if pattern in user_input:
                return data["responses"][0]

    return None

# Load intents from a separate file
with open("intents.json", "r", encoding="utf-8") as file:
    intents = json.load(file)["intents"]

while True:
    user_input = input("User: ")
    if user_input.lower() in ["exit", "quit"]:
        break

    # Check if there's a specific response based on intent
    intent_response = answer_question_based_on_intent(user_input, intents)
    if intent_response:
        print("Chatbot:", intent_response)
    else:
        for i, example in enumerate(datasets["train"]):

            context_response = answer_question_with_context(user_input,example['story'])
            if i==1:
                break
        print("Chatbot:", context_response)
```

Feature:

The chatbot features a user-friendly interaction loop where the bot responds to user input, providing answers based on intent matching or utilizing the question-answering model. It includes an improved exit condition, allowing users to type keywords like "exit," "quit," "goodbye," or "see you later" to end the conversation gracefully.

```
User: আপনি কে?
Chatbot: আমি টেড, একটি ডিপ-লার্নিং চ্যাটবট
User: টিকফা চুক্তিতে স্বাক্ষর করলে বাংলাদেশের জাতীয় নিরাপত্তা কাদের হাতে চলে যাবে?
Chatbot: যুক্তরাষ্ট্রের
User: exit
```

Dataset Description:

1. Introduction

Two different datasets that are specifically designed for different natural language processing tasks are included in this detailed description of datasets. We used two of the datasets, the first of which is “intents.” This dataset is essential for training and testing conversational agent models, which are intended to manage a variety of Bengali user interactions. It is curated and sourced by hand. The "alldata" dataset is used for training and assessing question-answering models, with an emphasis on contextual comprehension and information retrieval in Bengali narratives.

2. Dataset Origin

2.1 Chatbot Development Dataset: "intents" :

The "intents" dataset is a synthesis of diverse datasets, including machine translation (MT), abstractive text summarization (TS), question answering (QA), multi-turn dialogue generation (MTD), news headline generation (NHG), and cross-lingual summarization (XLS). All of these datasets work as catalysts to create a flexible chatbot system that improves language comprehension, response speed, user query accuracy, coherence in multi-turn conversations, news-related content generation, and multilingual comprehension and response capabilities.

2.2 Question-Answering Dataset: "alldata":

Collected from actual events, the "alldata" dataset includes a variety of narratives and questions that go along with them. It is inspired by diverse contexts and offers a demanding training environment for question-answering models.

3. Dataset Structure

3.1 Chatbot Development Dataset: "intents"

The "intents" dataset is carefully designed to capture the subtleties of conversations between users. It groups data according to various user intents, giving each one a distinct tag. For example, the dataset contains intents like "Google" for user inputs pertaining to Google searches and "thanks," "goodbye," and "greeting" for typical conversational situations. Every intent consists of a collection of patterns that indicate probable user inputs that cause the intent to occur. Every intent also has predefined responses linked to it, which show the anticipated responses from the chatbot. This framework makes it easier for chatbot models to be trained to comprehend user intent, spot patterns in user input, and produce appropriate, well-reasoned responses.

3.2 Question-Answering Dataset: "alldata"

The "alldata" dataset's structure is intended to make it easier to train and assess question-answering models. The dataset's entries are all labeled with unique identifiers, version information, and source details. Stories in Bengali that address a range of real-world situations are included in the entries. Each

entry has questions related to it that are divided into categories based on turn IDs and cover various aspects of the story. The dataset also contains answers, with the answer span itself, its start and end points within the narrative, and the related input text specified. By arranging the dataset in this way, it guarantees that there is ample context available for assessing how well question-answering models can understand and extract information from intricate stories.

4. Linguistic Context

These datasets only use Bengali, which guarantees a linguistic context that is in line with the tasks related to natural language understanding.

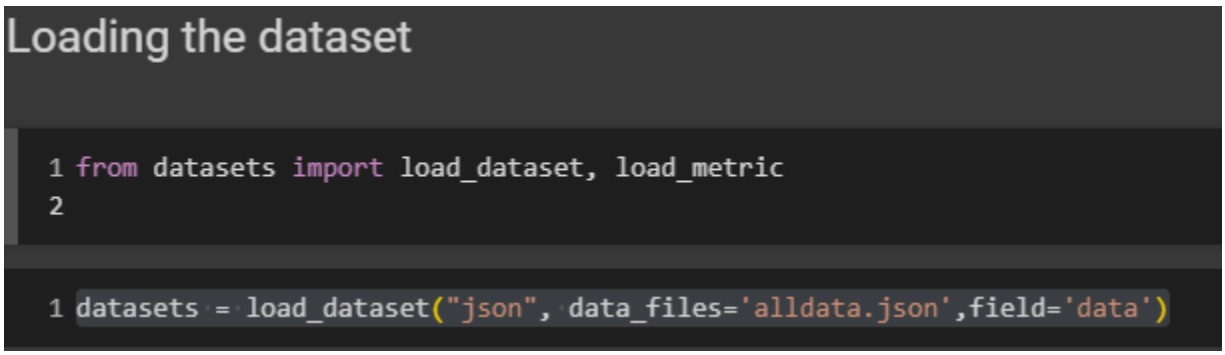
5. Contextual Considerations

The "alldata" dataset is specifically structured to encourage contextual understanding, making it difficult for question-answering models to understand and retrieve information from complex narratives. In contrast, the "intents" dataset has a "context" field for each intent.

6. Use Cases

These datasets are meant for particular uses in the field of natural language processing. While the "alldata" dataset supports question-answering models by providing them with tasks such as accurate information retrieval and contextual comprehension, the "intents" dataset supports research into the development of chatbot systems that can handle a variety of user interactions.

7. Dataset preprocessing :



- from datasets import load_dataset, load_metric: This imports the load_dataset and load_metric functions from the datasets library. These functions are used to load datasets and evaluation metrics, respectively.
- datasets = load_dataset("json", data_files='alldata.json', field='data'):
    - load_dataset is invoked with the following parameters:
        - "json": Specifies the dataset type, indicating that the data is in a JSON format.
        - data_files='alldata.json': Specifies the file name ('alldata.json') containing the dataset.
        - field='data': Indicates the specific field within the JSON file where the data is located. In this case, it assumes that the dataset is stored under the 'data' field within the JSON file.

```
1 datasets["train"][0]
```


```
{'source': 'manual',
 'question': [{'input_text': 'টিকফা চুক্তিতে স্বাক্ষর করলে বাংলাদেশের জাতীয় নিরাপত্তা কাদের হাতে চলে যাবে?',
   'turn_id': 1},
  {'input_text': 'কোথায় বলা হয় যে টিকফা চুক্তিতে স্বাক্ষর করলে বাংলাদেশের জাতীয় নিরাপত্তা যুক্তরাষ্ট্রের হাতে চলে যাবে?',
   'turn_id': 2},
  {'input_text': 'কোবে দেশব্যাপী বিক্ষোভ দিবস পালন করার আহ্বান জানানো হয়?',
   'turn_id': 3},
  {'input_text': 'দলের সভাপতি কে?', 'turn_id': 4}],
 'filename': '1804.txt',
 'answers': [{'input_text': 'যুক্তরাষ্ট্রের',
   'span_end': 84,
   'span_start': 0,
   'span_text': 'টিকফা চুক্তিতে স্বাক্ষর করলে বাংলাদেশের জাতীয় নিরাপত্তা যুক্তরাষ্ট্রের হাতে চলে যাবে',
   'turn_id': 1},
  {'input_text': 'বাংলাদেশের ওয়ার্কার্স পার্টির পলিটব্যুরোর সভায় গৃহীত প্রস্তাবে',
   'span_end': 177,
   'span_start': 0,
   'span_text': 'টিকফা চুক্তিতে স্বাক্ষর করলে বাংলাদেশের জাতীয় নিরাপত্তা যুক্তরাষ্ট্রের হাতে চলে যাবে। গতকাল সোমবার বাংলাদেশের
   'turn_id': 2},
  {'input_text': '১৯ মে রোববার',
   'span_end': 261,
   'span_start': 199,
   'span_text': '১৯ মে রোববার দেশব্যাপী বিক্ষোভ দিবস পালন করার আহ্বান জানানো হয়',
   'turn_id': 3},
  {'input_text': 'রাশেদ খান ',
   'span_end': 284,
   'span_start': 263,
   'span_text': 'দলের সভাপতি রাশেদ খান ',
   'turn_id': 4}],
 'name': '1804.txt',
 'id': 'eb74840dbbd0840cffe3f1ad4ad5da1f',
 'story': 'টিকফা চুক্তিতে স্বাক্ষর করলে বাংলাদেশের জাতীয় নিরাপত্তা যুক্তরাষ্ট্রের হাতে চলে যাবে। গতকাল সোমবার বাংলাদেশের ওয়ার্কা
জানানো হয়।দলের সভাপতি রাশেদ খান মেননের সভাপতিত্বে অনুষ্ঠিত সভায় উপস্থিত ছিলেন সাধারণ সম্পাদক আনিসুর রহমান মল্লিক সা
```


- datasets**:** This variable likely contains the loaded dataset object. When using **the** load_dataset function from the datasets library, the loaded dataset is typically stored in a variable named datasets.
- ["train"]: Indicates that you want to access the "train" split of the dataset. Many datasets are divided into different subsets for training, validation, and testing. "train" refers to the portion of the dataset designated for training.
- [0]: Refers to the first element within the "train" split. It retrieves the first data sample or item from the training subset of the dataset.

8. Pre- processing the training data:

We must preprocess the texts so we can feed them into our model. This is accomplished by a Transformers Tokenizer, which will produce the other inputs the model needs and, as its name suggests, tokenize the inputs (including converting the tokens in question into their associated IDs in the pretrained vocabulary) and put them in a format the model expects. We get the vocabulary that was used to pretrain this particular checkpoint, and we receive a tokenizer that matches the model architecture that we wish to utilize.

Conclusion:

In conclusion, the BanglaBot is a simple chatbot that incorporates two different approaches to handle user input. First, it checks if the input aligns with predefined intent patterns and responds accordingly. If a specific intent is detected, the chatbot delivers a predefined response associated with that intent. Secondly, if no intent match is found, the chatbot utilizes a question-answering model with context.
BanglaBot showcases a hybrid approach by combining a predefined pattern with a model driven method to handle context-specific inquiries. The use of a pre-trained question-answering model allows the chatbot to generate responses based on the given context, offering a more dynamic interaction. The project involves the use of BanglaBERT for question answering, that is focusing on NLU, it discusses the use of recurrent neural networks (RNNs) and transformers for maintaining context in chatbot conversations, it mentions the development of predefined responses for various conversational scenarios. In summary, this project represents a thorough approach to NLP in Bengali, addressing specific challenges while establishing the stage for future study and advancement in the field of chatbot building and question-answering in Bangla language.