CREATE A CHABOT IN PYTHON

PHASE 3 : Development Part – 1

INTRODUCTION:

The development part of this phase involves building the chatbot by loading and pre-processing the dataset. The chatbot is built in this phase by preparing the environment and implementing basic user interactions. The libraries required are installed which includes transformers for model integration and flask for web app development.

Dataset link : https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot

DATASET LOADING:

STEP 1:

- Import the Pandas library with the alias 'pd'.
- Use the **pd.read_csv()** function to read the contents of 'dialogs.txt'.
- Specify that the data is tab-separated by setting the **sep** parameter to '\t'.
- Define the column names for the DataFrame as 'User' and 'Chatbot' using the **names** parameter.
- Finally, print the resulting DataFrame 'df'.

```
import pandas as pd

df=pd.read_csv('dialogs.txt',sep='\t',names=['User','Chatbot'])
print(df)
```

Output:

```
                                                  User  \
0                                hi, how are you doing?
1                          i'm fine. how about yourself?
2                   i'm pretty good. thanks for asking.
3                     no problem. so how have you been?
4                      i've been great. what about you?
...                                                 ...
3720    that's a good question. maybe it's not old age.
3721                          are you right-handed?
3722                              yes. all my life.
3723  you're wearing out your right hand. stop using...
3724        but i do all my writing with my right hand.

                                                Chatbot
0                          i'm fine. how about yourself?
1                   i'm pretty good. thanks for asking.
2                     no problem. so how have you been?
3                      i've been great. what about you?
4               i've been good. i'm in school right now.
...                                                 ...
3720                          are you right-handed?
3721                              yes. all my life.
3722  you're wearing out your right hand. stop using...
```

STEP 2:

```
[2]  import string
     import re
     # importing regular expressions
     punc_lower = lambda x: re.sub('[%s]' % re.escape(string.punctuation), ' ', x.lower())
     # Lower case conversion
     remove_n = lambda x: re.sub("\n", " ", x)
     # removing \n and replacing them with empty value
     remove_non_ascii = lambda x: re.sub(r'[^\x00-\x7f]',r' ', x)
     # removing non ascii characters
     alphanumeric = lambda x: re.sub('\w*\d\w*', ' ', x)
     # removing alpha numeric values
     df['User'] = df['User'].map(alphanumeric).map(punc_lower).map(remove_n).map(remove_non_ascii)
     # using map function and applying the function on query column
     df['Chatbot'] = df['Chatbot'].map(alphanumeric).map(punc_lower).map(remove_n).map(remove_non_ascii)
     # using map function and applying the function on response column
     print(df)
     df.to_csv('modified_dataset.csv', index=False)
```

- Import the 'string' and 're' modules.
- Define several lambda functions to perform specific text preprocessing tasks:
  - **punc_lower**: Removes punctuation and converts the text to lowercase.
  - **remove_n**: Removes newline characters and replaces them with spaces.
  - **alphanumeric**: Removes alphanumeric characters.
  - **remove_non_ascii**: Removes non-ASCII characters from the text.
- Apply these lambda functions to the 'User' and 'Chatbot' columns of the DataFrame 'df' using the **map** function. This cleans and preprocesses the text in these columns.
- Finally, print the modified DataFrame 'df'.
- Save the modified DataFrame to a CSV file named 'modified_dataset.csv' using **df.to_csv()**. The **index=False** parameter ensures that the index column is not included in the output CSV file.

Output:

```
                                                  User  \
0                                hi  how are you doing
1                          i m fine  how about yourself
2                   i m pretty good  thanks for asking
3                     no problem  so how have you been
4                      i ve been great  what about you
...                                                 ...
3720    that s a good question  maybe it s not old age
3721                              are you right handed
3722                                  yes  all my life
3723  you re wearing out your right hand  stop using...
3724        but i do all my writing with my right hand

                                                Chatbot
0                          i m fine  how about yourself
1                   i m pretty good  thanks for asking
2                     no problem  so how have you been
3                      i ve been great  what about you
4               i ve been good  i m in school right now
...                                                 ...
3720                              are you right handed
3721                                  yes  all my life
3722  you re wearing out your right hand  stop using...
3723        but i do all my writing with my right hand
3724  start typing instead  that way your left hand ...

[3725 rows x 2 columns]
```

STEP 3:

```python
import torch
import spacy
import random
import torch.nn as nn
import torch.optim as optim

# Data Preprocessing
nlp = spacy.load("en_core_web_sm")
with open("modified_dataset.csv", "r") as file:
    lines = file.readlines()
    data = [line.split("\t") for line in lines]
```

- **torch**: This is the PyTorch library, which is an open-source machine learning framework that provides support for both traditional machine learning and deep learning.
- **random**: The **random** module is a standard Python library for generating random numbers or performing random operations.

- **nn**: This is part of PyTorch and stands for "neural network." It includes classes and functions for building and training neural networks, such as various layers, loss functions, and optimizers.
- **optim**: A part of PyTorch, this module contains various optimization algorithms for training neural networks, like stochastic gradient descent (SGD).
- **nlp = spacy.load("en_core_web_sm")**: This line loads the spaCy language model "en_core_web_sm," which is trained for English text processing and includes various language features for NLP tasks.

```
[4]  with open("modified_dataset.csv", "r") as file:
         words = file.read().split()
         print(words)
         unique_words = set(words)
         print(len(unique_words))
         print(unique_words)
```

- **words = file.read().split()**: This line reads the content of the file and splits it into words using whitespace as the delimiter. It creates a list of words stored in the **words** variable.
- **spacy**: This is a popular natural language processing (NLP) library in Python. It's used for tasks such as tokenization, part-of-speech tagging, named entity recognition, and more.
- **unique_words = set(words)**: This line converts the list of words into a set, which automatically removes duplicate words. This is an efficient way to find unique words in the text.
-

Output:

```
['User,Chatbot', 'hi', 'how', 'are', 'you', 'doing', ',i', 'm', 'fine', 'how', 'about', 'yourself', 'i', 'm', 'fine', 'how', 'about', 'yourself', ',i', 'm', 'pretty', 'good', 'thanks', 'for',
2685
{'tired', 'poodle', 'song', 'dark', 'gardening', 'stress', 'plenty', 'book', 'male', 'cheating', 'personal', 'sleeping', 'could', 'department', 'shakespeare', ',oops', 'blood', 'trade', 'pursu
```

```
[5]  print(lines)

['User,Chatbot\n', 'hi  how are you doing ,i m fine  how about yourself \n', 'i m fine  how about yourself ,i m pretty good  thanks for asking \n', 'i m pretty good  thanks for asking ,no prob
```

```
[6]  print(data)

[['User,Chatbot\n'], ['hi  how are you doing ,i m fine  how about yourself \n'], ['i m fine  how about yourself ,i m pretty good  thanks for asking \n'], ['i m pretty good  thanks for asking ,
```

STEP 4:

```
[8]  vocabulary = {"<PAD>": 0, "<UNK>": 1}  # Initialize with special tokens
     with open("modified_dataset.csv", "r") as file:
         for line in file:
             words = line.strip().split()  # Split by whitespace for adapting the data format
             for word in words:
                 if word not in vocabulary:
                     vocabulary[word] = len(vocabulary)
```

Output:

```
[9] print(vocabulary)

: 11, 'yourself': 12, 'i': 13, 'pretty': 14, 'good': 15, 'thanks': 16, 'for': 17, 'asking': 18, ',no': 19, 'problem': 20, 'so': 21, 'have': 22, 'been': 23, 'no': 24, 've': 25, 'great': 26, 'wha
```

- **vocabulary = {"<PAD>": 0, "<UNK>": 1}**: This line initializes a dictionary called **vocabulary** with two special tokens:
  - **<PAD>**: Used to pad sequences to the same length.
  - **<UNK>**: Represents unknown or out-of-vocabulary words.
- **vocabulary[word] = len(vocabulary)**: If the word is not in the vocabulary, it adds the word to the vocabulary and assigns it a unique index. The index is determined by the length of the current vocabulary, effectively building a mapping from words to numerical indices.
- **words = line.strip().split()**: This line splits each line into words, assuming that words are separated by whitespace. It strips leading and trailing whitespace from the line and then splits it into words.

STEP 5:

- **SimpleChatbot(nn.Module)**: This class inherits from **nn.Module**, which is the base class for all PyTorch models. It indicates that **SimpleChatbot** is a PyTorch model.
- **def __init__(self, input_size, hidden_size, output_size)**: This is the constructor method for the **SimpleChatbot** class. It takes three parameters:
  - **input_size**: The size of the input vocabulary, which is the number of unique words in your dataset.

- **hidden_size**: The size of the hidden state of the GRU (Gated Recurrent Unit), which is a recurrent neural network layer.
- **output_size**: The size of the output vocabulary, which is typically the same as the input vocabulary in a chatbot model.

```
    # Convert tokens to indices using the provided vocabulary
    indices = [vocabulary.get(token, vocabulary["<UNK>"]) for token in tokens]
    indices = [min(idx, output_size - 1) for idx in indices]

    # Convert the list of indices to a PyTorch tensor
    tensor = torch.LongTensor(indices)

    return tensor
```

```
[11] class SimpleChatbot(nn.Module):
        def __init__(self, input_size, hidden_size, output_size):
            super(SimpleChatbot, self).__init__()
            self.embedding = nn.Embedding(input_size, hidden_size)
            self.gru = nn.GRU(hidden_size, hidden_size)
            self.out = nn.Linear(hidden_size, output_size)

        def forward(self, input):
            embedded = self.embedding(input)
            output, hidden = self.gru(embedded)
            output = self.out(output)
            return output, hidden
```

❖ **self.embedding = nn.Embedding(input_size, hidden_size)**: This line defines an embedding layer. It converts input word indices into dense vectors of **hidden_size**. This layer helps the model learn meaningful representations of words.

❖ **self.gru = nn.GRU(hidden_size, hidden_size)**: Here, you create a GRU layer. The GRU is a type of recurrent neural network layer that processes sequential data and produces hidden state vectors of size **hidden_size**. It's commonly used in sequence-to-sequence models for tasks like chatbots.

❖ **self.out = nn.Linear(hidden_size, output_size)**: This line defines a linear layer, which is used to map the output of the GRU to the desired output size. In a chatbot model, this helps generate a response that matches the expected output vocabulary.

❖ **def forward(self, input)**: This method defines the forward pass of the model, which is used for making predictions. It takes an input tensor (**input**) containing a sequence of word indices.

- **output, hidden = self.gru(embedded)**: The embedded input is passed through the GRU layer, which produces both an output and a hidden state. The hidden state is used to capture context and dependencies between words.
- **output = self.out(output)**: The GRU output is then passed through the linear layer to produce the final output, which is typically a probability distribution over the output vocabulary.

Install Required Libraries:

1.Transformers

```
pip install transformers

Collecting transformers
  Downloading transformers-4.34.1-py3-none-any.whl (7.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.7/7.7 MB 15.2 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.12.4)
Collecting huggingface-hub<1.0,>=0.16.4 (from transformers)
  Downloading huggingface_hub-0.18.0-py3-none-any.whl (301 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 302.0/302.0 kB 31.2 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.31.0)
Collecting tokenizers<0.15,>=0.14 (from transformers)
  Downloading tokenizers-0.14.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.8/3.8 MB 32.6 MB/s eta 0:00:00
Collecting safetensors>=0.3.1 (from transformers)
  Downloading safetensors-0.4.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 34.9 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transformers) (2023.6.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transformers) (4.5.0)
Collecting huggingface-hub<1.0,>=0.16.4 (from transformers)
  Downloading huggingface_hub-0.17.3-py3-none-any.whl (295 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 295.0/295.0 kB 30.4 MB/s eta 0:00:00
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.3.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2023.7.22)
Installing collected packages: safetensors, huggingface-hub, tokenizers, transformers
Successfully installed huggingface-hub-0.17.3 safetensors-0.4.0 tokenizers-0.14.1 transformers-4.34.1
```

- **Transformer Model**: A deep learning architecture introduced for sequence-to-sequence tasks.
- **Self-Attention**: Core mechanism to weigh word importance in the input sequence.
- **Multi-Head Attention**: Uses multiple attention heads to capture various dependencies.
- **Positional Encoding**: Incorporates word order information.
- **Stacked Layers**: Multiple layers capture complex patterns and dependencies.
- **Encoder-Decoder Architecture**: Common for tasks like translation.
- **Residual Connections and Normalization**: Enhance training stability.
- **Attention Masking**: Prevents certain positions from attending to others.

- **Position-wise Feed-Forward Networks**: Process each position in the sequence.
- **Subword Tokenization**: Handles large vocabularies efficiently.
- **State-of-the-Art NLP**: Transformers have revolutionized natural language processing and inspired advanced models.

2.Flask

```
[13] pip install flask

     Requirement already satisfied: flask in /usr/local/lib/python3.10/dist-packages (2.2.5)
     Requirement already satisfied: Werkzeug>=2.2.2 in /usr/local/lib/python3.10/dist-packages (from flask) (3.0.1)
     Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from flask) (3.1.2)
     Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-packages (from flask) (2.1.2)
     Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (from flask) (8.1.7)
     Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2>=3.0->flask) (2.1.3)
```

- **Flask**: A lightweight Python web framework.
- **Microframework**: Designed for simplicity and ease of use.
- **Routing**: Supports URL routing to define application endpoints.
- **Web Development**: Used for building web applications and APIs.
- **Extensible**: Allows integration with various extensions and libraries.
- **Widely Adopted**: Popular choice for developing web applications in Python.

CONCLUSION:

Data loading is the initial step in handling data for various applications, including research, analysis, and machine learning. It involves gathering data from different sources, ensuring it's in the correct format, and making it ready for further processing. Proper data loading is vital to maintain data quality and usability throughout a project, as it sets the foundation for successful data-driven tasks.