CIS 8389 GenAI Project Documentation

By Ayush Rishi

# 1. Introduction

This project involves building and fine-tuning a machine learning model for customer support response generation. The aim is to create a system capable of understanding customer inquiries (instructions) and generating appropriate responses based on a dataset of customer service interactions. The notebook contains steps to preprocess data, train a model, and evaluate its performance.

# 2. Data Cleaning and Preprocessing

## 2.1 Importing Libraries

The project uses the following libraries:

* Pandas for data manipulation.
* NLTK for natural language processing tasks like tokenization, lemmatization, and stopwords removal.
* Regex for text normalization.

## 2.2 Loading Data

The dataset is loaded using the pandas.read\_csv() method. This function reads a CSV file containing customer service interaction data, specifically focusing on instructions and responses.

## 2.3 Cleaning Data

The data cleaning process involves:

* Removing Unnecessary Columns: Only the 'instruction' and 'response' columns are retained.
* Removing Duplicates: Duplicate entries are dropped based on the 'instruction' and 'response' columns.
* Handling Missing Values: Rows with missing instructions or responses are dropped.
* Text Normalization: Special characters are removed, and the text is tokenized, stopwords are eliminated, and words are lemmatized to ensure uniformity.

## 2.4 Example of Cleaned Data

The cleaned data consists of:

* Instruction: Customer inquiries or issues.
* Response: The corresponding reply from the customer support agent.

# 3. Dataset Class for Model Input

A custom Dataset class is defined to handle the data efficiently for model training:

from torch.utils.data import Dataset  
  
class CustomerSupportDataset(Dataset):  
 def \_\_init\_\_(self, data, tokenizer, max\_length=512):  
 self.data = data  
 self.tokenizer = tokenizer  
 self.max\_length = max\_length  
  
 def \_\_len\_\_(self):  
 return len(self.data)  
  
 def \_\_getitem\_\_(self, idx):  
 input\_text = f"Customer support: {self.data.iloc[idx]['instruction']}"  
 target\_text = self.data.iloc[idx]['response']  
  
 inputs = self.tokenizer(input\_text, max\_length=self.max\_length, padding='max\_length', truncation=True, return\_tensors="pt")  
 targets = self.tokenizer(target\_text, max\_length=self.max\_length, padding='max\_length', truncation=True, return\_tensors="pt")  
  
 return {  
 "input\_ids": inputs.input\_ids.flatten(),  
 "attention\_mask": inputs.attention\_mask.flatten(),  
 "labels": targets.input\_ids.flatten()  
 }

## 3.1 Key Features

* Tokenization: Both input and target texts are tokenized using a tokenizer, padding and truncating them to a maximum length of 512 tokens.
* Custom Dataset: This class extends torch.utils.data.Dataset to provide a clean and efficient way to access tokenized data for model training.

# 4. Training Setup and Execution

The training process involves:

* Model Architecture: Using a pre-trained language model (likely from the Hugging Face library) for fine-tuning.
* Loss Function: Typically, a cross-entropy loss is used for text generation tasks.
* Optimization: Standard optimizers like Adam can be employed to fine-tune the model.

# 5. Evaluation

After training, the model is evaluated based on its ability to generate coherent and contextually relevant responses. The evaluation metrics could include:

* Loss Function: To check the model’s performance during training.
* Human Evaluation: Subjective assessment of the relevance and helpfulness of the generated responses.

# 6. Future Work

* Model Optimization: Further tuning of hyperparameters and fine-tuning with more data could improve performance.
* Deployment: Once trained, the model can be integrated into a real-world customer support system.