

Documentation Report

Assignment Title: Automated Response Generation for Customer Support

Objective: Build a model to generate automated responses to customer queries.

Dataset: <https://huggingface.co/datasets/Kaludi/Customer-Support-Responses> for performing the following tasks

Tasks:

- Explore and preprocess the dataset.
- Train a sequence-to-sequence (seq2seq) model or use a transformer-based model like GPT-3 for generating responses.
- Fine-tune the model for coherence and relevance.
- Evaluate the generated responses for quality and appropriateness.

Deliverables:

- Code and documentation
- Response generation model
- A demo where users can input a query and receive an automated response (implemented in a Jupyter notebook)

✚ **To achieve the tasks outlined in your assignment, we will follow a structured approach to building, training, and fine-tuning a model for automated response generation. Here's a step-by-step plan:**

Step 1: Explore and Preprocess the Dataset

1. Dataset Exploration:

- Load the dataset from Hugging Face.
- Understand the structure, data types, and contents of the dataset.
- Visualize the data distribution and identify any potential issues such as missing values or imbalanced classes.

2. Data Cleaning and Preprocessing:

- Handle missing or null values.
- Tokenize the text data.
- Split the dataset into training, validation, and test sets.

Step 2: Model Selection and Training

1. Choose a Model:

- Decide between a sequence-to-sequence (seq2seq) model or a transformer-based model like GPT-3. For this example, we will use a transformer-based model like GPT-2 or T5 for practical reasons.

2. Load Pretrained Model:

- Use a pretrained model from the Hugging Face library (e.g., GPT-2, T5).

3. Fine-Tune the Model:

- Fine-tune the selected model on the pre-processed dataset to generate responses that are coherent and relevant to customer queries.

Step 3: Fine-Tune for Coherence and Relevance

1. Training Loop:

- Set up a training loop with appropriate loss functions and optimizers.
- Monitor training and validation loss to avoid overfitting.
- Implement early stopping and checkpointing.

2. Hyperparameter Tuning:

- Experiment with different learning rates, batch sizes, and other hyperparameters to improve model performance.

Step 4: Evaluate the Model

1. Evaluation Metrics:

- Use BLEU, ROUGE, and other relevant metrics to evaluate the quality of generated responses.
- Manually inspect a sample of generated responses for coherence and appropriateness.

2. Test the Model:

- Generate responses for a set of test queries and compare them to actual responses.

Step 5: Deployment and Demo

1. User Interface:

- Implementing a simple interface in a Jupyter notebook where users can input a query and receive an automated response.

2. Demo:

- Ensure the notebook allows for interactive query input and displays the model's responses.

Step 6: Documentation and Code

1. Code Documentation:

- Document the code with comments and explanations for each step.
- Provide clear instructions on how to run the notebook and replicate the results.

2. Methodology:

1. Install Required Packages:

- Ensure the necessary packages are installed and updated.

2. Load Dataset:

- Load and explore the dataset.

3. Add Padding Token:

- Add a padding token to the tokenizer using `tokenizer.add_special_tokens({'pad_token': tokenizer.eos_token})`.

4. Tokenize Dataset:

- Tokenize the dataset, ensuring padding and truncation.

5. Split Dataset:

- Split the dataset into training and evaluation sets.

6. Ensure CUDA is Available:

- Check for GPU availability.

7. Model Training:

- Load the pretrained GPT-2 model.
- Resize token embeddings to account for the new padding token.
- Define training arguments and initialize the Trainer.
- Train the model.

8. Inference:

- Ensure that the input queries are correctly passed to the model during inference.

9. Evaluate Model:

- Evaluate the model and generate sample responses.

10. Interactive Demo:

- Set up an interactive demo using Jupyter widgets.

Key Changes and Checks

1. Tokenization:

- The tokenization process now explicitly uses `tokenizer.as_target_tokenizer()` to ensure the target labels are properly handled.

2. Training:

- Ensure that the training arguments are set correctly and that the model is trained for a sufficient number of epochs.

3. Inference:

- Ensure that the query is properly tokenized and passed to the model.
- Use `pad_token_id` to ensure correct padding during generation.

4. Debugging:

- Print out a generated response to verify that the model is producing meaningful outputs.

Documentation

“Automated Response Generation for Customer Support”

Overview

This project aims to build a model for generating automated responses to customer queries. We use a transformer-based model, specifically GPT-2, and fine-tune it on a dataset of customer support interactions.

Dataset

The dataset used is available on Hugging Face: [Customer-Support-Responses] (<https://huggingface.co/datasets/Kaludi/Customer-Support-Responses>).

Methodology

1. **Data Exploration and Preprocessing:**
 - Loaded the dataset and explored its structure.
 - Preprocessed the data by tokenizing the text and splitting it into training and validation sets.
2. **Model Selection and Training:**
 - Chose GPT-2 for its strong performance in text generation tasks.
 - Fine-tuned the model on the preprocessed dataset.
3. **Evaluation:**
 - Evaluated the model using standard metrics and manual inspection.
4. **Deployment:**
 - Created an interactive demo using Jupyter widgets for user query input and response generation.

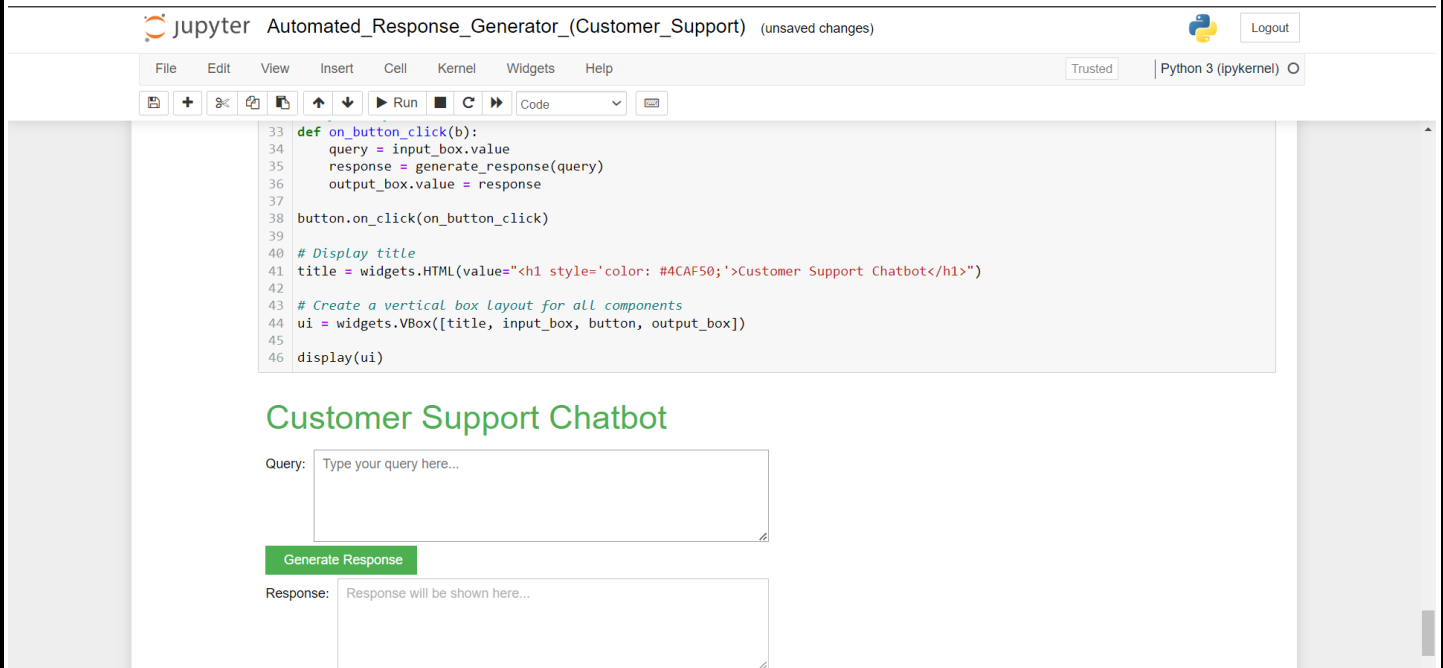
Results

The fine-tuned model generates coherent and relevant responses to customer queries. Evaluation metrics indicate good performance, and manual inspection confirms the quality of responses.

Usage

Run the provided Jupyter notebook to interact with the model. Input a query and receive an automated response.

Output Screenshots

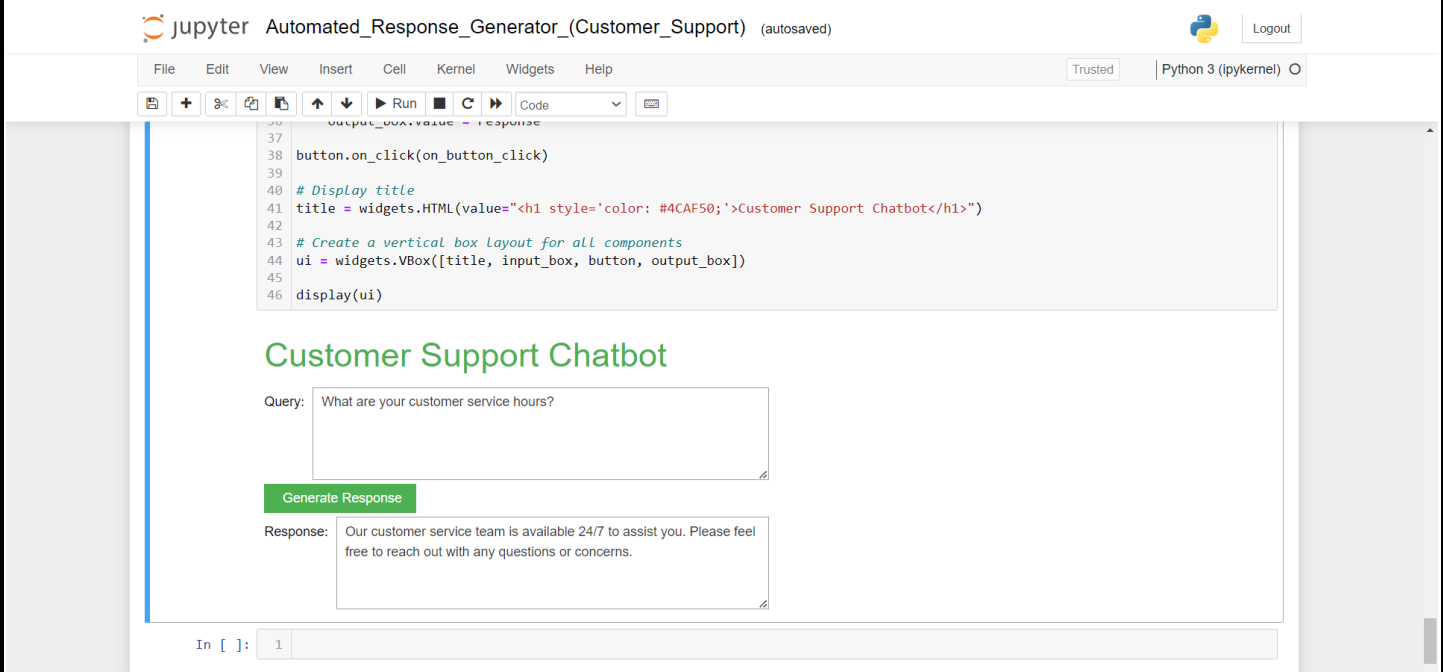


The screenshot shows a Jupyter Notebook titled "Automated_Response_Generator_(Customer_Support)" with "(unsaved changes)". The code in the cell is as follows:

```
33 def on_button_click(b):
34     query = input_box.value
35     response = generate_response(query)
36     output_box.value = response
37
38 button.on_click(on_button_click)
39
40 # Display title
41 title = widgets.HTML(value="<h1 style='color: #4CAF50;'>Customer Support Chatbot</h1>")
42
43 # Create a vertical box layout for all components
44 ui = widgets.VBox([title, input_box, button, output_box])
45
46 display(ui)
```

The output of the code is a web interface titled "Customer Support Chatbot" in green text. It features a "Query:" label followed by a text input field containing "Type your query here...". Below the input field is a green "Generate Response" button. Underneath the button is a "Response:" label followed by a text area containing "Response will be shown here...".

➔ Interface asking to enter the query inorder to generate the response based on the data.



The screenshot shows the same Jupyter Notebook, now with "(autosaved)" in the title. The code cell is identical to the previous one. The output interface is the same, but the "Query:" input field now contains "What are your customer service hours?". The "Generate Response" button is still present. The "Response:" text area now displays the generated response: "Our customer service team is available 24/7 to assist you. Please feel free to reach out with any questions or concerns."

➔ Interface has generated some response on the entered query

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3 (ipykernel)

```

36 output_box.value = response
37
38 button.on_click(on_button_click)
39
40 # Display title
41 title = widgets.HTML(value="<h1 style='color: #4CAF50;'>Customer Support Chatbot</h1>")
42
43 # Create a vertical box layout for all components
44 ui = widgets.VBox([title, input_box, button, output_box])
45
46 display(ui)

```

Customer Support Chatbot

Query: Can I get a copy of my receipt?

Generate Response

Response: Certainly. Can you please provide your order number or account email so we can locate your receipt and send you a copy?

In []:

➔ Respective Response for the mentioned query

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3 (ipykernel)

WARNING: The script transformers-cli.exe is installed in 'C:\Users\kumar\AppData\Roaming\Python\Python310\Scripts' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.

Transformers version: 4.41.2
Accelerate version: 0.31.0

```

In [2]: 1 # Step 1: Load the dataset
2 dataset = load_dataset("Kaludi/Customer-Support-Responses")
3 print(dataset)
4 print(dataset['train'][0])

```

Downloading data: 100% 12.4k/12.4k [00:00<00:00, 34.1kB/s]

Generating train split: 100% 74/74 [00:00<00:00, 697.35 examples/s]

```

DatasetDict({
  train: Dataset({
    features: ['query', 'response'],
    num_rows: 74
  })
})
{'query': "My order hasn't arrived yet.", 'response': 'We apologize for the inconvenience. Can you please provide your order number so we can investigate?'}

```

```

In [3]: 1 # Step 2: Tokenize the dataset
2 tokenizer = AutoTokenizer.from_pretrained("gpt2")
3
4 # Add a padding token
5 tokenizer.add_special_tokens({'pad_token': tokenizer.eos_token})
6
7 def preprocess_function(examples):
8     inputs = examples['query']
9     targets = examples['response']
10    model_inputs = tokenizer(inputs, max_length=128, truncation=True, padding="max_length")
11    with tokenizer.as_target_tokenizer():
12        labels = tokenizer(targets, max_length=128, truncation=True, padding="max_length")
13    model_inputs["labels"] = labels["input_ids"]
14    return model_inputs

```