BUAN 6390.002 - Analytics Practicum

Intermediate Report

GROUP 4

Alka Shrestha

Keshav K Rajbux

Navya Bhat

Sai Varshith Komirisetty

Srinaga Mouneeswar Mallipeddi

Table of Contents

1. Data Preprocessing

1.1. Methodology

Data preprocessing is a crucial step in preparing input data for generative models, such as language models in our case GPT2 Model.
Following are the data preprocessing steps employed for the project:

a. **Data Cleaning:**
   - Removing Noise: Data preprocessing helps in removing noise, irrelevant information, or artifacts from the input data. In text data, this could involve removing special characters, HTML tags, or other non-textual elements.
   - Handling Missing Data: For structured data, missing values may need to be imputed or dropped.

b. **Removing Outliers**: Outliers can adversely affect the training of generative models. Data preprocessing involves the identification and removal of outliers to improve model performance and robustness.

c. **Tokenization** (for Text Data): For natural language processing (NLP) tasks, text data needs to be tokenized, breaking it into words, subwords, or characters. Tokenization is essential for feeding data into language models like GPT, as they operate on tokens rather than raw text.

d. **Reducing Dimensionality**: In cases where the input data has a high dimensionality (e.g., images with many pixels), techniques like dimensionality reduction are applied to reduce the data's dimensionality while preserving important information.

e. **Ensuring Model Compatibility**: Some generative models require specific data formats. Preprocessing ensures that the data is compatible with the chosen model architecture.

1.2. Implementation

1.2.1 Utilizing Named Entity Recognition (NER) for Enhanced Recipe Generation

A distinctive approach has been implemented in the BUAN 6390.002 analytics practicum project to enhance the capabilities of the recipe generation model. The project leverages a Named Entity Recognizer (NER) to systematically identify and extract food-related entities from the dataset. This extracted information will be subsequently integrated as input for the recipe generator, contributing to the creation of recipes that are not only grammatically correct but also contextually relevant and highly specialized. This is essentially done to guide the GPT-2 model in generating recipes that

align with the culinary context. These tokens provide clear instructions to the model, ensuring that the generated recipes are not only linguistically sound but also contextually aligned. The dataset in use contains cooking recipes that have a specific format : a title, a list of ingredients with given amounts, and the instructions in a step-by-step format. The shortest part of the recipe, the title, should accurately name it and summarize its content.

1.2.2 Data Exploration and Structural Correction

1. In the initial phase of the project, the team commenced by identifying duplicated recipes that shared identical URLs.

2. During the exploratory data analysis phase, several issues related to the structural integrity of recipes were identified and addressed. Recipes that lacked both ingredients and instructions were identified as extraction errors and subsequently removed from the dataset. This process was undertaken to ensure the quality and consistency of the dataset used in the project.

3. Recipes in languages other than English were filtered out. The language of each recipe was determined by analyzing its instructions. This approach was chosen to mitigate the presence of foreign names, such as "croissant," which are frequently found in recipe titles and ingredient lists and can potentially lead to misclassification. To achieve this, the project team employed the Google Translate API for the language detection task.

1.2.3 Pre-Processing

1. The dataset is initially explored by inspecting its structure, displaying the first few rows, and scrutinizing the data types of its columns. Subsequently, preprocessing steps are implemented to filter out rows that fail to meet specific criteria. These criteria include excluding rows with very short titles, ingredients, or directions, as well as rows containing instructions with the words "step" or "mix all." The filtered DataFrame is updated by removing the undesired rows.

2. Following the data preprocessing, the DataFrame is divided into training and testing sets using the train_test_split() function from sklearn. The testing set represents 8% of the total dataset. To ensure consistency, the index of both DataFrames is reset.
3. To facilitate further analysis, a custom function named df_to_plaintext_file() is defined. This function iterates through each row of the DataFrame and extracts relevant information, such as the title, directions, ingredients, and NER data. It then transforms this information into a plaintext representation of a recipe and writes it to an output file.

4. To validate the contents of the generated "unsupervised_train.txt" file, the code reads the first two lines and stores them in variables. These lines are displayed, providing a brief glimpse of the data.

```
with open("unsupervised_train.txt", "r", encoding="utf-8") as file:
    # Read the first 2 lines
    first_record = file.readline()
    second_record = file.readline()

# Display the first 2 records
print("First Record:")
print(first_record)

print("\nSecond Record:")
print(second_record)
```

✓ 0.0s

```
First Record:
<RECIPE_ START> <INPUT_START> chicken <NEXT_INPUT> chicken gravy <NEXT_INPUT> cream of mushroom s

Second Record:
<RECIPE_ START> <INPUT_START> cold <NEXT_INPUT> sweet milk <NEXT_INPUT> butter <NEXT_INPUT> eggs
```

Fig 1.2.1

1.2.4 Tokenization

This process is significant because it ensures that the text data is in a format that can be effectively utilized by models like GPT-2. Here are the key aspects of this step:

1. GPT-2 Tokenization: The code begins by creating an instance of the GPT2Tokenizer using the "gpt2" model. This tokenizer is essential for breaking down the text data into smaller units called tokens, which is how models like GPT-2 process language.
2. Special Tokens: Special tokens are set up for various elements in the text, including titles, instructions, ingredients, and markers for the beginning and end of recipes. These tokens are crucial for later stages of the analysis and for maintaining the structure and context of the recipes.
3. Tokenization Update: The tokenizer is updated to include these special tokens, ensuring that it recognizes and processes them correctly during tokenization.
4. HDF5 File Creation: To store the tokenized data efficiently, the code creates an HDF5 file named "unsupervised.h5" using the h5py library.

This format is chosen for its suitability in handling large and structured data.

5. Loop Through Files: The code iterates through two specific filenames, "test" and "train," and opens the corresponding text files for reading. These text files contain the raw text data that needs to be tokenized.
6. Tokenization and Length Check: Within the loop, each line of the text file is read and tokenized using the GPT2Tokenizer. An important check is performed to ensure that the length of the tokenized sequence doesn't exceed 1024 tokens. This check is necessary because GPT-2 models have a maximum input limit and exceeding it would result in data loss.

Fig 1.2.2

```
    num = 0
    rows = 0
    last=[]
    for line in data:
        num+=1
        if num%1000 == 0:
            print("Read "+str(num)+" Written: "+str(rows))

        text_tokens = tokenizer.tokenize(line)
        if len(text_tokens) > 1024: #Recipe won't fit the model
            continue

        text_tokens_ids = tokenizer.convert_tokens_to_ids(text_tokens)

        if (len(last) + len(text_tokens_ids)) <= 1024:
            last+=text_tokens_ids
        else:
            while len(last) < 1024:
                last.append(end_token_id)
            out_np.append(last)
            last=text_tokens_ids
            rows+=1
    out_mat = np.matrix(out_np)
    print(filename,'-',out_mat.shape)
    hf.create_dataset(filename, data=out_mat)
print('done')
hf.close()
```

Fig 1.2.3

1.2.5 Language Model Training:

GPT-2, like other autoregressive models, is currently being trained using CLM loss. Here's how GPT-2 uses this loss function:

- Training Phase: During training, GPT-2 is presented with a text sequence where certain tokens are randomly masked out. The model's objective is to predict the masked tokens based on the context provided by the unmasked tokens.
- Loss Calculation: For each masked token in the sequence, a prediction is made by the model. The model's prediction is compared to the actual token, and a loss is calculated based on the discrepancy between the prediction and the ground truth.
- Backpropagation: The loss is backpropagated through the model's layers, allowing the model to learn and adjust its internal parameters (weights) to improve its performance in predicting masked tokens.

- Objective: GPT-2 aims to minimize the overall CLM loss across its training dataset. This encourages the model to learn the statistical properties of the language, including syntax, semantics, and context, to generate coherent and contextually appropriate text.
- Fine-Tuning**:** GPT-2 can be fine-tuned on specific downstream tasks, such as text generation, translation, or summarization, by minimizing task-specific loss in addition to the CLM loss.

The project is currently being trained using the Hugging Face Transformers library. This model will further be fine-tuned to perform a specific task related to recipe generation.

****