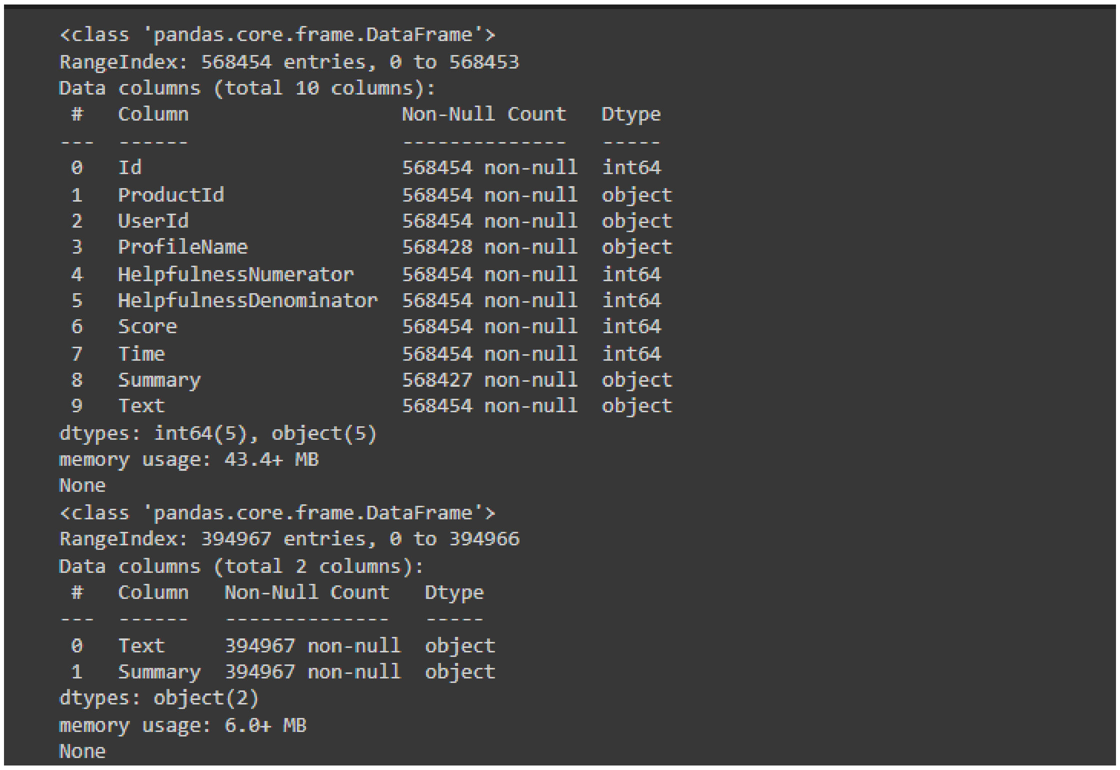
IR Assignment - 4

Medium Used

- Due to the computational demand for this assignment, I have trained and prepared my model using the resources available from Google Collab. I was using tensors. Hence, I set my runtime to T4 GPU, and my dataset was small.
- This assignment was very time-consuming on my part; it took a lot of time to train the models and get the work started.

Dataset Used:

- I used the provided Amazon fine reviews dataset for training and fine-tuning my gpt2 model for text summarization. Here I have shared information about the dataset.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568454 entries, 0 to 568453
Data columns (total 10 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   Id                      568454 non-null  int64
 1   ProductId               568454 non-null  object
 2   UserId                  568454 non-null  object
 3   ProfileName             568428 non-null  object
 4   HelpfulnessNumerator    568454 non-null  int64
 5   HelpfulnessDenominator  568454 non-null  int64
 6   Score                   568454 non-null  int64
 7   Time                    568454 non-null  int64
 8   Summary                 568427 non-null  object
 9   Text                    568454 non-null  object
dtypes: int64(5), object(5)
memory usage: 43.4+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 394967 entries, 0 to 394966
Data columns (total 2 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   Text     394967 non-null  object
 1   Summary  394967 non-null  object
dtypes: object(2)
memory usage: 6.0+ MB
None
```

- For this assignment - we are going to use only the summary and text column of the data frame, which were a result of preprocessing - handling the missing rows, dropping the duplicates and resetting the false indexes.

Preprocessing text and summary column

- I then applied the following preprocessing techniques on my dataste and created two new columns - CleanedSummary and CleanedText
- The Preprocessing Techniques used:
- Lowercasing: Lowercasing text ensures uniformity by treating words with different cases as the same, which helps standardize text data for analysis.

- HTML Tag Removal: Removing HTML tags cleanses text data extracted from web pages or HTML documents, eliminating formatting information irrelevant for analysis.
- Accented Character Removal: Normalizing accented characters to their ASCII equivalents resolves encoding issues and removes non-standard characters for better text processing.
- Acronym Expansion: Expanding common acronyms into full forms improves text readability and understanding, making the text more accessible for analysis or interpretation.
- Special Character Removal: Removing non-alphanumeric characters reduces noise and retains relevant information in text data, enhancing text quality for analysis.
- Stopwords Removal and Lemmatization: Removing stopwords and lemmatizing words to their base forms streamlines text data by eliminating irrelevant words and standardizing vocabulary for more accurate analysis and model training.
- I save them as CleanedReviews.csv so that I do not need to generate them again and again in my program, as they take a lot of time.

Loading the csv file and calculating the average length:

- We utilize pandas to load a CSV file (CleanedReviews.csv) containing cleaned review data into a DataFrame (df), facilitating structured data access and manipulation.
- Device availability is checked using torch.cuda.is_available() to determine if a CUDA-enabled GPU is present, enabling potential acceleration of computations for deep learning tasks.
- After loading the data, missing values are addressed by dropping rows with incomplete or null data (df.dropna()), ensuring data integrity and completeness for subsequent analysis.
- Duplicate rows are identified and removed (df.drop_duplicates()) to maintain data consistency and eliminate redundancy within the dataset.
- The DataFrame index is reset (df.reset_index()) to establish a clean and sequential index post-data manipulations, facilitating efficient data handling and processing.
- We concatenate the CleanedText and CleanedSummary columns into a single training column within the DataFrame (df['training']) to consolidate relevant information for model training and analysis.
- The average length of reviews, calculated by computing the total number of words divided by the number of reviews (avg_length), provides insights into typical review lengths and aids in setting appropriate model input parameters. So that we can assume a max length for the reviews that accommodates most of the reviews, the average was 85.0024 words, so a max length of 100 words was considered sufficient enough.

Initialization of Tokenizer and Custom Dataset Class:

- Tokenizer Initialization: We instantiate a tokenizer using AutoTokenizer.from_pretrained("gpt2") from Hugging Face's transformers library, which allows for text tokenization and encoding specifically tailored for the GPT-2 model.

- Token Encoding: The tokenizer.encode(" TL;DR ") function is used to convert the string " TL;DR " into token IDs, which helps in understanding the token representation of this specific text within the tokenizer's vocabulary.
- Custom Dataset Class (GPT2ReviewDataset): This custom dataset class is designed to preprocess and prepare text data for training or evaluation with a GPT-2 model. It takes inputs such as a tokenizer, reviews, and a maximum length (max_len) for encoding.
- Dataset Initialization: Within the GPT2ReviewDataset class, the __init__ method encodes each review using the provided tokenizer. It adds an end-of-sequence (EOS) token (self.eos) at the end of each review and handles padding or truncation to ensure consistency in sequence length (max_len).
- Padding/Truncation: The pad_truncate method within the dataset class manages the padding or truncation of tokenized sequences based on the specified maximum length (max_len). This ensures that all encoded sequences are of uniform length suitable for model input.
- Data Loading: The custom dataset class enables efficient data loading and batching for training or inference tasks using PyTorch's DataLoader. Each item retrieved from the dataset corresponds to a preprocessed and encoded review ready for model consumption.

Sampling and Division of Reviews into Train and Test Datasets

- Sampling Reviews: We randomly sample 20,000 reviews from the dataset df using df.sample(20000), ensuring diversity in the selected reviews for analysis.
- Resetting Index: After sampling, the index of the sampled DataFrame (reviews_df) is reset to ensure continuous indexing for subsequent operations.
- Dataset Information: We inspect the information of the sampled DataFrame (reviews_df) using reviews_df.info() to understand its structure and properties.
- Preparing Reviews and Summaries:
    - We extract the training column from reviews_df into a list called reviews using .values.tolist(), containing the preprocessed reviews.
    - Initialize a GPT2ReviewDataset instance (reviews_dataset) using the tokenizer and the extracted reviews, specifying the maximum length for encoding.
- Division into Train and Test:
    - Divide reviews_dataset into training (train_reviews) and testing (test_reviews) datasets based on a predefined split (e.g., 15,000 for training and 5,000 for testing).
    - Calculate the lengths of train_reviews and test_reviews using len(train_reviews) and len(test_reviews) to ensure proper division.
- Verification Sampling:
    - Randomly select 5 indices (random_numbers) from a range of 1 to 100 to verify the correctness of the division and encoding process.
    - For each selected index, extract and decode a review from both train_reviews and test_reviews to inspect their content.
    - Also retrieve the corresponding summaries (train_summaries and test_summaries) to ensure alignment with the reviews.

Dataloader and Model Training and Choosing hyperparameters:

- Selected a BATCH_SIZE of 32 to balance memory usage and training efficiency while loading and processing batches of reviews for the GPT-2 model.
- Defined EPOCHS as 3 to determine the number of complete iterations over the dataset during training, achieving a balance between training time and model convergence.
- Set LEARNING_RATE to 3e-4 to control the step size of optimization updates using the AdamW optimizer, influencing the model's learning dynamics.
- Initialized a pre-trained GPT-2 model (AutoModelWithLMHead) from Hugging Face's transformers library, leveraging transfer learning to fine-tune the model on the specific review dataset.
- Moved the model to the available computing device (device) to harness GPU acceleration for faster training and inference.
- Chose the AdamW optimizer (optim.AdamW) to optimize model parameters, with the specified learning rate (LEARNING_RATE), managing gradient updates efficiently during training.
- Developed a training function that iterated over the dataset for the defined number of epochs (EPOCHS), processing batches from the dataloader (train_dataloader) on the designated device.
- Within each epoch, reset model gradients, applied updates based on computed loss to progressively improve model performance.
- Explored different values for BATCH_SIZE, EPOCHS, and LEARNING_RATE to optimize training dynamics, influencing both training speed and final model quality.
- Adjusted hyperparameters to fine-tune model performance and achieve optimal text generation results.

Saved and Loaded Model Parameters:

- Saved the trained PyTorch model's state dictionary to a specified file path (model_path) using torch.save(model.state_dict(), model_path), ensuring preservation of learned parameters and architecture.
- Displayed a message confirming the successful saving of the model at the specified file path (model_path).
- Loaded a previously saved model from a specified file path (model_path) using torch.load(model_path, map_location=torch.device(device)), enabling retrieval of saved model parameters.
- Utilized AutoTokenizer and AutoModelWithLMHead to initialize an instance of the model, ensuring compatibility between the loaded model and its original architecture.
- Printed a confirmation message indicating the successful loading of the model from the specified file path (model_path).

Top-k Sampling Function:

- Defined a topk function that performs top-k sampling using PyTorch:
- Softmaxes the input probabilities (probs) along the last dimension to convert them into a probability distribution.
- Uses torch.topk to retrieve the top n probabilities and their corresponding indices (topIx).

- Normalizes the selected probabilities (tokensProb) and converts them to a numpy array for random selection.
- Makes a random choice (choice) from the top probabilities based on the normalized distribution.
- Returns the chosen token ID (tokenId) as an integer.

Model Inference Function (model_infer)

- Accepts a trained model, tokenizer, review text, and optional max_length.
- Encodes the review text using the tokenizer.
- Initializes the sequence (result) with the encoded review.
- Prepares the initial input tensor for the model by converting the encoded review into a tensor.
- Iteratively generates tokens from the model:
    - Retrieves logits from the model's output for the last token in the current sequence.
    - Utilizes the topk function to select the next token ID based on the model's predictions.
    - Appends the selected token ID to the result sequence.
    - Terminates the loop if the EOS token is predicted, decoding the generated sequence.
- Returns the decoded result after the loop or when reaching the specified max_length.

Prediction and Comparison:

- Predicted summaries are generated for each review in test_reviews using the trained model and tokenizer.
- Each predicted summary is extracted from the result of model_infer by splitting on " TL;DR " and selecting the second part.
- The number of test summaries (test_summaries) and predicted summaries (predicted_summaries) are printed to verify their lengths.
- An evaluation dictionary (eval_dict) is created using the test summaries and predicted summaries.
- This dictionary is used to construct a DataFrame (eval_df).
- Initial information about the DataFrame (eval_df.info()) is printed to inspect its structure.
- Rows with empty or None values in either the test or predicted summaries are dropped from eval_df.
- Iterating over the DataFrame rows using iterrows(), valid test and predicted summaries are filtered and collected into separate lists (test_summaries and predicted_summaries).
- The eval_df DataFrame is updated with the filtered lists of summaries.
- After filtering, the DataFrame information (eval_df.info()) is displayed again to verify the updated structure.
- ROUGE scores can then be calculated using the filtered test_summaries and predicted_summaries to assess the model's performance on the test set.

To finish off, I tested the rouge scores on preprocessed data and unpreprocessed data - the test summarization system works better in case of the preprocessed data.

Github Link:

https://github.com/tonythomasndm/CSE508_Winter2024_A4_2021360.git
Drive Link:

https://drive.google.com/drive/folders/1e8_5Rt_WQZQLBQvVsJ7hOGyXj29Uxaxm?usp=drive_li
nk