Text Summarization for Long Documents

Hossain Md Noor

Tokyo International University

Python for Data Science & AI

Prof. Hamza Rafik

December 11, 2024

Abstract

In a competitive world, people are always on a constant run and look for the most effective and efficient ways to do their tasks. This is where text summarization comes in handy. Text summarization is a way in which long articles or documents are shortened so that the users can focus on the main key points/highlights. It not only helps the users to save time but also reduces the effort by retaining the most important points and information from the text, making it simpler and quicker for the user to comprehend the content. In this project, we will summarize text using a pre-trained model from “Hugging Face” called “bart-large-cnn”. We will also select the data from “Hugging Face” called “cnn_dailymail”. Throughout the project, we will call out certain Python libraries and codes, and carry out certain steps to clean the data like preprocessing, and later we will generate the summaries of the articles from the data using the pre-trained model (without fine-tuning). Lastly, we will generate similarity scores using RapidFuzz and compare the generated summaries with the summaries given in the dataset to see how reliable or accurate the generated summaries are in comparison to the original summaries.

Keywords: text summarization, long articles, python, pre-trained model, dataset

Introduction

Text summarization is the process by which we condense large texts into shorter summaries while keeping the main idea or information of the text. Using the model, which has been antecedently trained on various large datasets, we carry out several steps to generate summaries efficiently. It is a challenge faced in natural language processing which seeks to transform lengthy documents into short summaries that encapsulate the content of the original document. This “game-changing” tech has changed how we absorb information, allowing us to understand the key ideas in complex material quickly. Text summarization algorithms help save time and effort by automatically extracting relevant information which allows for better processing of large amounts of text. In the case of long-form content, text summarization tends to be particularly useful. Whether this takes the form of a long research paper or a significant report, these types of algorithms are usually capable of condensing out the main points into “easy-to-understand” portions. This enables readers to understand the fundamental ideas without reading the entire document.

About the Dataset

The dataset that we have chosen for this project is “abisse/cnn_dailymail”, which is collected from the Hugging Face. The CNN/DailyMail is a dataset which consists of more than 300k unique articles that have been written on articles from CNN and the Daily Mail. In Hugging Face, there are three subsets, which are 1.0.0, 2.0.0, 3.0.0, and three splits, which are train, validation, and test. For this specific project, we are working with the “train” split of the subset “3.0.0”.

About the Pre-trained Model

The model that we have selected for this project is “facebook/bart-large-cnn”, which is also collected from the Hugging Face. Bart is a sequence-to-sequence model that adopted a

bidirectional (like BERT) encoder coupled with an autoregressive decoder. There are two ways in which BART is pre-trained: 1) it corrupts the data with an arbitrary noisy function and 2) it allows learning a model to reconstruct the corrupted or distorted original text. BART proved to be very efficient when fine-tuned for text generation (e.g. summarization, translation) but also for comprehension tasks (text classification, question answering). This specific checkpoint is fine-tuned on CNN Daily Mail, one of the largest collections of text-summary pairs.

Initial Section of the Coding

Initially, we installed the datasets using the "!pip" command, and by doing so we can get access to wide range of datasets that are used in machine learning. Then we imported necessary libraries, such as pandas, numpy, and load_dataset. Afterward, when we loaded the dataset and checked the contents it carries, we could see that "train" split, which we aimed to work with, had 3 columns and 287,113 number of rows. As all the articles in the datasets are exceedingly long, it would not be wise to work with all the rows as it would take a considerable duration to perform the operations. Therefore, we took a sample of the first 100 rows and worked on it throughout our project.

Importing the Pre-trained Model

In this section, we first installed the transformers library, which is crucial when engaging with a pre-trained model like BART. Using the transformers library, we set up a summarization library which we later used for generating summaries (we got the code snippet from Hugging Face).

Preprocessing the Sample

At the outset, we made sure that our sample is tokenized properly. Using "AutoTokenizer", it chose the most suitable tokenizer for the model without manual intervention. We used the process of tokenization because it becomes easier for the model to process the tokenized format and also increases the efficiency as well. Later, we preprocessed the sample

by lowercasing the texts, removing punctuation, removing extra whitespaces, and trimming any leading/trailing whitespaces. By preprocessing, our goal was to clean the data and omit unnecessary characters before processing the sample through the summarization model. Through this approach, it becomes easier for the model to pay attention to important words and contexts in the dataset, making it more compatible. This also could feasibly increase accuracy for text summarization.

Truncating the Sample

Before truncating, we have tried to generate the summaries of "preprocessed_sample_ds" using the summarization pipeline. However, it resulted in an error, showing "index is out of range itself", as BART's maximum length itself is 1024. Thus, we truncate (shorten) the preprocessed sample to a maximum length of 1024, to make sure that the BART model is capable of managing the input sequences within its limits.

Generating Summaries

This is the part where we carry out our main objective: generating the summaries with the help of the summarization pipeline. We iterated through all the articles in the preprocessed sample to enhance efficiency by automating recurring operations. Iteration also helps the pre-trained model (like truncation) by breaking the input data into smaller pieces so that the model can understand better. We named the list as "all_summaries" where we store all the generated summaries, and also used "tqdm" from "tqdm" library to display progress bars which we can inspect throughout the operations. However, using "tqdm" does not impact or benefit our operations, as its task is simply to show us the completion of each article generated, enabling convenience for us to track the summarization process. We use "max_length=130" to set the maximum length for each generated summary to 130 tokens, and "min_length=30" to set it to a minimum size of 30 tokens just like it is shown in the Hugging Face example for the pre-trained model. The purpose of putting "do_sample=False" in the code snippet is to disable sampling, which would lead to generating inconsistent summaries as the model would select the random

tokens throughout the operations. Then we run the snippet and generated the summaries of the preprocessed sample.

Initially, we took a sample of 1000 rows to work with. However, as we progressed and reached the “generating summaries” section, we noticed that it was taking a long time to generate each summary as each article is long. Thus, to increase time efficiency and reduce complexity, we intended to select the first 100 rows/articles of the datasets, which roughly took 30 minutes to generate all the summaries.

Evaluation Metrics

To evaluate the generated summaries with the original highlights, we used “rapidfuzz” library. “Rapidfuzz” is a powerful tool for Python that can be used to generate scores based on text strings matching between two data samples. We first found out all the similarity scores between each original highlight and its corresponding generated summary using the “fuzz.ratio” function. Since we need to find a value to determine the accuracy of the model for summarizing this dataset, we calculated the mean similarity score, which came out to be 51.145. As the average similarity score was low, we thought that there might have been some important characters that we mistakenly eliminated when we preprocessed the sample. Therefore, we later carried out the same operations to generate the highlights for the sample which was not preprocessed (only truncated) to assess the similarity score it would generate.

Without Preprocessing

As the previous mean similarity score was low (51.145), at this juncture we tried to generate summaries without preprocessing the sample. We used the same code snippet which we used previously. The only changes we made were to the assigned names (for example: “all_summaries2”, “truncated_sample_ds2”, etc.) to reduce the chance of mixing with the assigned names used while preprocessing in the previous section. This time, the similarity score we got was 42.56, which was lower than what we got after preprocessing. Thus, as the similarity

score for the preprocessed sample was better, we could say that preprocessing indeed played a significant role in generating summaries here.

Additionally, we also generated similarity scores using “fuzz.token_set_ratio” function. This function is quite similar to “fuzz.ratio” function, but instead, it focuses less on word order and string variations. For the preprocessed sample the mean score was 58.80, and for the sample that was not preprocessed, the score turned out to be 46.01. Therefore, we could conclude that preprocessing gives a better similarity score between the generated summaries and the original summaries based on the respective results.

Human Evaluation

Since the tested similarity scores were low, we conducted a human evaluation of the first 10 summaries. By reading each preprocessed summary, these are our results (setting our criterion to cohesiveness and relevance, between 0 (very poor) to 5 (being excellent):

1st summary – 3

2nd summary – 4

3rd summary – 5

4th summary – 4

5th summary – 4

6th summary – 3

7th summary – 3

8th summary – 3

9th summary – 5

10th summary – 5

Thus, the mean human evaluation score is 3.9 out of 5. It should be noted that this was a rough human evaluation to verify whether the generated summaries were coherent and relevant or not.

Conclusion

At the end of the project, we could conclude that preprocessing the sample gives better similarity scores. However, even after preprocessing, the mean similarity score was 51.15, which is a low score. Thus, we conducted a rough human evaluation and observed that almost all the summaries in the first 10 rows were coherent and meaningful. In conclusion, it is possible to state that generated summaries (using the BART model) may not be similar to the original summaries but can be consistent and relevant to its respective corresponding articles.

References

Hugging Face. (2024). *Hugging Face* [AI Platform]. Hugging Face. https://huggingface.co/

OpenAI. (2024). *ChatGPT* (GPT-4) [AI model]. OpenAI. https://openai.com/

Google. (2024). *Gemini* [AI model]. Google. https://gemini.google.com/app