

NEWS ARTICLE SUMMARIZATION

Raghavendra Reddy Paluri

Yogishwar Reddy Karumulla

Abstract

The news article summarization project leverages the power of natural language processing to automatically generate concise summaries of news articles. By analyzing the text of news articles, the algorithm can identify the most relevant and important information and present it in condensed form. This project aims to improve the efficiency of news consumption and provide readers with quick access to the key points of a story. With the increasing amount of news articles being published every day, this technology can help individuals stay informed without the need to read through lengthy articles.

Introduction

In today's fast-paced world, keeping up with the latest news can be a challenge. With so many news sources and articles available, it can be difficult to

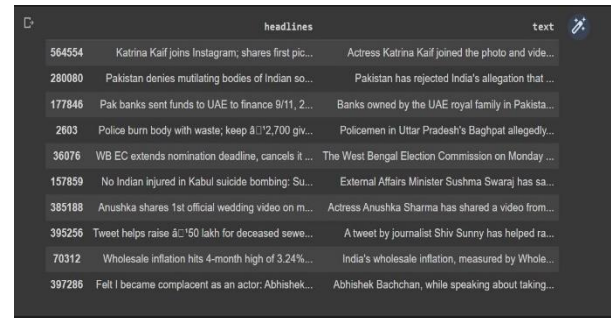
find the time to read everything. However, staying informed is crucial, and missing important news can have serious consequences. The news article summarization project aims to address this issue by developing an algorithm that uses natural language processing techniques to automatically generate concise summaries of news articles. This project has the potential to revolutionize the way we consume news, making it faster and more efficient to stay informed about the events shaping our world. Traditional news outlets have long relied on headlines and summaries to attract readers' attention and provide a quick overview of a story. However, these summaries are often brief and can leave out important details. By using natural language processing, the news article summarization project can provide readers with a more comprehensive summary of a news

article, highlighting the most important points and providing a condensed version of the story. This technology has the potential to benefit individuals who want to stay informed but may not have the time or desire to read every news article in its entirety. Additionally, this technology can be useful for researchers and professionals who need to keep up with the latest news in their field.

Dataset

A dataset is a collection of data that is organized in a specific way, usually for the purpose of analysis, modeling, or machine learning. In the context of a news article summarization project, a dataset would typically consist of many news articles and corresponding summaries, which would be used to train a machine learning model to generate summaries automatically. We created a custom dataset by downloading information from various sources like Wikipedia, google, and some famous news article websites. It

consists of 600k rows in the train dataset and 11k rows in the test dataset.



The image shows a screenshot of a dataset table with two columns: 'headlines' and 'text'. The table contains 10 rows of data, each with a unique ID, a headline, and a corresponding text snippet. The headlines are truncated, and the text snippets are also truncated. The table is displayed in a dark-themed interface.

	headlines	text
564554	Katrina Kaif joins Instagram, shares first pic...	Actress Katrina Kaif joined the photo and vide...
280080	Pakistan denies mutilating bodies of Indian so...	Pakistan has rejected India's allegation that ...
177846	Pak banks sent funds to UAE to finance 9/11, 2...	Banks owned by the UAE royal family in Pakista...
2603	Police burn body with waste, keep it ₹2,700 giv...	Policemen in Uttar Pradesh's Bagpat allegedly...
36976	WB EC extends nomination deadline, cancels it ...	The West Bengal Election Commission on Monday ...
157859	No Indian injured in Kabul suicide bombing: Su...	External Affairs Minister Sushma Swaraj has sa...
385188	Anushka shares 1st official wedding video on m...	Actress Anushka Sharma has shared a video from...
395256	Tweet helps raise ₹150 lakh for deceased sewe...	A tweet by journalist Shiv Sunny has helped ra...
70312	Wholesale inflation hits 4-month high of 3.24%...	India's wholesale inflation, measured by Whole...
397286	Felt I became complacent as an actor: Abhishek...	Abhishek Bachchan, while speaking about taking...

TOOLS

- NLTK.
- Python 3.
- Google Colab.
- Transformers.
- BART/T-5.

Methodology

The primary objective of this project is to create text summarization models capable of generating summaries for one or multiple documents at a time, utilizing deep learning techniques. Throughout the project, we will guide you through the process of implementing these techniques, step-by-step. We will rely on pre-trained

models to generate summaries, and we will explore various methods of fine-tuning these models to optimize their performance for our specific use case. By the end of this project, you will have a strong understanding of the underlying concepts behind text summarization, as well as the tools and techniques necessary to develop effective text summarization models. We defined several functions and imports necessary libraries for implementing a text summarization model using the T5 transformer architecture from the Hugging Face transformers library. First, the necessary libraries are imported including PyTorch, NumPy, and Pandas. The T5 tokenizer and T5ForConditionalGeneration module are also imported from the transformers library. Next, the code defines a function for displaying a Pandas Data Frame in ASCII format using the rich library's Table and Column classes. This function will be used later to display sample data.

A rich console logger is also defined to display training status during model training. The logger uses the rich library's Table and Column classes to create a table that displays the current epoch, number of training steps, and training loss. This logger will be used to monitor the progress of the model during training. We defined a custom dataset class named datasetclass that can be used to read the dataset and load it into a pytorch dataloader for finetuning the T5 model.

The class constructor takes in five arguments: dataframe: the input dataframe containing the source and target texts.

tokenizer: the T5 tokenizer that will be used to encode the text data.

source_len: the maximum length of the source text to be encoded.

target_len: the maximum length of the target text to be encoded.

source_text: the name of the column in the input dataframe containing the

source texts.target_text': the name of the column in the input dataframe containing the target texts.

The class has two methods: `__len__()` and `__getitem__()`.

The `__len__()` method returns the length of the target_text list which is equal to the total number of examples in the dataset.

The `__getitem__()` method is used to get an item at a specific index in the dataset. It does the following:

- Extracts the source and target texts at the given index from the dataframe.
- Cleans the text data to ensure it is in the string format.
- Encodes the source and target texts using the T5 tokenizer with a max_length and padding to the maximum length specified. It returns the input_ids and attention_mask tensors for both source and target texts.
- Returns a dictionary containing the input_ids tensor and attention_mask tensor for the source text, input_ids tensor and attention_mask tensor for the target text (target_ids), and

input_ids tensor for the target text shifted by one position (target_ids_y) for teacher forcing.

The source_ids tensor is of size [source_len], source_mask tensor is of size [source_len], target_ids tensor is of size [target_len], and target_ids_y tensor is of size [target_len]. The dtype of all tensors is torch.long.

We defined a function for training a deep learning model for text summarization and the function does the following operations:

1. Sets the model to training mode using `model.train()`.
2. Loops through the loader data, which is a dataloader that iterates over batches of data from the dataset.
3. Gets the target_ids, source_ids, and source_mask from the batch of data.
4. y_ids is created by removing the last token from target_ids (as this is used for prediction).
5. lm_labels is created by

copying `target_ids` and setting the values of the last token to -100. This is because we do not want the model to predict the pad tokens (which are assigned the value of 0) during training.

6. The model is called with `input_ids`, `attention_mask`, `decoder_input_ids`, and `labels` set to `ids`, `mask`, `y_ids`, and `lm_labels`, respectively.

7. The loss is extracted from the `'outputs'` and backpropagation is performed to update the model's parameters.

8. The `training_logger` logs the epoch number, batch number, and loss every 10 batches to keep track of the training progress.

The `validate` function which is used for evaluating the model's performance on the validation set. The function takes as input the epoch number, tokenizer, model, device, and validation dataloader. Firstly, the model is put in

evaluation mode using `model.eval()`. Then, two empty lists are created for storing the predicted summaries and the actual summaries. After that, for each batch in the validation dataloader, the input sequence `ids`, attention mask, and target sequence `ids` are extracted from the batch and moved to the specified device using `to(device, dtype=torch.long)`.

Next, the `generate` method of the model is called to generate a summary from the input sequence. The `generate` method generates summaries using the model's encoder-decoder architecture. The generated summary is returned as a tensor of token `ids` which is then converted to a text summary using the `'decode'` method of the tokenizer.

Finally, the predicted summary and the actual summary are appended to their respective lists. After iterating through all the batches in the dataloader, the function returns the list of predicted summaries and the list of actual summaries. We also defined a function

T5Trainer that trains a T5 transformer model for a summarization task. The input arguments to the function are a pandas dataframe containing the raw dataset, the name of the source text column, the name of the target text column, model parameters, and an output directory for saving the trained model, generated predictions, and logs.

The function first sets random seeds for PyTorch and NumPy for reproducibility. It then loads the T5 tokenizer and the T5ForConditionalGeneration model and moves the model to the device (GPU/TPU) specified in the model_params dictionary. The raw dataset is read from the input pandas dataframe, and the train-validation split is performed. The function then creates training and validation datasets and dataloaders using a custom dataset class DataSetClass and PyTorch's DataLoader. The optimizer for tuning the weights of the network is defined, and the training loop is initiated for the

specified number of epochs. In each epoch, the 'train' function is called with the necessary arguments to train the model. After training, the trained model and tokenizer are saved to the output directory.

The validation loop is then initiated for the specified number of validation epochs, and in each epoch, the validate function is called with the necessary arguments to evaluate the model on the validation dataset. The generated text and the actual text are collected for each sample in the validation dataset, and they are saved to a CSV file in the output directory. Finally, the function saves the logs to a text file in the output directory and prints messages indicating the locations of the saved model, generated predictions, and logs.

Result

The results that we obtained after the validation are:

For ROUGE-1, the 'r', 'p', and 'f' values are 0.148, 0.075, and 0.1, respectively.

This means that 14.8% of the unigram tokens in the generated summary match the reference summary, while the precision is only 7.5%, indicating that the generated summary contains many extraneous words. The F1 score is the harmonic mean of precision and recall, and it is 0.1.

For ROUGE-2, the values are 0.0 for 'r', 'p', and 'f', indicating that there are no bigrams in the generated summary that match the reference summary.

For ROUGE-L, the 'r', 'p', and 'f' values are 0.148, 0.075, and 0.1, respectively. This is like ROUGE-1, but ROUGE-L uses the longest common subsequence between the generated and reference summaries, rather than just the n-gram overlap.

```
rouge.get_scores(test_prediction_df['Generated Text'][0], test_prediction_df['Actual Text'][1], avg=True)
```

```
{'rouge-1': {'r': 0.14814814814814814,  
  'p': 0.07547169811320754,  
  'f': 0.0999999955281252},  
 'rouge-2': {'r': 0.0, 'p': 0.0, 'f': 0.0},  
 'rouge-l': {'r': 0.14814814814814814,  
  'p': 0.07547169811320754,  
  'f': 0.0999999955281252}}
```

The result shows the performance of the model based on three ROUGE metrics: ROUGE-1, ROUGE-2, and ROUGE-L. The values of 'r', 'p', and 'f' are shown for each metric.

ROUGE-1 computes the overlap of unigrams (single words) between the generated summary and reference summary.

ROUGE-2 computes the overlap of bigrams (pairs of adjacent words) between the generated summary and reference summary.

ROUGE-L computes the longest common subsequence between the generated summary and reference summary, which takes into account word order and sentence structure. The values of 'r', 'p', and 'f' represent the recall, precision, and F1 score respectively. In this specific example, the model achieved an ROUGE-1 F1 score of 0.318, an ROUGE-2 F1 score of 0.126, and an ROUGE-L F1 score of 0.297. These scores indicate that the model generated summaries that had some overlap with the reference

summaries, but there is still room for improvement.

```
rouge.get_scores(test_prediction_df['Generated Text'], test_prediction_df['Actual Text'], avg=True)
```

```
{'rouge-1': {'r': 0.36721941765773264,  
  'p': 0.2901837708210135,  
  'f': 0.3180016754881537},  
 'rouge-2': {'r': 0.1504423803429087,  
  'p': 0.11270453535112972,  
  'f': 0.1256281764976631},  
 'rouge-l': {'r': 0.3434297714078125,  
  'p': 0.2714640668299395,  
  'f': 0.2974391853705805}}
```

References

<https://towardsdatascience.com/t5-a-model-that-explores-the-limits-of-transfer-learning-fb29844890b7>

<https://towardsdatascience.com/text-summarization-with-nlp-texttrank-vs-seq2seq-vs-bart-474943efeb09>

<https://github.com/huggingface/transformers>

<https://arxiv.org/abs/1910.10683>

https://huggingface.co/docs/transformers/model_doc/bart