Environmental News NLP Dataset Summarization

Project Report (Milestones 1, 2, and 3)

Noor Magdy (16005059)   Mohamed Wael (19011272)

Contents

1 Milestone 1: Data Understanding and Exploration

1.1 Introduction and Motivation

Natural Language Processing (NLP) tasks heavily depend on the quality and characteristics of the input data. Before applying any summarization techniques, it is essential to understand the structure, content, and limitations of the dataset being used. The goal of this project is to explore text summarization techniques on broadcast news data, evaluating both custom-built neural networks and state-of-the-art pre-trained transformer models.

The motivation behind this work stems from the increasing volume of televised and digital news. Providing accurate, concise summaries of environmental news allows policymakers, researchers, and the general public to quickly grasp essential information without parsing through lengthy transcripts. Through basic data inspection, cleaning, exploratory data analysis, and subsequent modeling, this project establishes a solid foundation for comparing extractive and abstractive summarization methods.

1.2 Literature Review

Text summarization is a fundamental problem in Natural Language Processing, broadly categorized into two main approaches: extractive and abstractive summarization. Extractive summarization involves identifying and extracting the most salient sentences or phrases directly from the source text and concatenating them to form a summary. Traditional methods relied on statistical and graph-based algorithms, such as Term Frequency-Inverse Document Frequency (TF-IDF) and TextRank. These algorithms score sentences based on word frequencies or graph centrality. While extractive methods guarantee grammatical correctness (as they copy human-written text), they often suffer from poor coherence and an inability to condense information naturally.

The advent of Deep Learning introduced abstractive summarization, which aims to generate novel sentences that capture the core meaning of the source text, much like a human would. Early neural approaches relied on Sequence-to-Sequence (Seq2Seq) architectures powered by Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks. A standard Seq2Seq model utilizes an Encoder to compress the input text into a fixed-length context vector, and a Decoder to generate the summary token by token. However, fixed-length vectors struggle to retain information from long documents.

To resolve the bottleneck of fixed-length context vectors, the Attention Mechanism was introduced. Attention allows the decoder to dynamically ”focus” on relevant parts of the input sequence during each step of the generation process. Additive attention and multiplicative attention significantly improved the fluency and accuracy of RNN-based abstractive summarizers.

Despite the success of RNNs with attention, they suffer from sequential processing constraints, making training slow and limiting their ability to model very long-range dependencies. This led to the paradigm shift brought by the Transformer architecture, which relies entirely on self-attention mechanisms and allows for massive parallelization.

Transformers paved the way for large pre-trained language models. For summarization, encoder-decoder transformers like BART (Bidirectional and Auto-Regressive Transformers) and T5 (Text-to-Text Transfer Transformer) represent the current state-of-the-art. BART is pre-trained by corrupting text with an arbitrary noising function and learning a model to reconstruct the original text. This denoising objective makes BART exceptionally powerful for text generation and abstractive summarization tasks. In this project, we traverse this historical progression by first building an LSTM-based Seq2Seq model from scratch, followed by fine-tuning the pre-trained BART architecture.

1.3 Data Analysis, Insights, and Limitations

The dataset used in this project is the *Environmental News NLP Dataset*, obtained from Kaggle. It consists of short news snippets extracted from television news programs (CNN, MSNBC, BBC News, FOX News) related to environmental and public policy topics.

Data Overview & Insights: The dataset was downloaded into the Google Colab environment and combined from 418 separate CSV files, resulting in a single DataFrame of 94,858 records. Basic exploratory data analysis revealed the following:

- **Snippet Length Distribution:** The dataset contains snippets with an average length of approximately 41 words. The distribution is heavily concentrated between 30 and 50 words. The shortest snippet contains 2 words, while the longest reaches 194 words.

- **Program Consistency:** Boxplot analysis indicates that despite originating from different TV programs, the median snippet length remains consistent at around 40 words across all shows. Outliers are present in shows like BBC News and Hardball, indicating occasional extended commentary.

- **Temporal Stability:** A time-series analysis of average snippet lengths from 2009 to 2020 shows stable trends with only minor fluctuations, suggesting a standardized editorial style in broadcast news over the decade.

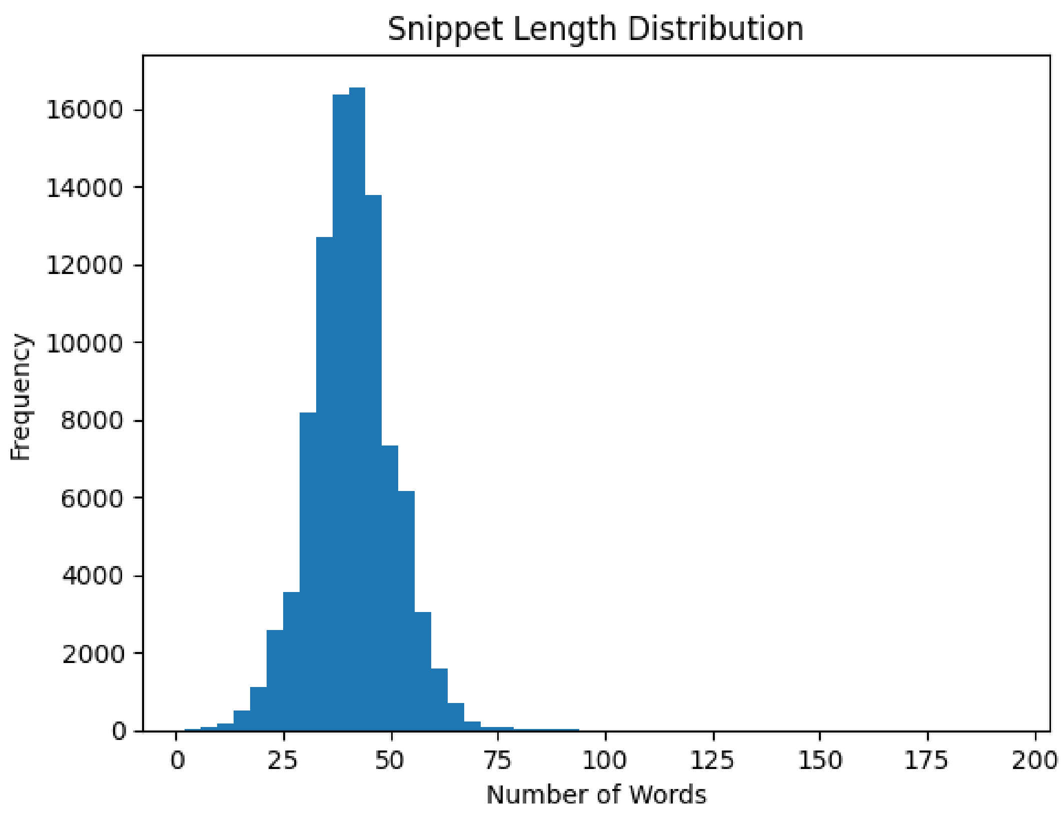

Figure 1: Exploratory Data Analysis showing snippet length distribution and program consistency.

Limitations: The primary limitation of this dataset is the absence of ground-truth reference summaries (labels). Because the texts are already relatively short (avg. 41 words), the potential for strong compression is limited. Furthermore, training supervised neural networks requires target labels. To overcome this in subsequent milestones, we utilize a frequency-based extractive heuristic to generate ”silver standard” labels for training our abstractive models.

2 Milestone 2: Neural Network Model from Scratch

2.1 Methodology

2.1.1 Silver Label Generation and Preprocessing

Because the dataset only contains raw news snippets without corresponding target summaries, training a supervised neural network required the creation of pseudo-targets. We utilized a frequency-based extractive summarization algorithm to isolate the most important sentence in each snippet based on word frequency scores. This extracted sentence acts as our "Silver Label" ground truth.

Text preprocessing involved lowercasing all text, stripping special characters, and limiting multiple spaces. For the target summaries, we appended `sostok` (Start of Sequence) and `eostok` (End of Sequence) tokens to guide the decoder during sequence generation.

2.1.2 Model Architecture

In accordance with the project requirements, we built a Sequence-to-Sequence (Seq2Seq) neural network from scratch without relying on pre-trained embeddings or architectures.

- **Encoder:** The encoder processes the input snippet. It consists of a randomly initialized Embedding layer (dimension = 100, Vocabulary = 10,000) followed by three stacked Long Short-Term Memory (LSTM) layers. Each LSTM layer utilizes 256 latent units and a dropout rate of 0.4 to prevent overfitting.
- **Decoder:** The decoder generates the summary one word at a time. It features its own Embedding layer (dimension = 100) and a single LSTM layer (256 units), initialized with the final hidden and cell states of the Encoder.
- **Attention Mechanism:** To allow the model to focus on salient words in the source text, we integrated an `AdditiveAttention` layer. The attention context vector is concatenated with the decoder LSTM output.
- **Output Layer:** A `TimeDistributed` Dense layer with a Softmax activation function maps the decoder outputs to the 10,000-word vocabulary, outputting the probability distribution of the next word.

The entire architecture contains approximately 8.9 million trainable parameters.

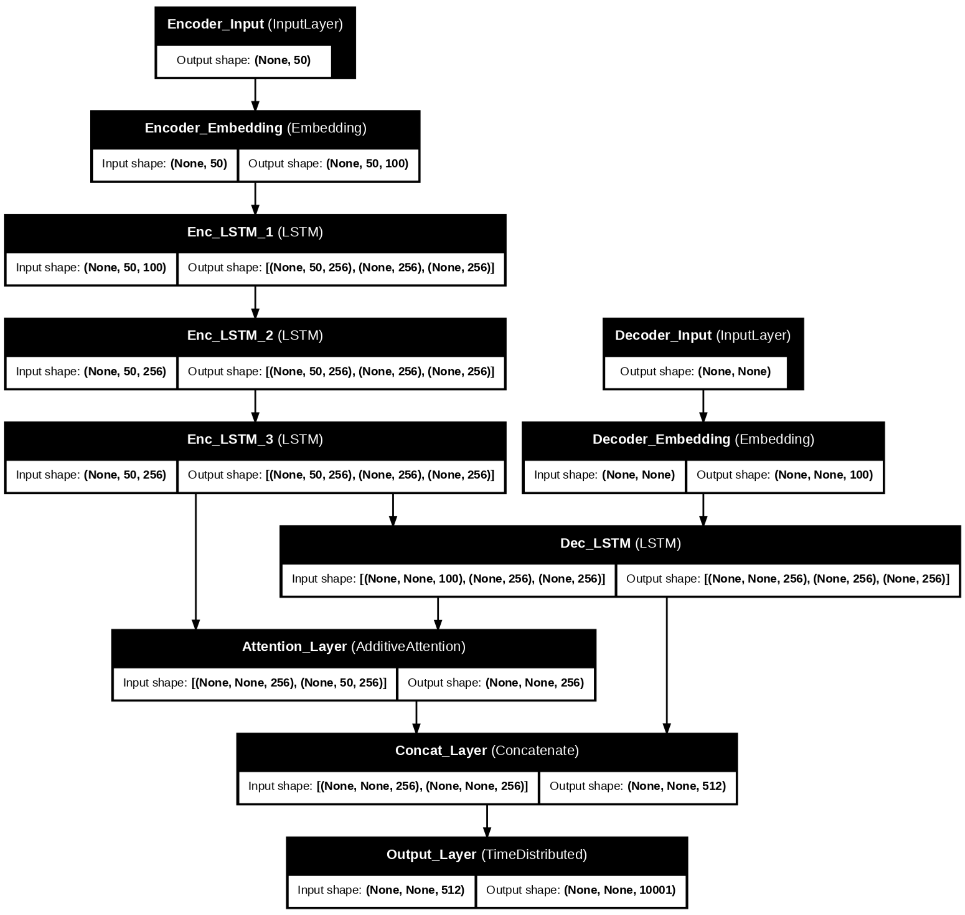
Encoder_Input (InputLayer)
Output shape: (None, 50)
Encoder_Embedding (Embedding)
Input shape: (None, 50)
Output shape: (None, 50, 100)
Enc_LSTM_1 (LSTM)
Input shape: (None, 50, 100)
Output shape: [(None, 50, 256), (None, 256), (None, 256)]
Enc_LSTM_2 (LSTM)
Input shape: (None, 50, 256)
Output shape: [(None, 50, 256), (None, 256), (None, 256)]
Decoder_Input (InputLayer)
Output shape: (None, None)
Enc_LSTM_3 (LSTM)
Input shape: (None, 50, 256)
Output shape: [(None, 50, 256), (None, 256), (None, 256)]
Decoder_Embedding (Embedding)
Input shape: (None, None)
Output shape: (None, None, 100)
Dec_LSTM (LSTM)
Input shape: [(None, None, 100), (None, 256), (None, 256)]
Output shape: [(None, None, 256), (None, 256), (None, 256)]
Attention_Layer (AdditiveAttention)
Input shape: [(None, None, 256), (None, 50, 256)]
Output shape: (None, None, 256)
Concat_Layer (Concatenate)
Input shape: [(None, None, 256), (None, None, 256)]
Output shape: (None, None, 512)
Output_Layer (TimeDistributed)
Input shape: (None, None, 512)
Output shape: (None, None, 10001)

Figure 2: Architecture of the custom LSTM Sequence-to-Sequence model with Attention.

2.2 Training and Evaluation Results

The network was compiled using the RMSprop optimizer and the sparse categorical crossentropy loss function. The dataset was split into an 80/20 train-validation ratio. The inputs were padded to a maximum length of 50 tokens, and targets to 20 tokens. We trained the model in mini-batches of 60 over a maximum of 40 epochs.

An `EarlyStopping` callback was implemented (monitoring validation loss with a patience of 2) to prevent the high-capacity model from memorizing the training data. Training halted at Epoch 39 as validation loss ceased to improve, reaching a final validation accuracy of $\sim 92.14\%$ and a validation loss of 0.4657.

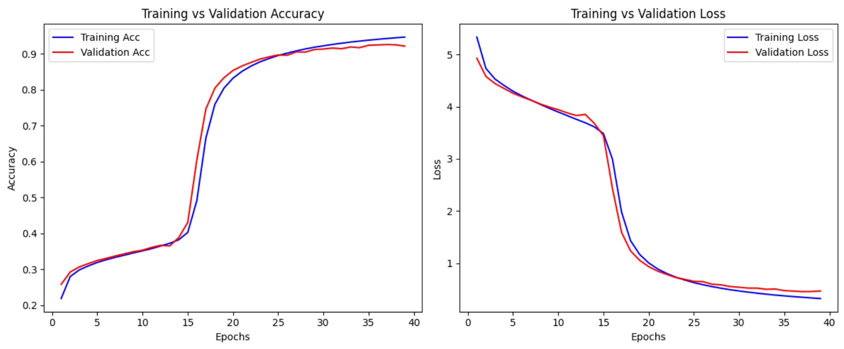

Figure 3: Training and validation loss and accuracy curves over the training epochs.

Inference and Generation: Because Seq2Seq models generate text autoregressively, we built a separate inference model. The model encodes the input, and the decoder loop uses the generated output at time step t as the input for step $t+1$ until the `eostok` token is produced.

While the evaluation for this milestone is strictly based on the successful implementation of the architecture, we calculated baseline ROUGE metrics on 1,000 unseen validation samples to benchmark against Milestone 3. The LSTM achieved a **ROUGE-1 of 0.5058**, **ROUGE-2 of 0.4232**, and **ROUGE-L of 0.4819**.

3 Milestone 3: Pre-Trained Model and Fine-Tuning

3.1 Updated Methodology

In this milestone, we replaced our scratch-built LSTM model with a state-of-the-art pre-trained transformer: **BART** (Bidirectional and Auto-Regressive Transformers), specifically the `facebook/bart-large-cnn` checkpoint provided by Hugging Face.

Data Preparation for Fine-Tuning: BART uses its own specific sub-word tokenizer and handles special sequence tokens internally. Therefore, we removed the manual `sostok` and `eostok` tags from our silver labels. The input texts were tokenized and padded to a maximum length of 512, and the target summaries to a maximum length of 60.

Fine-Tuning Architecture: We utilized the `Seq2SeqTrainer` API from the Hugging Face `transformers` library. The training was optimized using mixed precision (fp16) on a GPU to accelerate training and manage VRAM. We configured the training with a learning rate of $3e^{-5}$, a batch size of 4, and gradient accumulation steps of 4. We incorporated an `EarlyStoppingCallback` to prevent overfitting.

3.2 Training and Evaluation Results

The model was fine-tuned on a subset of the data over 10 epochs. The early stopping callback successfully halted training at Epoch 5, registering a minimal validation loss of 0.0782.

To satisfy the requirement of evaluating the system on natural text unseen during the training phase, we isolated a test split of 1,000 samples. Using the `evaluate` library, we calculated ROUGE scores for the fine-tuned BART model and compared them directly to our custom LSTM from Milestone 2.

Metric	Milestone 2 (LSTM)	Milestone 3 (BART)	Improvement
ROUGE-1	0.5058	0.5908	+0.0850
ROUGE-2	0.4232	0.5748	+0.1516
ROUGE-L	0.4819	0.5876	+0.1057

Table 1: Performance Comparison on Unseen Validation Data (1,000 samples)

3.3 Findings and Conclusion

The results distinctly highlight the superiority of transfer learning in NLP tasks. The fine-tuned BART model significantly outperformed the custom LSTM Seq2Seq model across all metrics. The most notable jump was in the ROUGE-2 score (+0.1516), which measures bigram overlap. This indicates that the pre-trained transformer generates significantly more fluent and grammatically sound summaries compared to the LSTM, which had to learn language syntax entirely from scratch on a relatively small dataset.

Abstractive vs. Extractive Characteristics: A deeper visual analysis of the generated outputs revealed that the LSTM model often struggled to form coherent, novel sentences. Conversely, BART leveraged its vast pre-trained knowledge to generate true abstractive summaries. For example, instead of merely copying the first sentence of a news snippet (as our baseline algorithm did), BART successfully paraphrased and combined concepts from different parts of the text.

Compression Ratio: We also analyzed the compression ratio (Summary Length / Original Snippet Length). The baseline labels had a compression ratio of ~ 0.47. The BART model generated summaries with a ratio of ~ 0.74. While the summaries were slightly longer, they were far more comprehensive, successfully retaining the core narrative of the environmental news snippets without losing context.

In conclusion, this project validates that while building sequence models from scratch is excellent for understanding network architectures and attention mechanisms, leveraging pre-trained transformer models via fine-tuning yields vastly superior, production-ready abstractive text summaries.