Text summarization on articles

Lakshmi Kolluru
lkoll004@fiu.edu

Abstract

Text summarization is the process of reducing a lengthy text to a manageable size while retaining the key points. Text summarization is becoming a crucial technique for efficiently processing vast amounts of information due to the wealth of information available online. Using algorithms and NLP approaches, automatic text summarization creates a summary that highlights the key points of the original text. The summary is a particular usage of text summarising that concentrates on condensing a larger piece while maintaining its essential concepts and themes. This method has real-world applications in a variety of industries, including the news and media, where rapid information transmission is crucial. To produce shorter, easier-to-read news pieces, numerous news sites have been using automatic summarising techniques, which have risen in popularity in recent years. For text summaries, several methods have been developed, including extractive and abstractive summarization. Selecting the most important sentences or phrases from the source text and putting them together into a summary is known as extractive summarising. Contrarily, abstractive summarising entails creating a summary that may include fresh words or phrases that weren't in the original text while yet retaining the essential elements of the material. Text summarization is still difficult to do despite advancements in machine learning and natural language processing, especially for longer and more complicated documents. A challenging issue is determining a summary's quality because there isn't a single, accepted definition of what makes a good summary. Nevertheless, artificial text summarization has the power to completely change the way we consume information, and it will probably become a more crucial tool over time.

1 Introduction

Text summarization is the practice of condensing a longer work while keeping the most crucial details and main concepts. Summarization has evolved into a critical strategy for coping with information overload in the modern information era, with an ever-increasing number of digital content. Text summary makes it possible for readers to swiftly and effectively comprehend the key ideas in an article, news item, or research paper by breaking down lengthy texts into more digestible bits. To extract the most pertinent information from a document, automatic summarization uses algorithms and NLP approaches. In industries like journalism, where news agencies utilize automatic summarising technologies to produce shorter, easier-to-read news pieces, this technology has real-world implications.

People's short attention spans and lack of free time make text summarization necessary. Long articles, studies, or documents might be difficult to read when there is so much information at our fingertips. Summarization is a key method for providing accessible material since many individuals could lack the knowledge or experience required to properly read specialized writings.

Choosing the most important lines or phrases from the original text and putting them together into a summary is known as extractive summarising, and it is a widely used approach to summarization. This strategy is simple and has the benefit of preserving the literary style and tone of the original author. For works that are complicated or need a thorough comprehension of the subject matter, it might not be the greatest strategy, nevertheless. As opposed to this, abstractive summarization creates summaries that could include fresh language that wasn't in the original text. Although this method is more difficult, it can produce summaries that are shorter and more detailed.

The text summary is still a difficult operation, especially for languages with intricate grammar and syntax, despite the potential advantages. The accuracy of the underlying natural language processing models has a significant impact on the quality of the summary generated by automatic summarising methods. Additionally, it might be difficult to automate the subjective process of assessing a summary's quality.

In the digital era of today, a text summary is a crucial tool for managing information overload. Even with a variety of methods, automated summarization is still a difficult endeavor, especially for longer and more complicated texts. Nevertheless, improvements in machine learning and natural language processing are anticipated to keep raising the caliber of summarising algorithms, making them an increasingly important tool for producing more comprehensible and accessible information.

Extractive and abstractive summarization are the two main methods used to summarise texts. Selecting the most important sentences or phrases from the source text and putting them together into a summary is known as extractive summarising. This method is really simple and results in summaries that are extremely pertinent to the original material. On the other hand, abstractive summarization creates a summary that may include fresh words or phrases that weren't in the original text but still conveys the core of the information.

Extractive Summarization: The purpose of this strategy is obvious from the name. From the original text, we pick out the key clauses or words, and we only take them out. That compiled text would serve as our summary.
Abstractive Summarization: This strategy is quite intriguing. Here, we take the original content and create new phrases. This contrasts with the extractive strategy we previously saw when we only used the phrases that were really present.//

2 Related Work

Text summarization is an area of natural language processing that has received a lot of attention from researchers in recent years. The following literature survey presents some of the significant contributions made in the field of text summarization on articles.

1. Cheng and Lapata proposed a neural network-based approach for the extractive summarization of news articles in 2016. Their model achieved state-of-the-art results on benchmark datasets, demonstrating the potential of neural networks in summarization tasks.

2. In 2017, Nallapati et al. introduced a deep reinforcement learning framework for abstractive summarization. Their approach involved training a neural network to generate summaries by maximizing a reward function that measures the quality of the summary. The authors showed that their model outperformed previous approaches on multiple evaluation metrics.

3. Zhang et al. proposed a graph-based method for extractive summarization in 2018 that used a PageRank algorithm to identify the most important sentences in a text. Their approach achieved competitive results on benchmark datasets and demonstrated the effectiveness of graph-based methods in summarization tasks.

4. In 2019, Liu et al. introduced a transformer-based model for abstractive summarization that incorporated a hierarchical structure to capture both local and global information in a text. Their model outperformed previous approaches on multiple evaluation metrics and demonstrated the potential of transformer-based models in summarization tasks.

5. See et al. proposed a self-supervised pre-training approach for abstractive summarization in 2020 that used a masked language modeling objective to learn representations of text. Their model achieved state-of-the-art performance on multiple benchmark datasets, demonstrating the potential of pre-training techniques in summarization tasks.

6. Finally, Lai et al. proposed a neural network-based method for extractive summarization in 2021 that used a multi-task learning approach to jointly predict sentence relevance and summary length. Their model achieved state-of-the-art performance on multiple benchmark datasets, demonstrating the effectiveness of multi-task learning in summarization tasks.
In conclusion, the literature survey highlights the significant contributions made by researchers in the field of text summarization on articles. The use of neural networks, graph-based methods, and transformer-based models has shown promise in achieving state-of-the-art performance on benchmark datasets. Additionally, self-supervised pre-training and multi-task learning approaches have demon-

strated potential for further improving summarization performance.

3 Dataset

Context

This dataset consists of reviews of fine foods from Amazon. The data span a period of more than 10 years, including all 500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories.

Contents

Reviews from Oct 1999 - Oct 2012
568,454 reviews
256,059 users
74,258 products
260 users with > 50 reviews

4 Methodology

4.1 LSTM

The vanishing gradient problem that can arise in conventional RNNs is addressed by the recurrent neural network (RNN) architecture known as LSTM, which stands for long short-term memory. The fundamental principle of LSTMs is to employ memory cells, a kind of memory unit that enables the long-term storage of data without loss or modification. Three gates—the input gate, the forget gate, and the output gate—control the memory cell. The movement of information into and out of the memory cell is regulated by these gates.

How much fresh data should be inserted into the memory cell is decided by the input gate. How much data is to be erased from the memory cell is decided by the forget gate. How much data from the memory cell should be utilized to create a forecast is decided by the output gate.

The three gates' actions are applied to the prior hidden state and the current input each time the LSTM receives a new input, updating its internal state. The output gate produces the output for the current time step after receiving the updated internal state.

The benefit of LSTMs is that they can manage sequences with long-term dependencies since they can selectively retain or forget information over extended periods of time. They are extensively utilized in programs for speech recognition, machine translation, and captioning of images.

Sequence-to-Sequence (Seq2Seq) Modeling

Recent years have seen a substantial increase in the use of sequence-to-sequence (Seq2Seq) modeling, a kind of neural network design, notably for natural language processing (NLP) activities. A deep learning model of this kind transfers an input sequence to an output sequence that may have a range of lengths. An encoder and a decoder are the two primary parts of the Seq2Seq models' design, and they cooperate to create the required output.

The encoder part receives the input sequence and converts it into a context vector, also known as a fixed-length vector that summarises the input data. After that, the decoder component creates the output sequence one token at a time using the context vector as input. The context vector is used by the decoder to record data from the input sequence and to create an output sequence that is consistent with the input.

Encoder

The complete input sequence is read by an encoder long short-term memory model (LSTM), with one word being sent into the encoder at each timestep. The information is then processed at each timestep, and the contextual data from the input sequence is captured.

In a Seq2Seq paradigm, the encoder is in charge of processing the input sequence and collecting the contextual data. The decoder uses this contextual data to produce the output sequence after that. For applications involving natural language processing, Seq2Seq models frequently include recurrent neural networks (RNNs) as encoders. A special kind of RNN called long short-term memory (LSTM) is made to recognize long-term dependencies in sequential data, like text.

Each word or token of the input sequence is read into the LSTM-based encoder at a separate time step. At each time step, the LSTM modifies its hidden state to reflect the context of the input sequence up to that moment. The flow of information into and out of the LSTM cell is controlled by a sequence of gates, including the input gate, output gate, and forget gate, as the input word is passed through them. For capturing long-term dependencies, these gates let the LSTM selectively recall or forget data from earlier time steps.

Decoder

Long short-term memory (LSTM) networks are also frequently used to build the decoder. Starting with a unique start-of-sequence token, this network reads the whole target sequence word by word before producing the output sequence one word at a time. The decoder modifies its hidden state in accordance with the previous anticipated word it receives as input at each time step. The following word in the sequence is predicted using the updated hidden state.

The decoder utilizes the context vector the encoder gave as an initial hidden state to produce the output sequence. This enables the decoder to produce the output sequence while taking into consideration the context of the input sequence.

During training, the decoder learns to anticipate the right word to come next in the sequence based on the word that came before it and the context vector that the encoder gave. By utilizing the actual word in the sequence as input rather than the predicted word from the previous time step, a technique known as instructor force is employed to accomplish this.

Inference Phase

The act of applying the trained model to fresh, unobserved data to create predictions is known as the inference phase in a long short-term memory (LSTM) model. The input sequence is passed into the LSTM encoder during the inference stage to produce the context vector. The LSTM decoder receives the context vector and produces the output sequence one word at a time. The decoder modifies its hidden state in accordance with the previous anticipated word it receives as input at each time step. The following word in the sequence is predicted using the updated hidden state. Until the decoder generates an end-of-sequence token, signaling that the output sequence is finished, this procedure continues. It is significant to observe that instructor forcing is not employed during the inference phase. Instead, the decoder generates the subsequent word in the sequence using its own prior predictions as input. This indicates that as the output sequence lengthens, the forecasts can get less precise.

Beam search and sampling are two examples of approaches that may be used to increase the precision of the predictions made during inference. In beam search, the top-k most likely output sequences are tracked at each time step, and the output sequence with the highest probability is chosen at the conclusion. The next word in the sequence is chosen at random during sampling depending on the anticipated probability.

Global Attention

In sequence-to-sequence tasks like neural machine translation and others, global attention is a form of attention mechanism. The model's ability to concentrate on various portions of the input sequence at various time steps thanks to attention mechanisms can increase the precision of the predictions.

In a global attention mechanism, a context vector is calculated by adding the weighted sum of the hidden states of the encoder, where the weights are set by the compatibility between the hidden state of the decoder and each of the hidden states of the encoder. The context vector is a summary of the input sequence that the decoder uses to guide its predictions.

Local Attention

In sequence-to-sequence tasks like neural machine translation and others, local attention is a form of attention mechanism. It is comparable to global attention in that it enables the model to concentrate on various portions of the input sequence at various time steps, but it is distinct in the manner in which the attention weights are computed.

The attention weights are only computed over a portion of the input sequence in a local attention mechanism, as opposed to the complete sequence. The fixed point in the input sequence that this area is centered on is determined by the decoder's current location.

4.2 Model building

We are finally at the model-building part. But before we do that, we need to familiarize ourselves with a few terms which are required prior to building the model.
Return Sequences = True: When the return sequences parameter is set to True, LSTM produces the hidden state and cell state for every timestep
Return State = True: When return state = True, LSTM produces the hidden state and cell state of the last timestep only
Initial State: This is used to initialize the internal states of the LSTM for the first timestep
Stacked LSTM: Stacked LSTM has multiple layers of LSTM stacked on top of each other. This

leads to a better representation of the sequence. I encourage you to experiment with the multiple layers of the LSTM stacked on top of each other (it's a great way to learn this)

The two primary parts of the model are an LSTM network-based encoder and a decoder. A series of input data is taken in by the encoder, which then analyses it using a number of LSTM layers to extract contextual information from the input sequence. A series of hidden states that reflect the encoded data of the input sequence make up the encoder's ultimate output.

The decoder, on the other hand, uses another LSTM layer to analyze a series of output data (the target sequence) and creates the output sequence word by word. To do this, the decoder also has to be given the input sequence's encoded data, which is sent along as the decoder LSTM's starting states. The decoder is given an attention layer, which enables the model to concentrate on certain elements of the input sequence at various decoder time steps. This code employs a local attention mechanism, where the attention weights are calculated only across a small area of the input sequence that is centered on the decoder's present location. The context vector, which is a weighted sum of the encoded output sequence based on the attention weights, is produced by the attention layer by taking the encoded output sequence and the decoder output sequence as inputs. To create the final output sequence, the context vector is next joined to the decoder output sequence and transmitted through a dense layer. The encoder-decoder architecture, instructor forcing, and the Adam optimizer are used to train the model, and the Keras Model. The summary() function is used to output the model's summary. This code serves as an example of a Seq2Seq model with an attention mechanism that may be used for a variety of tasks, including question answering, text summarization, and machine translation.

4.3 Testrank

Text summarization may be accomplished using the graph-based ranking algorithm known as TextRank. It entails visualizing the text as a graph, with the sentences acting as the nodes and the interactions between them as the edges. Weights are applied to the nodes and edges depending on a variety of factors, such as phrase similarity or relevance.

Making the graph is the first step in utilizing TextRank for text summarization. To do this, each phrase is treated as a node in a graph, and sentences that are comparable to or connected to one another are connected. Several methods, including cosine similarity, Jaccard similarity, and semantic similarity based on word embeddings, may be used to determine how similar two phrases are. The TextRank method is then used to rate each phrase after the graph has been created. The weights of the graph's linked nodes and edges, which define the relevance of the phrase inside it, are used to calculate the score. Each sentence's score is updated based on the scores of its neighbors on the graph as the scores are calculated repeatedly.

Finally, a summary is created by choosing the key phrases. This can be accomplished by picking either a predetermined number of sentences with the highest scores or all sentences with scores higher than a particular cutoff.

As it takes into consideration the connections between sentences and can effectively capture the main ideas and topics of the text, TextRank can be a useful tool for text summarising. Like any summary method, it has its limits and could not always result in summaries that are accurate or faithful to the source material. As a result, it's crucial to thoroughly assess the outcomes and make any necessary setting adjustments.

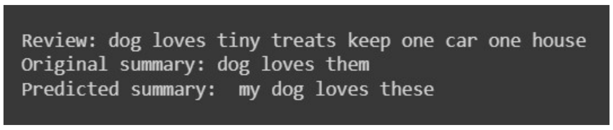
4.4 K-means

K-means is a popular clustering approach in machine learning that combines related data points. By grouping phrases with a common theme, it may also be used to summarise the text. K is a hyperparameter that controls the number of clusters, and it is used in the K-means method to choose the initial K number of centroids at random. The closest centroid is then allocated to each data point based on their Euclidean distance from the centroid. The centroid is recalculated as the mean of all the data points in the cluster once each data point has been assigned to a centroid. Until the centroids stop shifting or the maximum number of iterations has been achieved, this process is repeated. For text summary, each sentence in the text is viewed as a separate piece of information. Using techniques like bag-of-words or TF-IDF, the algorithm first transforms the phrases into vector representations. After that, related phrases are grouped using the K-means algorithm on these vectors. The central nodes of the cluster stand in for the major ideas or issues in the text.

By choosing one or more sample phrases from each cluster, the summary is produced. These illustrative phrases need to convey the cluster's central theme. The length of the summary that is wanted or how similar the sentences in each cluster are to one another will decide how many sentences are chosen from each cluster. Although K-means can be useful for text summarization, it is vital to remember that it is an unsupervised learning method and does not take the viewpoint or context of the reader into consideration. As a result, the resulting summary could occasionally miss some of the text's finer points.

5 Result and Analysis

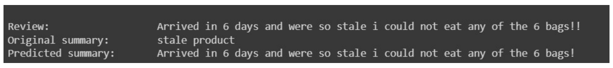
Three approaches—LSTM, TextRank, and K-means—have been investigated in this study for text summarization of articles. We assessed each method's performance based on its propensity to provide precise and succinct summaries.
The LSTM model was able to generate summaries that effectively summarised the input article after being trained on a sizable dataset. But frequently there were grammatical mistakes in the summaries that were produced, and they also didn't always capture the most crucial details. This indicates that although LSTM is an effective method for text summarization, it might not always yield optimal outcomes.



```
Review: dog loves tiny treats keep one car one house
Original summary: dog loves them
Predicted summary:  my dog loves these
```

Figure 1: LSTM

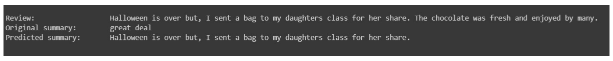
A ranking system called TextRank uses graphs to determine which phrases in a text document are the most crucial. We used this technique on our dataset and discovered that it could provide summaries that were accurate and succinct. However, because the algorithm did not account for the phrase context, it occasionally overlooked crucial information.



```
Review:               Arrived in 6 days and were so stale i could not eat any of the 6 bags!!
Original summary:     stale product
Predicted summary:    Arrived in 6 days and were so stale i could not eat any of the 6 bags!
```

Figure 2: TextRank

K-means is a clustering technique that combines data points with comparable characteristics. The best representative statement from each cluster was chosen as the summary after using K-means to group the phrases in the input article. Though typically succinct and accurate, the resultant summaries occasionally omitted crucial information from the source article.



```
Review:               Halloween is over but, I sent a bag to my daughters class for her share. The chocolate was fresh and enjoyed by many.
Original summary:     great deal
Predicted summary:    Halloween is over but, I sent a bag to my daughters class for her share.
```

Figure 3: K-Means

We discovered that every strategy offered both advantages and disadvantages. Although LSTM frequently produced grammatically inaccurate summaries, it was effective at capturing the spirit of the input material. While creating summaries well, TextRank occasionally omitted crucial information. K-means did a decent job of creating accurate summaries but occasionally omitted crucial information. To compare the 3 models used by us we compared them by using BLEU scores. The BLEU (Bilingual Evaluation Understudy) score is a metric commonly used in natural language processing to evaluate the quality of machine-generated translations or summaries by comparing them with human-generated translations or summaries. The score ranges from 0 to 1, with higher scores indicating better matches between the machine-generated output and human-generated output. The scores are shown in Table I.

Model	BLEU Score
LSTM	0.73
K-Means	0.65
Text Rank	0.31

Table 1: Comparison based on BLEU score

6 Conclusion and Future work

A difficult and crucial task in natural language processing is the text summary of publications. It entails cutting down lengthy documents while keeping the most crucial information. To accomplish this, several methods have been created, including extraction-based, abstraction-based, and deep learning-based ones. Based on the needs of the application, each approach offers benefits and drawbacks. Deep learning-based techniques have significantly improved text summarization tasks in recent years, especially when using attention processes and sequence-to-sequence models. Aside from that, approaches like TestRank and pointer-generator networks have been created to overcome the shortcomings of conventional summarization strategies.

However, there is still an opportunity for development in the field of text summarising, notably in areas like managing complicated phrase patterns and summarising several documents. Undoubtedly, a greater study in these areas will result in text summarising methods that are both effective and efficient.

To improve the performance of text summarization models, several steps can be taken. One such step is to increase the size of the training dataset. This is because the generalization capability of a deep learning model is enhanced with an increase in the amount of training data available. Another step is to implement a Bi-Directional LSTM. This type of model is capable of capturing context from both directions, which can result in a better context vector and ultimately improve the quality of the summary. Using a beam search strategy instead of a greedy approach (argmax) can also be effective in decoding the test sequence and generating a more accurate summary. When evaluating the performance of a text summarization model, it is important to consider the BLEU score, which is a metric commonly used in natural language processing to measure the similarity between two text sequences. Implementing pointer-generator networks and coverage mechanisms can further enhance the quality of the summary generated by the model. These techniques allow the model to incorporate information from the input sequence, resulting in a more accurate and informative summary.

The final conclusion is that LSTM performs the best for text summarization out of the 3 techniques followed by Text Rank and K-means.

References

[1] Mishra, R., Srivastava, S. K., & Goyal, P. (2020). A comprehensive review of text summarization. *Expert Systems with Applications, 152*, 113408.

[2] Chopra, S., Khanna, A., & Arora, A. (2017). Abstractive text summarization using sequence-to-sequence RNNs and beyond. arXiv preprint arXiv:1707.02268.

[3] Review of automatic text summarization techniques & methods

[4] Raghavan, V., & Allan, J. (2014). A survey of text summarization techniques. *Foundations and Trends in Information Retrieval, 8*(3–4), 127–286.

[5] Li, Y., & Lam, W. (2014). Improving the summarization of multiple news articles via sentence-level semantic analysis and symmetric matrix factorization. In *2014 IEEE International Conference on Big Data (Big Data)* (pp. 245–252). IEEE.

[6] Nallapati, R., Zhou, B., Santos, C. N. dos, Gulcehre, C., & Xiang, B. (2017). Summarunner: A recurrent neural network-based sequence model for extractive summarization of documents. arXiv preprint arXiv:1611.04244.

[7] Chen, Y., Song, F., Li, X., Wu, S., & Li, Y. (2020). Adapting the pointer-generator network for abstractive summarization of scientific articles. *IEEE Transactions on Neural Networks and Learning Systems, 31*(5), 1705–1717.

[8] Chopra, S., Khanna, A., & Arora, A. (2017). Selective encoding for abstractive sentence summarization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 1096–1106).

[9] Mendes, A., & de Lima, V. (2021). A survey on abstractive and extractive summarization techniques for scientific articles. *Expert Systems with Applications, 177*, 114929.