Automatic Text Summarization

Alejandro CASTRO ROS, Salma ELGHOURBAL, Eric Xavier N'GUESSAN

Master IASD, April 2021

Contents

1 Introduction

In this document we will present the results we have obtained for the task of text summarization as part of our course in Natural Language Processing. The Python code used during the project is publicly available in a GitHub repository[1] specifically created for that purpose.

1.1 Text summarization in NLP

Summarization is the task of condensing a piece of text to a shorter version, reducing the size of the initial text while at the same time preserving key informational elements and the meaning of content. Since manual text summarization is a time expensive and generally laborious task, the automation of the task is gaining increasing popularity. There are two main types of text summarization tasks: **extractive text summarization** and **abstractive text summarization**. The first one consists of identifying the significant sentences of the text and creating a summary with them. The abstractive text summarization is a bit more complex; the goal is to identify the important sections, interpret the context and reproduce the meaning in a new and coherent way. This ensures that the core information is conveyed through shortest text possible. Thus, it's not restricted to simply selecting and rearranging passages from the original text. Even though abstractive summarization is a more challenging task, there have been many important advances so far, specially thanks to recent developments in the deep learning area [5].

2 Performance metrics

Since we are going to present different methods for text summarization, it's important to define a common way of evaluating their performance so we can make a fair comparison between them.

In order to evaluate the performance of summarization and machine translation, a quite common metric used in NLP is the so-called ROUGE metric (Recall-Oriented Understudy for Gisting Evaluation). Numerous variants of ROUGE are implemented in the Python package *rouge*[2], and we will use them for evaluating our results. We follow the work done by Koupaee and Wang in the original Wikihow paper [1] and use the following metrics: ROUGE-1, ROUGE-2 and ROUGE-L.

ROUGE-1 refers to the overlap on a unigram level between the reference summary and the predicted summary. ROUGE-2 refers to the overlap on a bigram level between the two summaries. Finally, ROUGE-L measures the longest common subsequence (LCS) between the prediction and the reference summary; i.e., it analyzes the longest sequence of tokens that is shared between both. Hence, this metric takes into account sentence level structure similarity in a natural way.

Another metric that we used for evaluating our models is METEOR (Metric for Evaluation of Translation with Explicit ORdering), which is originally an automatic metric for machine translation evaluation and that was introduced by Satanjeev Banerjee and Alon Lavie in 2005 [8]. This metric is based on a generalized concept of unigram matching between the prediction and human-produced ground truth. In this way, unigrams can be matched based on their surface forms, stemmed forms, and meanings. Furthermore, METEOR can be easily extended to include more advanced matching strategies. It was designed to fix some of the problems found in the more popular BLEU metric that was introduced by Papineni et al in 2002 [9]. The advantage of METEOR is that it yields a good correlation with human judgement at the sentence or segment level. This differs from the BLEU metric in that BLEU seeks correlation at the corpus level.

[1] `https://github.com/XavierNg1/NLP_Project`
[2] `https://pypi.org/project/rouge/`

3 Data

The dataset we'll work with is WikiHow. This dataset was first introduced by Mahnaz Koupaee and William Yang Wang in the paper *WikiHow: A Large Scale Text Summarization Dataset* [1]. It is a dataset of more than 230,000 article and summary pairs extracted and constructed from an online knowledge database written by different human authors. It contains articles about various topics written in different styles.

Each paragraph of an article starts with a sentence summarizing it. The summary of the article was obtained by merging those sentences. In each article, an author explains the steps to do some task. In Figure 1 we can see an example extracted from this dataset.

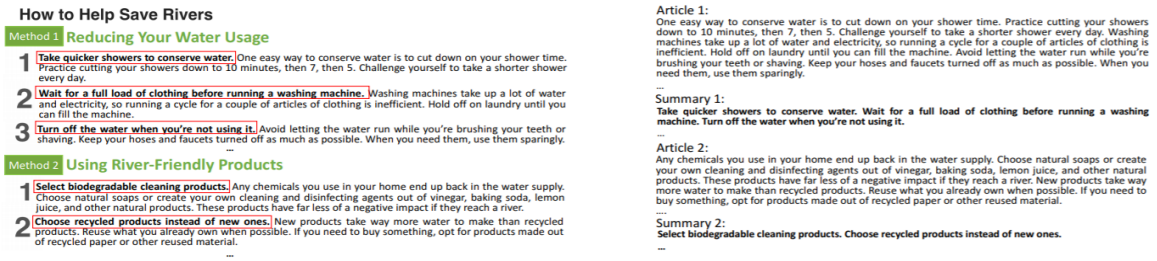

Figure 1: Picture taken from [1] with the framed sentences form the summary and the remaining parts form the article.

3.1 Data preprocessing

The Data preprocessing step is very important in many tasks in NLP, including text summarization. Depending on the model that we want to implement, different preprocessing tasks need to be done. For instance, for a TextRank type model that we will discuss later, we remove punctuation, numbers and special characters from the sentences of each article because they don't affect the summarization task and usually give better performance. We also lowercased all the words and removed stop words in order to reduce the vocabulary size. We often need to vectorize the text before applying any model. In TextRank model, we can use a Word2Vec [3] or a Glove [4] word embedding model in order to represent words as vectors and then sentences as a mean of the word embeddings of the words they contain.

The vectorization step is different for a sequence to sequence model based on RNNs or LSTMs. Firstly, we study the statistical distributions of the articles and the summaries. This will allow us to fix the maximum length of each one of them. In order to illustrate that, we found that in the training set of Wikihow, 97.87% of the summaries have length below 100. Hence, we were able to fix a maximum length for the summary ($max\_summary\_len = 100$). We also fixed a maximum length for the articles ($max\_text\_len = 500$). One important step is adding START and END special tokens to the beginning and end of the summaries. The same preprocessing is applied to the test set. After that, we tokenize the sentences of the articles and the summaries, making each token to be uniquely identified by an integer. The text sequences are replaced by integers sequences and short sequences are padded with 0 until they have the desired length (max_text_len for articles and $max\_summary\_len$ for summaries). We can choose to remove rare words in the tokenization step. Now that we prepared the data, we have all ready to start building the model.

4 Extractive approach

In this section we will present the method we used to tackle the text summarization task with an extractive approach.

4.1 TextRank

As an extractive approach to summarization, we have chosen the algorithm TextRank that was firstly introduced in the paper by Rada Mihalcea and Paul Tarau in 2004 [2] as an application of the well known PageRank algorithm to text.

4.1.1 Overview of the method

The TextRank algorithm is a graph-based ranking model for text processing, so it's not a learning method and therefor it does not require training. It has been used for several NLP tasks with quite a degree of success. In summarization, for instance, the article is split into sentences. We compute the vector representation of each sentence as a mean of the word embeddings of the tokens, as we mentioned in the data preprocessing section. Then we compute a similarity matrix where each entry i, j represents the cosine similarity between sentence i and sentence j. Based on this similarity matrix, we run an iterative algorithm to compute the rank of each sentence. Arranging the sentences on the basis of their text ranks will give us the most important sentences which can be used as a summary. The number of sentences to choose for creating a summary is a parameter of our choice. Other types of similarity functions can be used instead of the cosine similarity.

4.1.2 Results

We applied TextRank to the test set of Wikihow containing about 5000 articles with different choices of the size of the summary. We assessed the quality of the results by computing the performance metrics ROUGE-1, ROUGE-2, ROUGE-L and METEOR in order to compare the values we obtain with the values retrieved by the authors. In Table 1 we can see the results on the entire Wikihow dataset provided by the authors in the original paper. In the same way, in Table 2 we can see the metrics that we obtained, this time only on the test dataset.

	ROUGE-1	ROUGE-2	ROUGE-L	METEOR
Results	0.27	0.074	0.20	0.13

Table 1: Performance metrics of TextRank given by the original authors.

Summary size	ROUGE-1	ROUGE-2	ROUGE-L	METEOR
0.05	0.1	0.02	0.06	0.06
0.1	0.18	0.04	0.12	0.10
0.2	**0.22**	**0.06**	**0.14**	**0.19**

Table 2: Performance metrics obtained for TextRank.

As we can see, we got the closest results to their values for a size of the summary of 0.2 (i.e 0.2 times the size of the article). However, there's still a slight difference due to the fact that we evaluate our algorithm only on the test set.

Since the performance of TextRank is quite low and the quality of its summarization is not very good, we will use this method as a baseline for our project and we will try to obtain better performing

methods by using more sophisticated NLP models.

5 Abstractive approach

In this section we will present and compare the different methods that we used to tackle the text summarization task with an abstractive approach.

5.1 Sequence to Sequence model with attention

5.1.1 Overview of the method

The Summarization task can be interpreted as a mapping between multiple length-variable text sequences, that is, as a Many-to-Many Seq2Seq problem. Hence, we will use a Seq2Seq architecture composed of an encoder and a decoder based on LSTMs or other types of sequence models. An illustration of this architecture is given in Figure 2. LSTMs are preferred as components of the model since they handle the problem of vanishing gradients that vanilla RNNs suffer from.

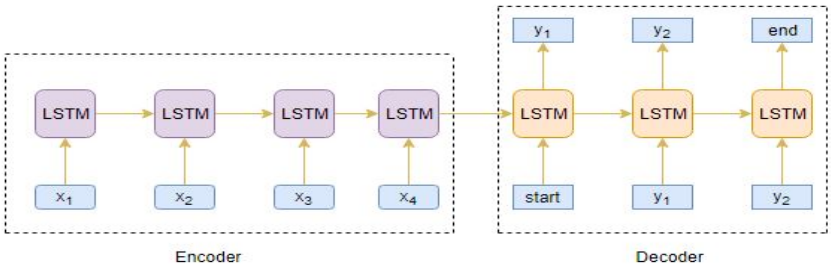

Figure 2: Encoder-decoder architecture with LSTM

During the training, at each time step, the encoder processes a token at a time and updates its hidden and cell state. When the entire sequence is fed to the encoder, the final states are used to initialize the decoder network. The decoder is trained to start generating the output sequence depending on the information encoded by the decoder. During inference phase, the desired output (summary in our case) is unknown to the decoder. A basic approach for inference is the following:

1. Pass the entire input sequence to the encoder and initialize the decoder with the internal states of the encoder.
2. Pass *START* token as an input to the decoder.
3. Run the decoder for one time step with the internal states.
4. The decoder outputs a probability distribution across the vocabulary for the next word. So we select the word with maximum probability.
5. Pass the retrieved word to the decoder for the next time step in order to update the internal states.
6. Repeat steps 3-5 until generating *END* token or reaching the maximum length of sequence.

We can use the Beam search technique instead of the greedy selection of the word of maximum probability at each time step. However, the encoder-decoder architecture as we described has some

limitations. It doesn't work well for very long sequences because it is hard to represent them with a fixed length vector. One possible solution for this issue is to use the attention mechanism. It aims to predict a word by looking at a few specific parts of the sequence only, rather than the entire sequence. The global attention mechanism is described in Figure 3. At each time step i of the encoder and each time step of the decoder j, we compute their hidden states and define an alignment score e_{ij}. It can be for example the dot product between the two hidden states. After that, attention weights a_{ij} are computed by applying the softmax function to score functions. The context vector c_i is calculated as a linear sum of products of the attention weights a_{ij} and the hidden states of the encoder. Finally, at time step i, this context vector is concatenated with the hidden state of the decoder and fed to a dense layer to produce the output y_i.

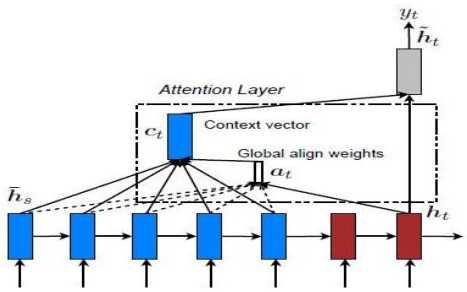

Figure 3: Global attention

5.1.2 Results

We have trained this model during 126 epochs with the RMSprop Optimizer. We have used a learning rate of 10^{-4} that was reduced by a factor of 0.9 when the loss was not decreasing (*Reduce on plateau* method). The summary of the loss obtained for this model can be seen in Figure 4. We can see that the loss was still decreasing when we stopped the training, so a further training with more computational power could increase the performance.

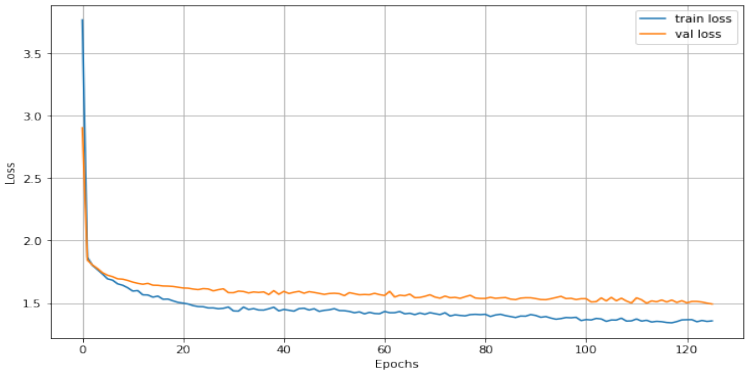

Figure 4: Encoder-decoder architecture with LSTM

The results of evaluating this model with the test set can be seen in Table 3.

We can see that this model has better performance that TextRank for the ROUGE-2 and ROUGE-L metrics, whilst worse results when evaluating with the ROUGE-1 or METEOR metrics. We strongly

ROUGE-1	ROUGE-2	ROUGE-L	METEOR
0.18	0.07	0.15	0.09

Table 3: Performance metrics of Seq2Seq model.

believe that the performance of this model can be further improved by using more training epochs.

5.2 T5 Transformer

In this section we explore another approach based on what is known as *transfer learning*. This kind of technique consists on using models that have been trained to learn a specific task to perform another somewhat related task. Before using the model to this new task, it's necessary to make a small new training to let the model learn the new task, what is called *fine-tuning* the model.

5.2.1 Overview of the method

Another novel method that can be used for abstractive summarization is the Text-to-Text Transfer Transformer (T5) model. This model is a type of Transformer encoder-decoder architecture that can be trained on a variety of tasks. It was introduced by Google AI in 2020 [7] and is based on formulating different text processing problems as text-to-text tasks and using an attention architecture very similar to the one originally proposed by Vaswani et al [6]. Firstly, the model maps the input tokens to a sequence of embeddings that is passed to the decoder, formed by self-attention followed by feed-forward networks. It also uses layer normalization, residual skip connections and finally dropout to avoid overfitting.

This model was pretrained with a big unlabeled dataset called *Colossal Clean Crawled Corpus* (C4), that is two orders of magnitude bigger than the entire Wikipedia corpus, which provides the model with a huge flexibility to perform different NLP tasks.

5.2.2 Results

In this section, we analyze the results of a fine-tuned T5 model for the summarization task on a subset of the Wikihow dataset. We used a tool called *Wandb*[3] (Weights and Biases) that allows us to keep track of the model performance as well as its parameters. The data preprocessing step is a bit different from previous models. In fact, we create a *Dataset* class that accepts our Wikihow as a *DataFrame* and we tokenize the text and summary using the T5 tokenizer. The data is afterwards split into a training and a validation set. We use the a *Dataloader* with a defined batch size because We can't load all the data at once in the memory.

Due to ressource limitations, we only do two training experiments: one with 8000 samples and another with 16000 samples in the training set. The number of training epochs is 2, since it was taking a lot time to compute each one of them. Even if this number of epochs might look very small, it's important to note that it was part of the fine-tuning process with an already trained language model. The maximum length for summary sequence is of 150 and the maximum length for the text sequence is 512. We compute the model metrics on unseen data, with 2000 samples and 4000 samples, respectively. The results are reported in Table 4. Moreover, the evolution of the training loss with respect to the time steps can be seen in Figure 5.

The performance metrics for the T5 model on a test set are already higher that the metrics we obtained using TextRank. We think that by training the model on a larger training set we could enhance even

[3] `https://wandb.ai/`

Training samples	Test samples	ROUGE-1	ROUGE-2	ROUGE-L	METEOR
8000	2000	0.32	0.11	0.24	0.18
16000	4000	**0.33**	**0.12**	**0.26**	**0.19**

Table 4: Performance metrics of fine-tuned T5 on validation set.

Figure 5: T5 Training loss, training data size = 8000

more the performance.

6 Qualitative comparison of the implemented methods on a few examples

In this section, we compare the predictions of summarization models on unseen articles. We also use a pretrained T5 on summarization provided by HuggingFace without fine tuning. The use of such model is quite straightforward. In Figure 6 we can see a few code lines that we run to get a prediction with the pretrained non-fine-tuned model.

```
from transformers import T5Tokenizer, TFT5ForConditionalGeneration
### load the model and the tokenizer
tokenizer = T5Tokenizer.from_pretrained('t5-base')
model = TFT5ForConditionalGeneration.from_pretrained('t5-base')
##Input text to summarize
input_ids = tokenizer("This method looks to get as much of the air and mineral impurities out of the water as possible before freezing, \
start with already distilled water. Filtered bottled water will work, or any water purified by reverse osmosis.;\n, \
Boiling removes air bubbles from the liquid, allowing the water molecules to stick together even harder in the freezer.\n\n\nAfter boiling the first time,\
 allow the water to cool. Then boil again.\nKeep the cooling water covered to prevent any dust from collecting on the surface.\n\n, \
 Make sure that you have let the water cool a bit before pouring it in the tray so that it doesn't melt the plastic. \
 If you really want to impress, try making extra-large clear ice cubes and ice spheres.\
  Nothing like drinking a cocktail on one very large rock.\n\n, Leave it for few hours to freeze.\n\n,",
                    return_tensors="tf").input_ids  # Batch size 1
##Prediction
output = model.generate(input_ids)
preds = [tokenizer.decode(g, skip_special_tokens=True, clean_up_tokenization_spaces=True) for g in output]
```

Figure 6: Python code for the use of a pretrained T5 model.

In order to make a qualitative comparison based on the human perception of language, we select three different articles from the WikiHow dataset and analyze the results given by the different models. We present them, as well as the ground truth summaries, in Table 5.

We observe that the TextRank model is able to select good sentences for the summary if we correctly fix the size of the summary and also if the sentences are not long. However, it doesn't capture all the steps described in the text. On the other side, the fine-tuned T5 models yields better predictions than the non-fine-tuned one. In addition, it captures the fact that the sentences of the summary should start with verbs. However, it tends to generate small summaries compared to ground truth and there are some punctuation problems. The non-fine-tuned T5 doesn't yield complete sentences and the output is even shorter.

7 Conclusions

All along this document we have analyzed different methods for text summarization. We have seen both extractive and abstractive approaches, with different degrees of complexity and also with different results. This experiments have given us a good insight about the techniques used in NLP: from adaptation of algorithms coming from other domains (such as PageRank) to more specific techniques that are nowadays part of the state of the art (transfer learning with Transformer architectures) in multiple language-related tasks.

We have performed a comparison of several summarization methods and we have familiarized ourselves with the complexity of text processing techniques, given the vast amount of data and also the long training times required to obtain a good-performing model.

It's also important to note that in order to obtain more results related to this task, it would be interesting to have more computation resources.

References

[1] M. Koupaee, M. and W. Y. Wang. *WikiHow: A Large Scale Text Summarization Dataset.* arXiv preprint arXiv:1810.09305. 2018.

[2] R. Mihalcea and P. Tarau. *TextRank: Bringing Order into Texts.* Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP. 2004.

[3] T. Mikolov et al. *Efficient estimation of word representations in vector space.* CoRR, 2013

[4] J. Pennington, R. Socher, and C. Manning. *Glove: Global vectors for word representation.* In Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543, 2014.

[5] A. M. Rush, S. Chopra, and J. Weston. *A neural attention model for abstractive sentence summarization.* EMNLP. 2015.

[6] A. Vaswani et al. *Attention Is All You Need.* Advances in Neural Information Processing Systems, 2017.

[7] C. Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer.* JMLR, 2020.

[8] S. Banerjee and A. Lavie. *METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments*, Proceedings of the ACL 2005 Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization, 2005.

[9] K. Papineni et al. *BLEU: a method for automatic evaluation of machine translation.* In ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, 2002.

Ground truth	TextRank	Seq2Seq	T5 (fine-tuned)	T5
use pure water. boil the water twice. pour water into an ice tray or other mold and cover with a plastic wrap to keep out particles. place the ice tray in the freezer. take out the tray and gently remove your clear ice cubes.	Boiling removes air bubbles from the liquid, allowing the water molecules to stick together even harder in the freezer. After boiling the first time, allow the water to cool.	get impurities from air. boiling dripping liquid. cool a bit ice cubes	boil the water. pour the water into a freezer tray. make ice cubes and spheres.	the water to cool. let it cool. ice cube
Download the app. Get the app by going the app store on the iPod or iPhone or iPad. Download the software. Go to www.skjm.com to download the software. Start up the software. Create a password.You don't need to care about the password just type in 'g' and press enter. Start up iCam on the iPhone or iPod or iPad.Go to the info sign type in the password.	If the webcam doesn't have audio you can still use it.	click the file. plug it in „ download	download and install the webcam course. click on the link that you just downloaded., click on the ""icamscourse"" tab., open the app	It's about 1/150th the cost of a brand new video camera
if your cat is snoring, take your cat to the vet. give your cat some medication. get your cat's weight down.	Your vet can tell you the best and safest type of medication to stop your cat's snoring.If you do use medication to decrease your cat's snoring, be sure you follow the directions very carefully. Snoring in cats can be caused by obesity, so getting your cat's weight down can help reduce snoring.	take your cat. follow the directions. feeding them different	take your cat to the vet. try medication. get your cat's weight down.	you can take them to the vet. snoring is not a

Table 5: Comparison on a few examples. The TextRank setting had a ratio of 0.2.