Generating Short Description of Videos

Karthikk Tamil Mani[1], Srikrishna Sasidharan[2], Thanigaiselvan Senthil Shanmugam[3], and Muthukumar Rajendran[4]

[1]NLP course project, Winter 2020

April 5, 2020

Abstract

'Generating short description of videos' is an academic project in which various extractive text summarization methodologies are used to generate a short description of videos. In this project, Sentence embedding and word embedding techniques such as TF-IDF, Glove/Fasttext, Skip thought vectors, and Universal sentence encoder is used. Along with these embedding techniques, K-means clustering, Smooth inverse frequency, and Page ranking are used to generate an extractive summary of the videos. Additionally, ROGUE metrics are used to evaluate the performance of the generated summaries with the golden summaries. Based on the results observed, Universal sentence encoder outperformed other summarization techniques.

1 Introduction

Our goal in this project is to generate a summary or a description of the videos. Videos, in general, have some disadvantages over textual data. Some of the limitations include not being able to search for the vital sections and skim through the videos. One of the other critical aspects is determining if our desired content appears in the video or not. Hence instead of watching an entire 10-minute video, if there's a way, we could learn the material in a minute. Other places where this summary could be helpful is in terms of search. Currently, videos are being searched with the title, which is a mere five or six-words in length. So with these summaries, the search can be much more effective and efficient since it is a text search. Other advantages of these summaries include providing more filters, classifying the videos precisely (topic modelling).

2 Implementation

2.1 Dataset extraction

The extraction of the dataset is an integral part of this project as it deals with summarizing the text generated by the videos. To begin with, We tried to convert the YouTube videos into the expected text format.

2.1.1 Converting videos to Text

We have explored various methodologies to convert videos into the text format, and One such exploration is using Google's speech to text API. The process was time-consuming, as the API needs to listen to the complete video in real-time to convert into the text. In some cases, there were issues with the accent, and the API could not understand it. As the above process was time-consuming, we researched other methods in which the subtitles can be extracted. We decided to extract the subtitles that are auto-generated by YouTube. A python program was developed that uses packages such as BeautifulSoup and Selenium to Web scrape the YouTube web page to extract the YouTube subtitle

2.1.2 Extracting subtitles from YouTube

To have the proper dataset for the text summarization, we searched for the video channels that contain the short description of the videos along with the subtitles, as this can be used as the golden summary to evaluate the generated summary. We used the videos uploaded by the 'Global news' channel on YouTube because the videos had both short descriptions and subtitles in it [1].

Additionally, The Global news mostly had videos in the English language, and since it is a new channel, the video they uploaded was closely matched with an article in a newspaper but in the form of video. The python program developed to download the subtitles and extract the short-description that was run for two days. The program obtained 1688 captions and short-descriptions.

2.1.3 Adding Punctuation to Subtitles and Short-description

The extracted data did not have any punctuation to it. We understood that an article lacking punctuation could significantly affect the performance of text summarization. We started exploring the ways to add the necessary punctuation to an article automatically.

We used two neural network-based pre-trained punctuator models.

1. DeepCorrect
2. Punctuator

DeepCorrect: DeepCorrect [3] uses the seq2seq model to punctuate the text. The python package was installed, and with the help of the pre-trained model and python program, we punctuated the text. One disadvantage of the Deepcorrect was it has the maximum input length of 200. We tried to tweak the model by changing the input length. Since we were using a pre-trained model, we are unsure how the efficiency of the model will be affected based on the max input value. This disadvantage of Deepcorrect forced us to use the other punctuation model punctuator, which does not have an upper limit.

Punctuator: Punctuator [4] is a bi-direction recurrent neural network-based punctuator model that uses attention mechanisms to restore the missing punctuation in a text. We developed a python program that uses the Punctuator model and updates the punctuation for all the 1688 subtitles and short description.

2.1.4 Drawbacks of YouTube dataset

Some of the short description extracted from the YouTube videos does not meet the standard of it to be a golden summary. In some cases, short descriptions were empty. So we decided to use the extracted data for generating the short description. But to evaluate the performance of the summarization, we required a data set that has both articles and the short description that qualifies to be a golden summary.

We found a BBC data set that had both articles and a short description [6]. But the number of articles were relatively less to develop an abstractive based text summarization, so we decided to implement various extractive based summarization and evaluate the performance of it.

2.2 Approaches

2.2.1 TF-IDF SentenceRanking

TF-IDF, short for term frequency–inverse document frequency, is a numeric measure that is used to score the importance of a word in a document based on how often did it appear in that document and a given collection of documents.

$$\text{Term Frequency (TF)} = \frac{\text{Number of times the word appears in a document}}{\text{Total number of words in the document}}$$

$$\text{Inverse Document Frequency (IDF)} = \frac{log(\text{Total number of documents})}{\begin{array}{r}\text{Number of documents with the word(w)}\\ \text{in that document}\end{array}}$$

$$TF - IDF = TF * IDF \tag{1}$$

Here for summarizing the text we need to pre-process the text. Pre-processing the text includes removal of special characters, digits, one letter words and stop-words. Next we need to calculate the frequency of each word. This has to be performed on the pre-processed dataset. The frequency of the words in the document is calculated by counting the unique words in the document.

Then we have to calculate the sentence score. This is done using the TF-IDF score of each word. This process involves the Parts Of Speech tagging which will return only the noun and verb phrases. These returned phrases are used to calculate the TF-IDF score of that particular word.

In-order to calculate the TF-IDF score of the word we need to find TF and IDF separately. Here TF is calculated as the number of times a word appears in the sentence to the total number of words in the sentence. Whereas, the IDF value can be found by dividing the total number of sentences by number of sentences containing the word and then taking the logarithmic of that value. Now by taking the product of the TF and IDF value we can get the TF-TDF value of the word.

In-order to find the important sentence we should take the individual sentences from the tokenized sentences and then compute the sentence score. Once after calculating the score, the sentences are sorted from highest to the lowest order. Based on the retention percentage given by the user, the summary is generated for the particular input.

2.2.2 Glove/FastText vector and SIF

Word embedding is a popular technique in NLP to achieve extractive text summarization. A numerical representation of a sentence was needed to estimate the similarity between two sentences—conventional techniques like Skipgram or a bag of words can be used to get a numerical representation of a word. The Glove/ FastText word embedding provides a multidimensional vector representation of a word in a sentence. So, rather than using conventional techniques, We have tried to use the latest approach to generate word vectors using the Glove model. For this, we used a pre-trained Glove vector model from Stanford[9]. The corpus used by Stanford to train this model is Wikipedia. We loaded this pre-trained model in our project to generate a 300-dimensional vector for each word from a news article.

Although we were able to generate vectors for each word in the article or text, we were still not able to compute the similarity between two sentences, i.e. we didn't vectorize each sentence of an article. We came across the need to compute sentence embeddings. We initially tried to estimate the average all word vectors of a sentence to get the sentence vector. But later, we studied that this approach is not useful because even the irrelevant words in a sentence gain higher weights compared to the relevant words. Due to this problem, we looked out for other possible methods to group all word vectors in a sentence into a single vector space, and we used Smooth Inverse Frequency(SIF).

The smooth inverse frequency (SIF) sentence embedding has a competitive performance compared to other approaches [11]. The weighing average of a sentence using SIF formula

$$a/(a + freq) \tag{2}$$

where a is a constant value a=1e-3, and freq or p(w) is the estimated frequency of the word in a reference corpus.

We then multiplied this value with the glove vector for each word in the sentence and computed the summation of all these vectors to obtain a sentence vector.

After computing sentence embedding vectors for all the sentences in the document, We wanted to estimate which sentences are similar in the article. The main reason to compute similarity is we needed to know which sentences are in the actual context of the article. There are several ways to compute similarities between two vectors, and the most common approaches are cosine similarity and manhattan similarity. We calculated cosine similarity for all two combinations of sentences in an article. PageRank is used to rank the sentences that have a higher similarity to the context of the article. Based on ranking, sentences with high similarity are grouped to form the summary. Similar to the Glove word vector representation, we also tried using the pre-trained model of FastText word embedding[10]. The end results of using FastText with SIF were quite similar to that of using Glove vectors.

2.2.3 Skip-thought vectors with K-means clustering

Skip-thought vectors with K-means clustering is a sentence embedding based text summarization technique. In this approach of extractive text summarization, we used Skip-thought vectors to place the semantically similar sentence in the vector space. The k-means clusters will then use this to group the related sentence in the cluster and summarization of the article can be formed. The initial step involved in this approach is to tokenize the article based on the sentence. To do this, we have used the NLTK sentence tokenizer to tokenize the articles based on the sentence. The skip thought vector would then use the generated output thought vectors to place the semantically similar sentence in the same vector space. We have used a pre-trained model [5] to achieve the results. Skip-thought vectors have two networks an encoder network and a decoder network. In the Encoder network, it will typically be a GNU-RNN, which generates a vector representation for the given sentence. Similarly, the decoder network will predict the previous and the next sentence by using the vector representation. It is observed that the semantically similar and relative sentences will be next to each other.

k-means clustering is used to group the related sentences in clusters. The number of clusters will be decided dynamically based on the number of lines in the article. Based on the vector space representation generated by the encoded network in the skip-thought vector, the K-means groups the sentence into clusters. The number of clusters will be directly proportional to the number of sentences in summary. All the sentence in a cluster represents the closely similar meaning, and a candidate sentence will be chosen from each cluster to represent their group in summary.

The candidate sentence will be chosen for which the sentence lies near the cluster center. The order of the candidate sentences in the summary will be the same as the order of the sentence in the original article.

2.2.4 Universal sentence encoder

Universal sentence encoder is a multi-task learning model that combines several training objectives in one training scheme. In this project, we have used Google's pre-trained universal sentence encoder for sentence embedding. The universal sentence encoder is trained on a one-to-many multi-tasking learning framework to learn a universal sentence embedding by switching between several tasks. The reason we went with universal sentence encoder is that we wanted an embedding technique that could embed semantically similar sentences. This encoder understands the syntactic properties better as it is used in the multi-language neural machine translation task [2]. E.g., ”What is your age?” ”How old are you?” although are different sentences and a different set of words, this encoder embeds them as 99% similar. So our idea was to find sentences that are most semantically similar to one another and present those sentences to be the most important sentences in the context meaning the abstract of the paragraph.

For grouping the semantic similarity and finding the best, we went with the famous PageRank algorithm. PageRank algorithm will rank the sentences in order of their semantic similarity with all other sentences. So the most similar sentence is ranked the 1st. We used NLTK to tokenize the sentences and used the combination of universal sentence encoder and PageRank algorithm to rank those sentences. After listing the sentences, the top 10 sentences or half the number of sentences whichsoever is minimum is taken and is considered to be the summary of the paragraph.

3 Experimental Results/Evaluation

To evaluate the generated summary by various extractive summarization techniques, we used ROUGE. ROUGE(Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics for assessing the summarization of texts. ROUGE metrics should have a golden summary for each generated text summarization. ROUGE will use this golden summary to compare with the generated summary and calculates precision, recall and F-measure. ROUGE-1 refers to the comparison of unigrams between the generated summary and the golden summary. ROUGE-2 refers to the comparison of bigrams between the generated summary and the golden summary.

We chose F-measure as the evaluation factor because we wanted to check the optimal efficiency of these summarization techniques. We estimated the F-measure of a model to understand how much it deviates from the golden summary and to evaluate how much information the model captures that is relevant to the golden summary.

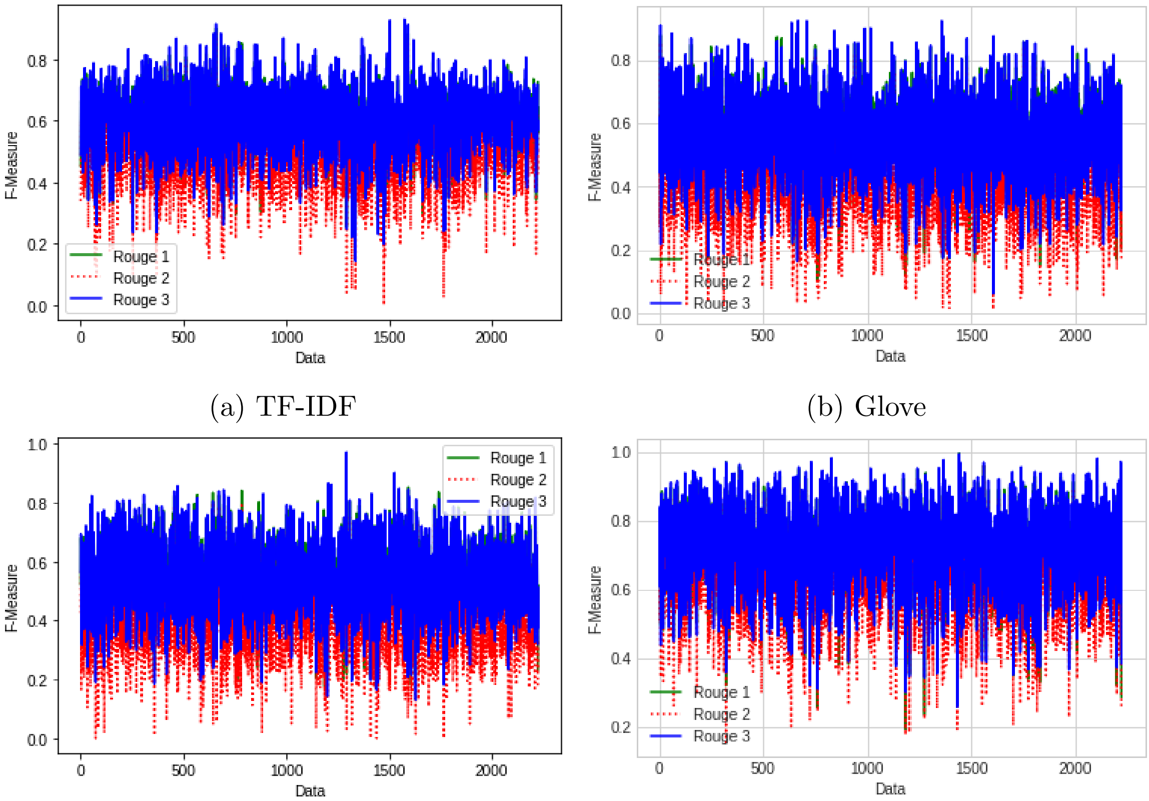
We have plotted the values obtained from the ROUGE metrics to visualize. Figure 1. represents the ROUGE metrics F-measure value generated for all the summary and Figure 2. represents the mean F-measure value obtained for the extractive summarization techniques. It is observed that the sentence embedded based summarization techniques performed better when compared with the word embedded based technique.

Table 1: Mean F-measure value for extractive text summarization techniques

Mean F-measure			
Summarization Technique	**Rouge-1**	**Rouge-2**	**Rouge-L**
Universal Sentence Encoder	0.72069	0.64721	0.72473
TF-IDF	0.61772	0.54188	0.63172
Skip-Thought with k-means clustering	0.55391	0.4153	0.53905
Glove word embedding with SIF	0.57346	0.44517	0.56018

Universal sentence encoder provided 70% of matching summary when compared to the results with the golden summaries. In comparison, the other summarization technique yielded 60%, 55% and 57% for TF-IDF, Skip-thought vector and Glove vector techniques, respectively. Universal sentence encoder performed better because it understands the semantic meaning of each sentence and generates a summary based on the context of the sentence.

The table 1 represents the numerical mean F-measure value of the used extractive text summarization technique.



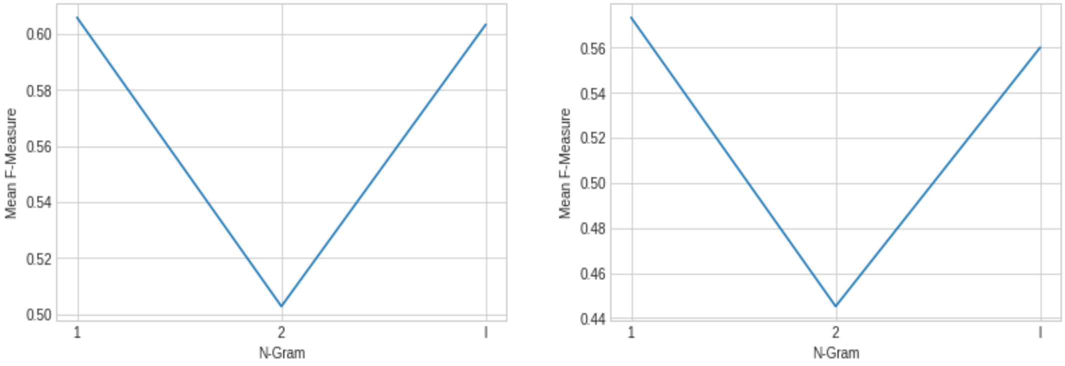
(a) TF-IDF

(b) Glove

(c) Skip-Thought

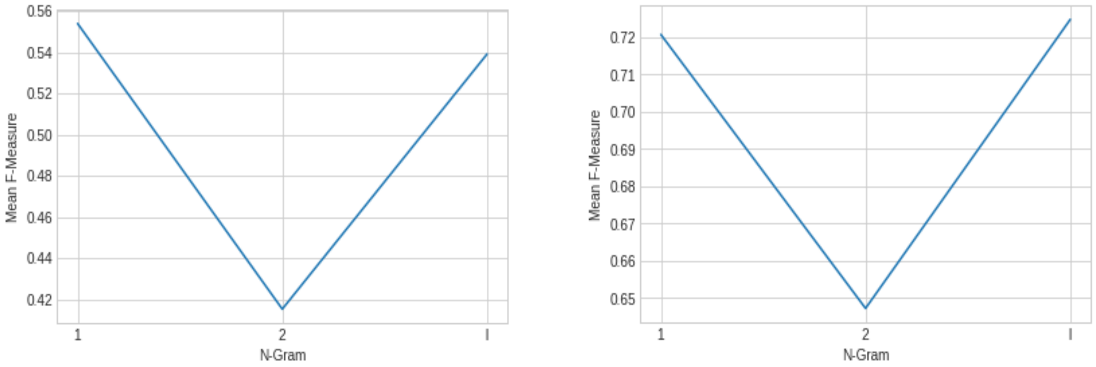
(d) Universal Sentence Encoder

Figure 1: F-measure of Rogue 1,2 and l metrics for all articles in Extractive text summarization techniques



(a) TF-IDF

(b) Glove



(c) Skip-Thought

(d) Universal Sentence Encoder

Figure 2: Mean F-measure of Extractive text summarization techniques

4 Conclusion

The universal sentence encoder outperformed other techniques since it is a multi-task learning model and the state-of-the-art one. Other word and sentence embedding techniques performed almost the same. For sentence embedding techniques, they were even performance changes between spacy and NLTK sentence tokenizers. This was the same with word embeddings too. Glove and Fasttext performed almost the same. Our analysis was done on the BBC dataset, but this could be applied to any dataset even on the YouTube data that we extracted. The pre-trained encoding models we made use of are publicly available for research purposes. In the future, we would like to perform the abstractive method of summarization of the videos using modern and advanced DeepLearning techniques.

References

[1] Politics, "Global News," YouTube. 2020.

[2] T. Wolf, "The Current Best of Universal Word Embeddings and Sentence Embeddings," Medium, 14-May-2018. [Online]. Available: `https://medium.com/huggingface/universal-word-sentence-embeddings-ce48ddc8fc3a`. [Accessed: 20-Apr-2020].

[3] "punctuator," PyPI, 04-Oct-2019. [Online].
Available: `https://pypi.org/project/punctuator/`. [Accessed:20-Apr-2020].

[4] "deepcorrect," PyPI, 02-Apr-2019. [Online].
Available: `https://pypi.org/project/deepcorrect/`. [Accessed: 20-Apr-2020].

[5] Kushal Chauhan, "Unsupervised Text Summarization using Sentence Embeddings," Medium, 06-Aug-2018. [Online]. Available: `https://medium.com/jatana/unsupervised-text-summarization-using-sentence-embeddings-adb15ce83db1`. [Accessed: 20-Apr-2020].

[6] S. K. George, "BBC news dataset," Kaggle.com, 2018. [Online]. Available:`https://www.kaggle.com/shineucc/bbc-news-dataset`. [Accessed: 20-Apr-2020].

[7] Ashna Jain, "Automatic Extractive Text Summarization using TF-IDF," Medium, Apr-2019. [Online]. Available: `https://medium.com/voice-tech-podcast/automatic-extractive-text-summarization-using-tfidf-3fc9a7b26f5`. [Accessed: 20-Apr-2020].

[8] A. Sieg, "Text Similarities: Estimate the degree of similarity between two texts," Medium, 04-Jul-2018. [Online].
Available:`https://medium.com/adriensieg/text-similarities-da019229c894`.
[Accessed: 20-Apr-2020].

[9] J. Pennington, "GloVe: Global Vectors for Word Representation," Stanford.edu, 2014. [Online].
Available: `https://nlp.stanford.edu/projects/glove/`. [Accessed: 20-Apr-2020].

[10] “Wiki word vectors · fastText,” Fasttext.cc, 2016. [Online]. Available: https://fasttext.cc/docs/en/pretrained-vectors.html. [Accessed: 20-Apr-2020].

[11] Hicham EL BOUKKOURI, “How deep does your Sentence Embedding model need to be?,” Medium, 10-Jan-2019. [Online]. Available:https://medium.com/data-from-the-trenches/how-deep-does-your-sentence-embedding-model-need-to-be-cdffa191cb53. [Accessed: 20-Apr-2020].

[12] “beautifulsoup4,” PyPI, 05-Apr-2020. [Online]. Available: https://pypi.org/project/beautifulsoup4/. [Accessed: 20-Apr-2020].

[13] “Selenium with Python — Selenium Python Bindings 2 documentation,” Readthedocs.io, 2011. [Online]. Available: https://selenium-python.readthedocs.io/. [Accessed: 20-Apr-2020].