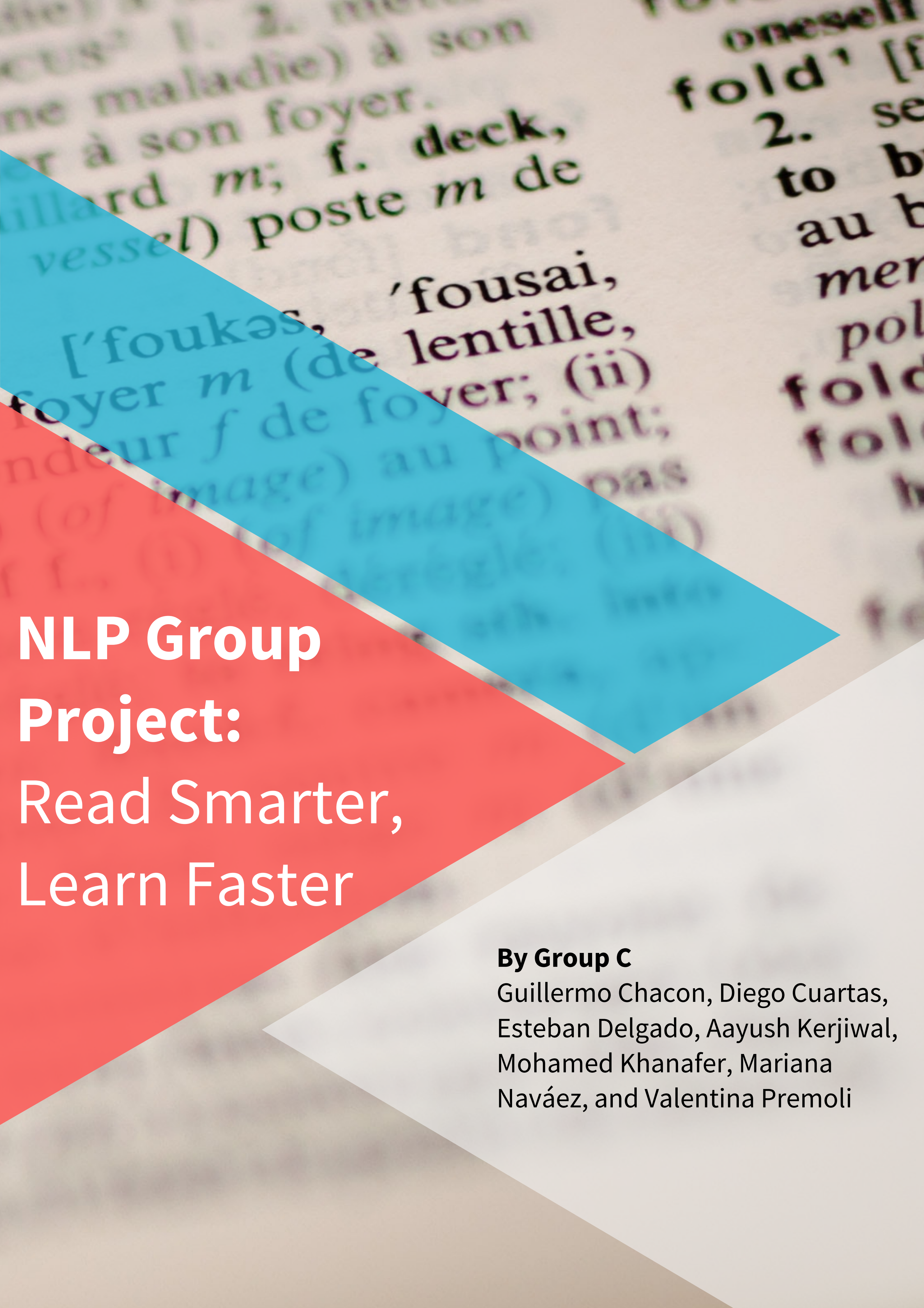


NLP Group Project: Read Smarter, Learn Faster

By Group C
Guillermo Chacon, Diego Cuartas, Esteban Delgado, Aayush Kerjiwal, Mohamed Khanafer, Mariana Naváez, and Valentina Premoli

Table of Content

Introduction

As we know, globalization and technology are changing the world we live in. This is producing data at an unprecedent speed and volume, generating an information overload. For this reason, innovative solutions that process and analyze large amounts of data are a need for companies striving to make the most profit and be at the top of their game.

But is it just analytics, running and deploying machine learning models? What do we do with all the data collected? How is it stored? How can we access it? How can we know what type of information we have in a fast way? This information overload has raised new opportunities in many different sectors and an interesting solution raised: Text summarization.

Despite not being a brand-new solution, text Summarization is a very relevant topic in recent years due to better performance of models in the way they summarized texts. In the educational sector, where time is a constraint, and knowledge is produced by the second, a challenging scenario is in front of students. Specially on higher education, where work overload for MBA programs is extenuating, and the amount of resources to be consumed are infinite.

In a small survey we conducted on 20 former MBA students at IE, we discovered that they were assigned an average of 120 business cases throughout the program. From such cases, nearly 60 % of them were read by the surveyed students. When asked about the reason they desist to read the other business cases, 80% of them answered that they were too long and time consuming.

With this reality so close to us, wouldn't be time efficient to have an overview of a long paper before deep diving on it? What about a brief summary of a complex business case? With that in mind, we created and present to you our Text Summarizer: **Read Smarter, Learn Faster.**

The project covers an application that allows students of Undergraduate and Graduate Programs at IE to summarize online sources and business cases, efficiently saving time in the research and studying stages. In order to accomplish this goal, we have analyzed and developed a set of text summarizers: 4 extractive and 1 abstractive text summarizers.

To be able to have solid models, we had to use proper data to evaluate our models. It was difficult to find business cases with their respective summaries that could be used. That is why we rather used a dataset provided by the BBC containing articles from 2004 to 2005 with their respective summaries. We specifically used the business-related articles to make sure our models were performant on business-related texts. Additionally, we tested our models on IE-provided cases to make sure their output was relevant. The metric used to assess the results of the models is Recall-Oriented Understudy for Gisting Evaluation (ROUGE), with a 3 granular output.

The best extractive summarizer is the Text Rank Model with scores of around 0.60 for R1, 0.47 for R2 and 0.57 for R-L. This model offered a better readability when it comes to the output. It considers semantically similar words by using the embeddings approach. We here dive deeper into the theory behind this algorithm. The script of the various mentioned models is provided along this report, we highly suggest spending a good amount of time reading it as most of our work can be understood from there as well.

We also provide a Web Application of our idea in order to fully illustrate its implementation. This application was developed using Flask and following tutorials referenced later on. The application is able to take any text as input and output the summary after running our python code in the backend. While this is just a prototype, our goal is to build on it and adding other features later on.

We now turn to elaborating on text summarization theory before explaining our thinking in the following sections.

Text Summarization

Definition

Automatic text summarization is the technique which automatically creates an abstract or summary of a text. Text summarization could also be defined as the process of automatically creating a compressed version of a given document preserving its information content.

The process of text summarization can be decomposed into three phases: analysis, transformation, and synthesis. The analysis phase analyzes the input text and selects a few features. The transformation phase transforms the results of the analysis into a summary representation. Finally, the synthesis phase takes the summary representation, and produces an appropriate summary corresponding to user’s needs.

Taxonomy

Automatic text summarization systems can be categorized into different types according to their approach:

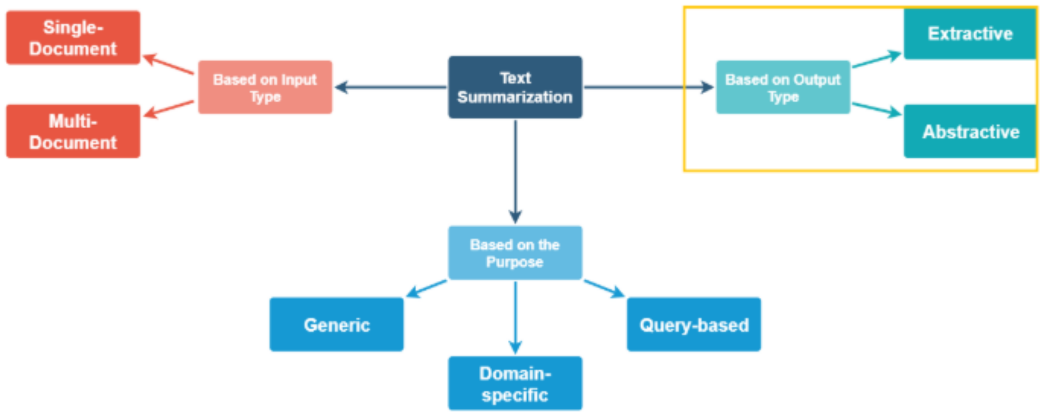


Image 1: Text Summarizers Taxonomy.

By input

A. Single Document: Single document summarization produces summary of single input document. The most known methods to tackle single input documents are the frequency-based approach, the feature based and the machine Learning approach.

B. Multi Document: Multi document summarization produces a summary of multiple input document. These multiple inputs are often documents discussing the same topic. The most popular methods are the cluster-based method and the graph-based method.

By purpose

A. Generic: This type of summarization aims at summarizing regardless of the content of the input.

B. Domain specific: Text summarizers in this category refer to a specific topic or domain. The use of specialized vocabulary for a patient clinical history, a scientific research, technology or engineering investigations are some examples in this category.

C. Query-based: Query-based text summarization is aimed at extracting essential information that answers the query from original text. The answer is presented in a minimal, often predefined, number of words.

By output

A. Extractive: Extractive summarization methods reduce the problem of summarization into the problem of selecting a representative subset of the sentences in the original documents.

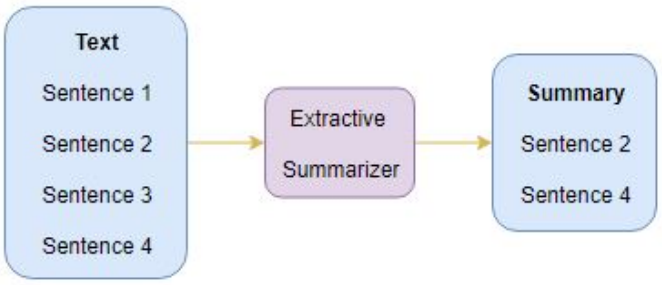


Image 2: Extractive Summarization

B. Abstractive: Abstractive summarization is the technique of generating a summary of a text from its main ideas. Abstractive summarization involves a process of understanding the language, the context and generating new sentences.

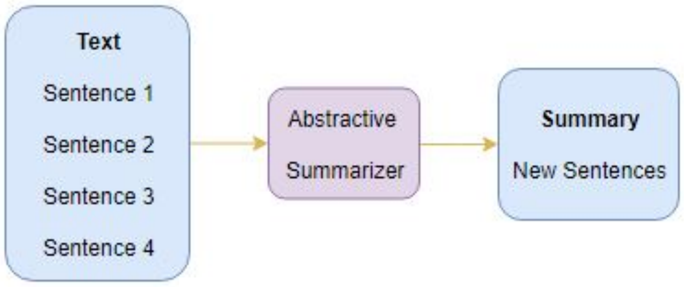


Image 3: Abstractive Summarization

In our case, we will focus on the output approach. Indeed, we will analyze Extractive and Abstractive types of summarizing.

Modelling

Datasets and Metrics

In order to assess the performance of the models, we need to have some texts and a corresponding human made summary. We found a Kaggle dataset containing over 400 news articles of the BBC from 2004 to 2005. For each article, summaries are provided. Because our use case focuses on summarizing texts related to business, we also needed articles from that sector, so we included MBA material in form of business cases for that matter.

To assess the performance of the models, we used ROUGE, which stands for Recall-Oriented Understudy for Gisting Evaluation which basically is a set of metrics for evaluating the summarization of texts. ROUGE presents its output in three different granular ways:

ROUGE-1: Which refers to overlap of unigrams between our model summary and the original summary.
ROUGE-2: Which refers to overlap of bigrams between our model summary and the original summary.
ROUGE-L: Which measures the longest matching sequence of words using LCS (which stands for Longest Common Subsequence).

Extractive Summarization

An extractive summary is essentially the original text, with a few sentences filtered out to make it shorter. In order to do this, we need some basis or score, which can be used to distinguish between the sentences.
We developed 4 alternatives for extractive text summarizing and ran them in 3 sample texts. In our first model, we used a very simple method to calculate such scores for sentences. We calculated the average term frequency of nonstop words in a sentence and filtered out those sentences who were below average in the above metric.
This method is very basic, and therefore, we changed the scoring method to use TF-IDF. This way the words were scores based on their relative importance in the entire piece of text, and therefore sentences with important words would be more important to our summary. Here, we still used a simple average score of words in a sentence.
Our next model would be based on cosine similarity, so that we can also incorporate some form of sentence importance based on the similarity to other sentences. This allows us to get sentences that represent ideas in the text that are most recurrent. Finally, Text Rank Algorithm is applied as an alternative to see if the model outperforms the previous three attempts.

Model 1: Summarizing based on word frequency

With term-frequency we represent each term in our vector space; the term-frequency is nothing more than a measure of how many times the terms present in our document.

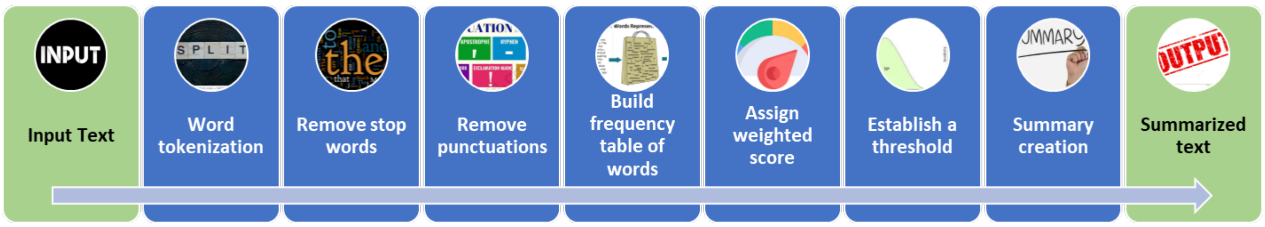
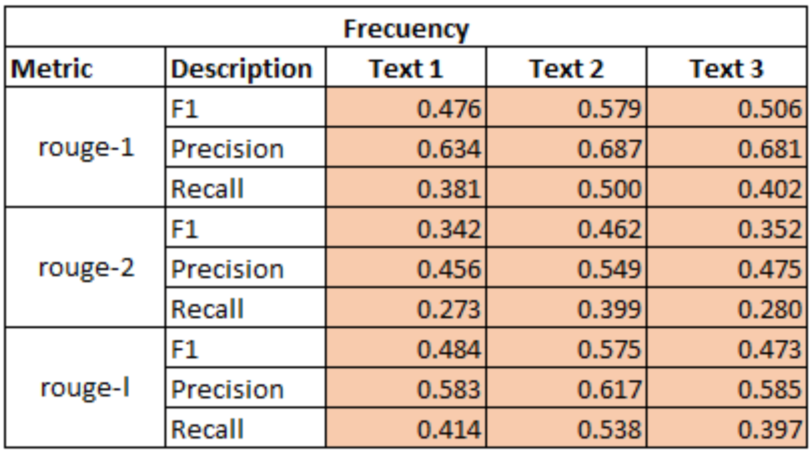


Image 4: Word Frequency Pipeline Text Summarizer

As seen in Image 4, the Input text is tokenized, cleaned and a frequency table is created. After getting the frequency table, we can get the word with the highest frequency and use it to divide the other words in the text. This is to get the weighted frequencies and reduce bias in long sentences. A threshold is then established to define the size of the summary and finally the summary is created.



		Frecuency		
Metric	Description	Text 1	Text 2	Text 3
rouge-1	F1	0.476	0.579	0.506
	Precision	0.634	0.687	0.681
	Recall	0.381	0.500	0.402
rouge-2	F1	0.342	0.462	0.352
	Precision	0.456	0.549	0.475
	Recall	0.273	0.399	0.280
rouge-l	F1	0.484	0.575	0.473
	Precision	0.583	0.617	0.585
	Recall	0.414	0.538	0.397

Image 5: Word Frequency Model Evaluation

Depending on the text we analyzed, the accuracy achieved is around 50%. We could say that being the output of an extractive summarizer, the result is not bad as a starting point.

Model 2: Summarizing based on TF-IDF Matrix

In our first model we only considered the frequency of the words. In this next step we decided to use the TF-IDF Matrix that has a broader grasp of the relevance of the word.
The tf-idf weight comes to solve the problem left by word frequency model. What tf-idf gives is how important is a word to a document in a collection, and that’s why tf-idf incorporates local and global parameters, because it takes in consideration not only the isolated term but also the term within the document collection.

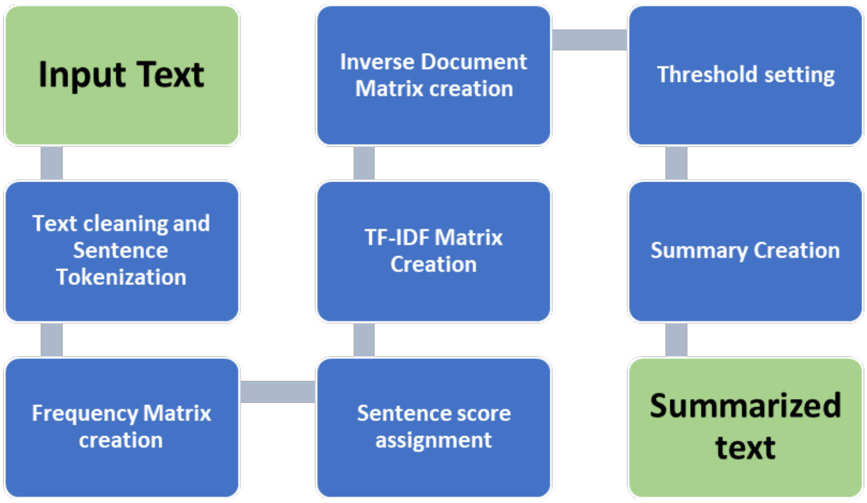
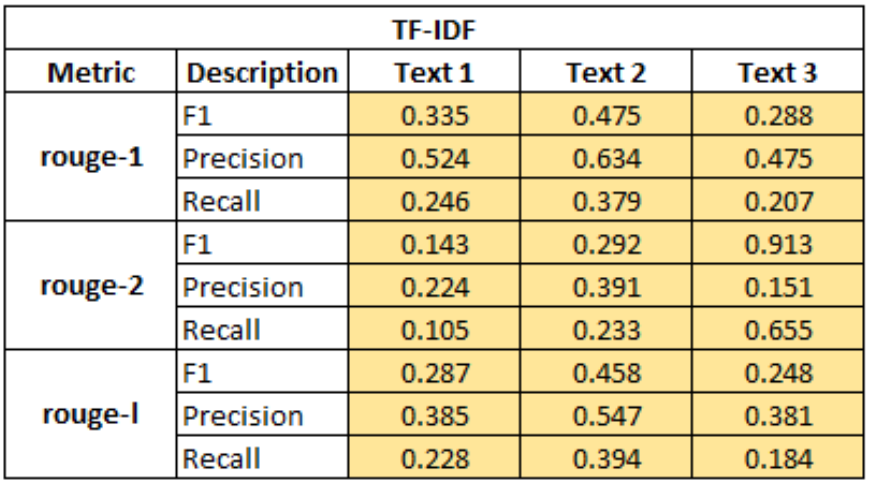

Input Text
Text cleaning and Sentence Tokenization
Frequency Matrix creation
Sentence score assignment
TF-IDF Matrix Creation
Inverse Document Matrix creation
Threshold setting
Summary Creation
Summarized text

Image 6: Word Frequency Pipeline Text Summarizer

For this second model, additional steps in the text cleaning stage are done. URL, HTML are removed, and contractions are replaced to their base forms. The text is then tokenized and the matrix TF-IDF is created. Scores are assigned and we finally set a threshold to determine which sentences to include in our summary.



TF-IDF				
Metric	**Description**	**Text 1**	**Text 2**	**Text 3**
rouge-1	F1	0.335	0.475	0.288
	Precision	0.524	0.634	0.475
	Recall	0.246	0.379	0.207
rouge-2	F1	0.143	0.292	0.913
	Precision	0.224	0.391	0.151
	Recall	0.105	0.233	0.655
rouge-l	F1	0.287	0.458	0.248
	Precision	0.385	0.547	0.381
	Recall	0.228	0.394	0.184

Image 7: TF-IDF Model Evaluation

This seems to be equally accurate than just taking the frequency of the words as a discriminant factor for the sentences as there is not a major improvement in the accuracy.

Model 3: Summarizing based on Cosine Similarity

The cosine similarity between two vectors (or two documents on the vector space) is a measure that calculates the cosine of the angle between them. This metric is a measurement of orientation and not magnitude. The range of values we can get go from 0 to 1, an represent how similar are two normalized vectors.

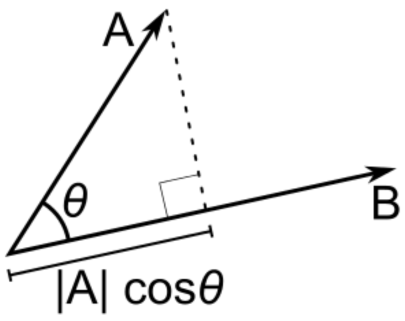
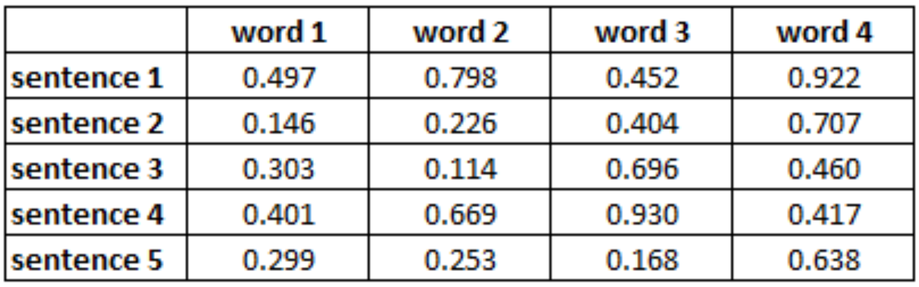


Image 8: Cosine Similarity

So how it is done?
First, we need to create the tf-idf matrix. For the rows of the matrix, the whole text is sentence tokenized and placed in the row position. For the matrix columns, each column is composed of the vocabulary of the document, which is created by word tokenizing and removing stop words of the text.

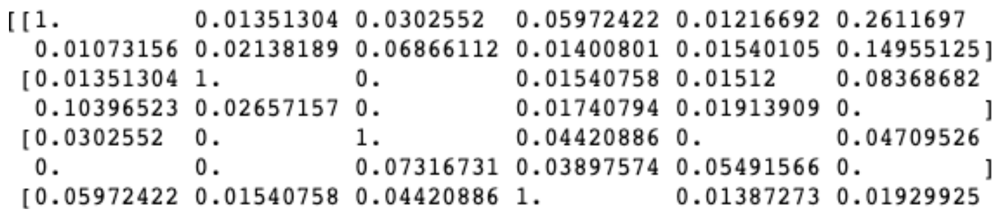
Using the TF-IDF weights for each sentence converts each row into a vector and stores them in a matrix, as seen in Image 9.



	word 1	word 2	word 3	word 4
sentence 1	0.497	0.798	0.452	0.922
sentence 2	0.146	0.226	0.404	0.707
sentence 3	0.303	0.114	0.696	0.460
sentence 4	0.401	0.669	0.930	0.417
sentence 5	0.299	0.253	0.168	0.638

Image 9: TF-IDF matrix creation

As second step, we find the cosine-similarity of each TF-IDF vectorized sentence pair. For example, the first sentence is paired to each sentence of the text (including itself) a cosine similarity is applied, returning values between 0 and 1.



```
[[1.         0.01351304 0.0302552  0.05972422 0.01216692 0.2611697
  0.01073156 0.02138189 0.06866112 0.01400801 0.01540105 0.14955125]
 [0.01351304 1.         0.         0.01540758 0.01512    0.08368682
  0.10396523 0.02657157 0.         0.01740794 0.01913909 0.        ]
 [0.0302552  0.         1.         0.04420886 0.         0.04709526
  0.         0.         0.07316731 0.03897574 0.05491566 0.        ]
 [0.05972422 0.01540758 0.04420886 1.         0.01387273 0.01929925
```

Image 10: Cosine Similarity

Image 10 represents the array of cosine similarity for sentence pairs. The first sentence in the article compared to itself gives us 1 as result. Then, the process continues until it covers the entire document.

Finally, after finding the cosine-similarity for all vectorized pairs, weights of each vectors are averaged and sorted to return the indexes of the vectors with the highest averages.

These indexes are then used to pull out the sentences from the original text for the summarization. The sentences with the highest average weights will capture the unique and important sentences from the original text. The Cosine Similarity model pipeline is shown in image 11.

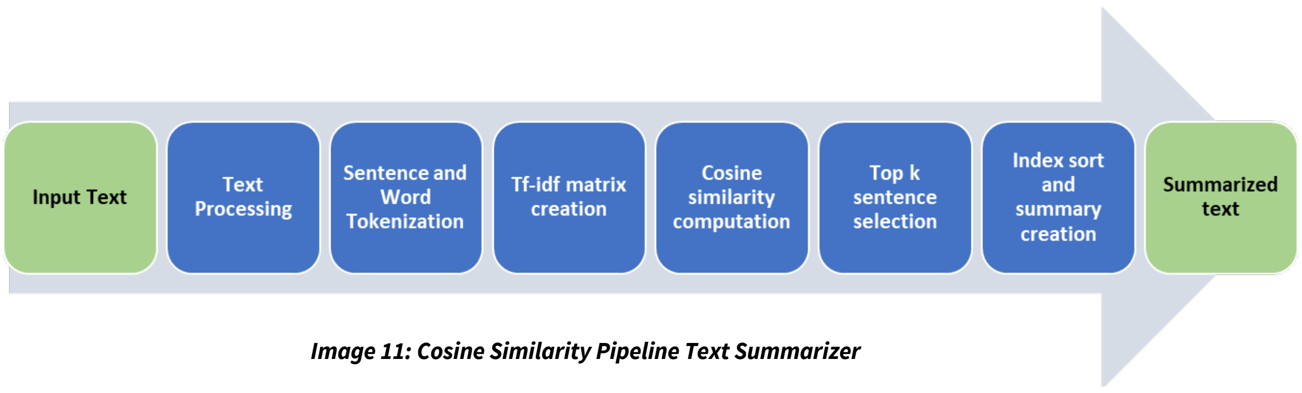
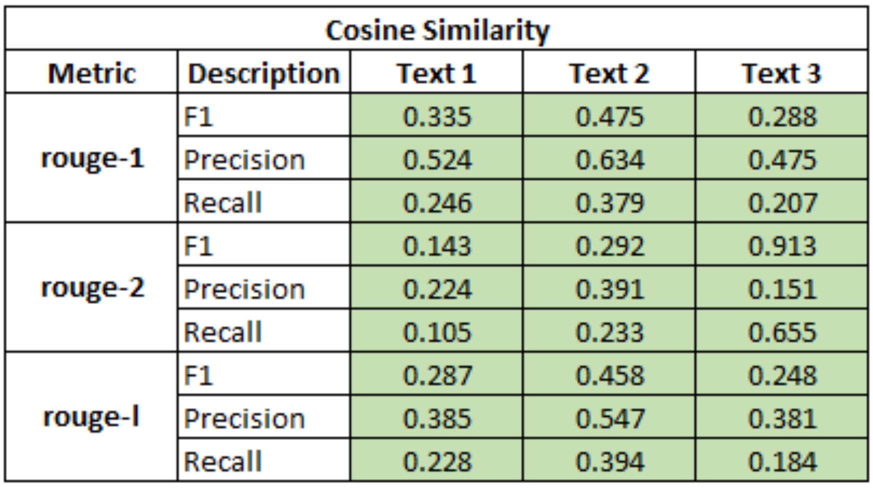

Input Text
Text Processing
Sentence and Word Tokenization
Tf-idf matrix creation
Cosine similarity computation
Top k sentence selection
Index sort and summary creation
Summarized text

Image 11: Cosine Similarity Pipeline Text Summarizer

Preprocessing the input text is the first step and includes stop words, special characters and punctuation removal. Sentence and word Tokenization based on N-Grams are executed to then create TF-IDF Matrix. Calculation of the TF - IDF score of each word in the sentence is applied and finally cosine similarity is executed for each pair of sentences. We select top k sentences and sort the indexes for the summary creation.



Cosine Similarity				
Metric	**Description**	**Text 1**	**Text 2**	**Text 3**
rouge-1	F1	0.335	0.475	0.288
	Precision	0.524	0.634	0.475
	Recall	0.246	0.379	0.207
rouge-2	F1	0.143	0.292	0.913
	Precision	0.224	0.391	0.151
	Recall	0.105	0.233	0.655
rouge-l	F1	0.287	0.458	0.248
	Precision	0.385	0.547	0.381
	Recall	0.228	0.394	0.184

Image 12: Cosine Similarity Model Evaluation

As before, the ROUGE metric seems to indicate a below average accuracy (Image 12), like the first 2 models. What we could say is that even if this scoring method is quite accurate, there are some issues. A common one is that words that are semantically similar, are not being leveraged separately.

Considering a semantic embeddings approach looks to overcome this shortcoming. We thus next turn on to trying to implement the Text Rank algorithm which takes into consideration the word embeddings as well as the cosine similarities between the sentences.

Model 4: Summarizing based on Text Rank Algorithm

The TextRank algorithm is based on the famous PageRank algorithm in which a matrix calculates the probability that a given user will move from one page to the next.

In our case here with TextRank, we calculate a cosine similarity matrix where we get the similarity of each sentence compared to the others. Then, a graph is generated from this cosine similarity matrix we calculated. The PageRank ranking algorithm is then applied to the graph to get the scores for each sentence in the text.

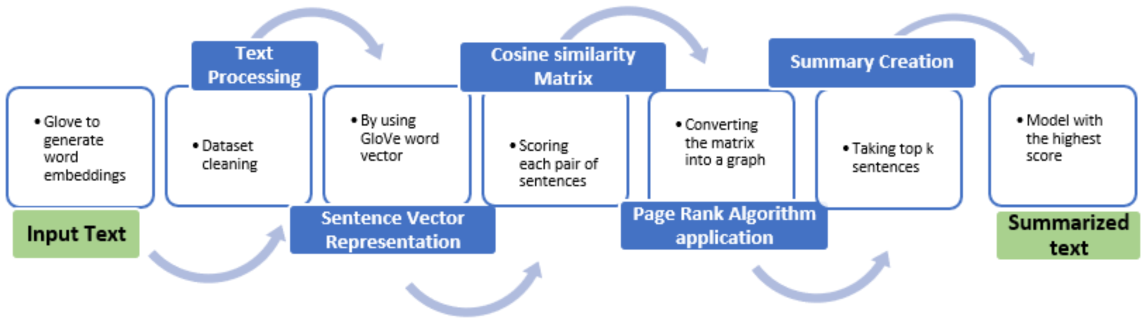
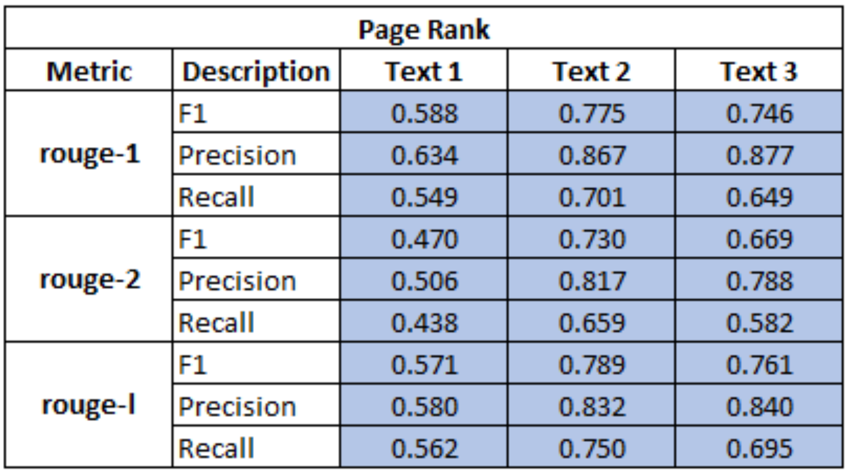

Input Text
Glove to generate word embeddings
Text Processing
Dataset cleaning
Sentence Vector Representation
By using GloVe word vector
Cosine similarity Matrix
Scoring each pair of sentences
Page Rank Algorithm application
Converting the matrix into a graph
Summary Creation
Taking top k sentences
Summarized text
Model with the highest score

Image 13: Text Rank Pipeline Text Summarizer

In our case here with TextRank, we calculate a cosine similarity matrix where we get the similarity of each sentence compared to the others. Then, a graph is generated from this cosine similarity matrix we calculated. The PageRank



Page Rank				
Metric	**Description**	**Text 1**	**Text 2**	**Text 3**
rouge-1	F1	0.588	0.775	0.746
	Precision	0.634	0.867	0.877
	Recall	0.549	0.701	0.649
rouge-2	F1	0.470	0.730	0.669
	Precision	0.506	0.817	0.788
	Recall	0.438	0.659	0.582
rouge-l	F1	0.571	0.789	0.761
	Precision	0.580	0.832	0.840
	Recall	0.562	0.750	0.695

Image 14: Text Rank Evaluation

As we can see in Image 14 this model outperforms the other three models being the best solution as an extractive summarizer model.

Abstractive Summarization

In abstractive summarization our model generates new sentences from the original text we give it. This is opposed to the extractive approach we have been following so far where we used only the sentences that were present in the input text. For this type of text summarization, we are using a different set of articles compared to the previous extractive text summarizers.

Model 5: Summarizing based on T5

After the recent success in various of their released algorithms like BERT, Google recently published another very powerful neural network pre-trained model, the "Text-to-Text Transfer Transformer" (T5). T5 is basically a very large neural network model that has been trained on a bunch of unlabeled text (the C4 collection of English web text) as well as labeled data from some popular NLP tasks. This model is then fine-tuned for each subsequent task that we want to solve. The performance those pre-trained models can achieve set the T5 as a state-of-the-art tool for many NLP tasks such as text classification, question-answering and summarization tasks.

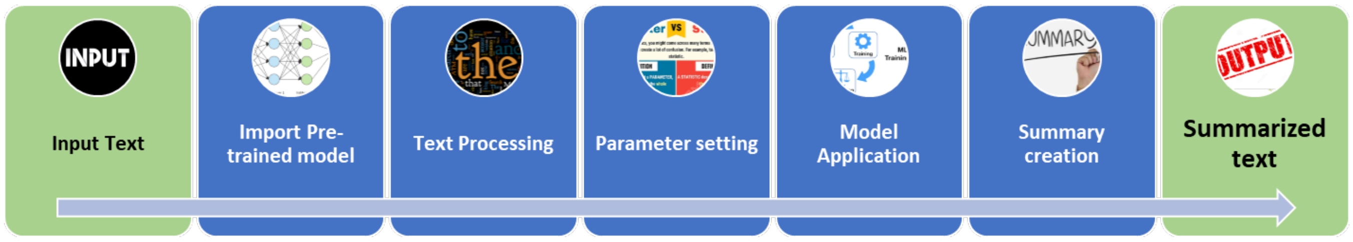


Image 15: T5 Abstractive Text Summarizer Pipeline

To have an abstractive summarization we first need to import a pretrained Conditional Generator model of T5. As a second step, the text is cleaned and tokenized and converted to tensors. Once the text is prepared, we run the model and obtain the summarized text. This text is later evaluated with ROUGE as we did with the extractive summarizer models.

T5			
Metric	**Description**	**Text 4**	**Text 5**
rouge-1	F1	0.689	0.323
	Precision	0.785	0.386
	Recall	0.614	0.278
rouge-2	F1	0.589	0.194
	Precision	0.672	0.232
	Recall	0.524	0.167
rouge-l	F1	0.684	0.346
	Precision	0.755	0.379
	Recall	0.625	0.319

Image 16: T5 Abstractive Text Model Evaluation

The performance of the abstractive seems worse than any of the extractive text summarizers. However, here is where we have to look at the limitations of the ROUGE metric and add our opinion:

- The ROUGE metrics only assess the content selection and do not really take into consideration quality aspects, like the fluency, grammar, or coherence;
- As we mentionned above, to measure the content selection, they rely mostly on the lexical overlap between the text generated and the reference summary. And here an abstractive summary is actually expressing the same content as the reference text without any lexical overlap.

Thus, comparing the metrics could not be very significant. So, we looked at the output of the model and concluded that the summary provided was better in terms of fluency and grammatical flow. This could be explained by the fact that abstractive summarization has more resources (words) to do the summarization. But the con of it is that the summary is very short, and it is hard to extract longer than 70 words summaries.

Final Model Performance & Selection

Extractive Summarizers Comparison

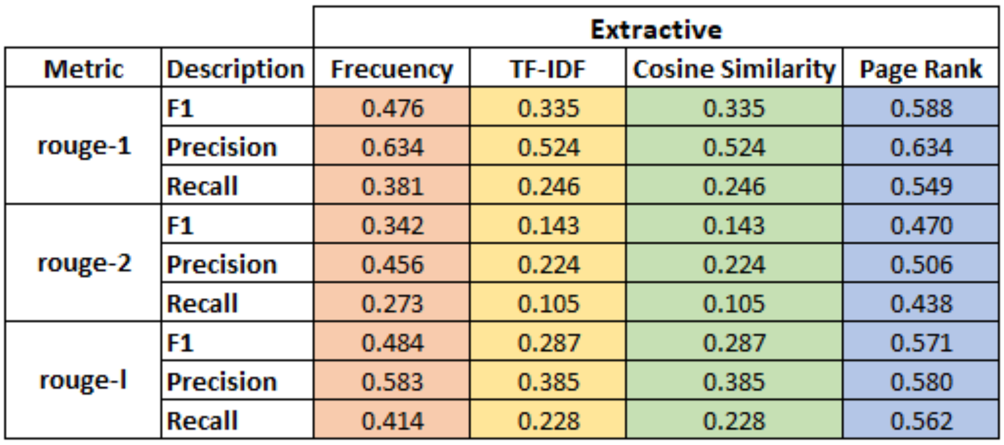
To summarize the different models obtained and analyzed here is a brief wrap up:

- Model 1 basic summarizer: this model assigned a score to each sentence according to the words in the frequency table.

- Model 2 TF-IDF approach: this model did not consider the words frequency alone but computed the product of the term frequency and the inverse document frequency.

- Model 3 Cosine Similarity: this model is based on cosine similarities between the sentences, and instead of considering the magnitude, the cosine similarities consider rather the orientation.

- Model 4 TextRank Algorithm: this model is based on a graph generated from a cosine similarity matrix built using sentence embeddings.

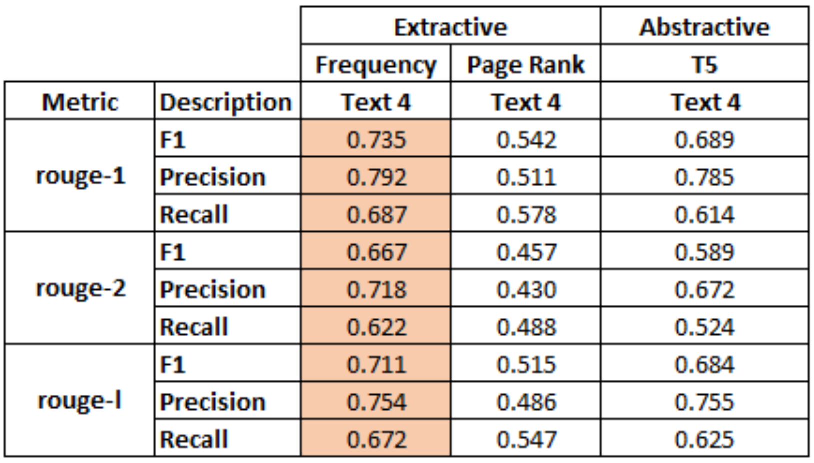


		Extractive			
Metric	**Description**	**Frecuency**	**TF-IDF**	**Cosine Similarity**	**Page Rank**
rouge-1	F1	0.476	0.335	0.335	0.588
	Precision	0.634	0.524	0.524	0.634
	Recall	0.381	0.246	0.246	0.549
rouge-2	F1	0.342	0.143	0.143	0.470
	Precision	0.456	0.224	0.224	0.506
	Recall	0.273	0.105	0.105	0.438
rouge-l	F1	0.484	0.287	0.287	0.571
	Precision	0.583	0.385	0.385	0.580
	Recall	0.414	0.228	0.228	0.562

Image 17: Extractive Summarizer Model Comparison in Text

The Text Rank Algorithm outperformed all the extractive summarization models. The main reason, we believe, is due to the use of sentence embeddings.

Comparison with the T5 Abstractive



		Extractive		Abstractive
		Frequency	Page Rank	T5
Metric	**Description**	**Text 4**	**Text 4**	**Text 4**
rouge-1	F1	0.735	0.542	0.689
	Precision	0.792	0.511	0.785
	Recall	0.687	0.578	0.614
rouge-2	F1	0.667	0.457	0.589
	Precision	0.718	0.430	0.672
	Recall	0.622	0.488	0.524
rouge-l	F1	0.711	0.515	0.684
	Precision	0.754	0.486	0.755
	Recall	0.672	0.547	0.625

Image 18: Extractive and Abstractive Model Comparison in Text 4

To be able to compare the performance of our text summarizers, we also run the T5 abstractive summarizer to see if our approach was solid. By comparing these 3 models (2 extractive and 1 abstractive), we see that the frequency and T5 model are better than Text Rank for text 4, but this comparison is not an even evaluation because T5 abstractive summarization con generate additional vocabulary compared to the traditional extractive ones.

As a final attempt to take the text summarizers to a further step, we decided to combine our two models, by treating the output of our Text Rank summarizer as the input for the T5 abstractive summarizer to see the result.

An abstractive model seems to perform better on the original text rather than processed text coming from the TextRank model. This would be explained by the fact that the Abstractive model would better assess the text given a broader context and the output would be better built.

Probably the best way to combine the 2 models would be to use them together rather than combining them. For instance, the abstractive model could be used as a general overview with a good wording while the extractor one would be used for a more in-depth analysis.

Final Model Selection

We made the decision to go with the **Text Rank model** due to the following reasons:

The model is an easy one to build which offered a better performance than Models 1,2,3 in terms of the ROUGE metric. When compared to the other models, this model offered a better readability when it comes to the output. It considers semantically similar words by using the embeddings approach.

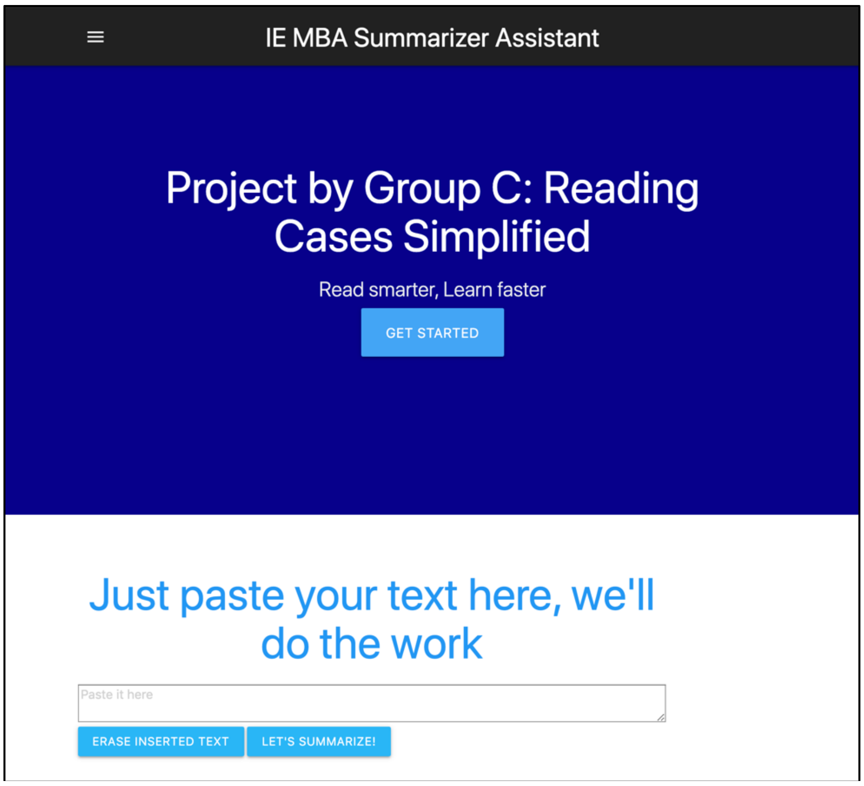
It can scale to big texts and not only give a brief summary. As noted above, the T5 abstractive model is limited in terms of the length of the output. When we tried it with a lengthy text, it summarized it accurately in just 2-3 sentences but did not give more than 70 words. This could be an advantage when we have short texts; but when talking about lengthy business cases we believe a 10-20 sentences summary would be better, which is what our model has been able to accurately give us.

As also explained, abstractive summarizers are harder to build and require an advanced understanding of Neural Networks. Even implementing a pre-trained model like the T5 took a long time to set up and download. Given all those reasons and knowing we wanted to implement our model into a Web Application, the choice of the TextRank algorithm seemed the most appropriate.

Application Development __

As mentioned in the introduction, we used Flask to develop our web application. We followed a tutorial (*Building a Text Summarizer Flask App with SpaCy,NLTK,Gensim & Sumy* by JCharisTech & J-Secur1ty) referenced in the application and were able to include in the code of the TextRank model we have built in the script. To be able to rerun our Application code, you should download the Glove embeddings ('glove.6B.100d.txt') and add them to

the files because we could not send it as it is a very large file. We also used a miniature of a TextRank library available in Python, the Sumy library for the small Overview section in our application.
Like demonstrated in our presentation, the final interface looks like this:


IE MBA Summarizer Assistant

Project by Group C: Reading Cases Simplified

Read smarter, Learn faster

GET STARTED

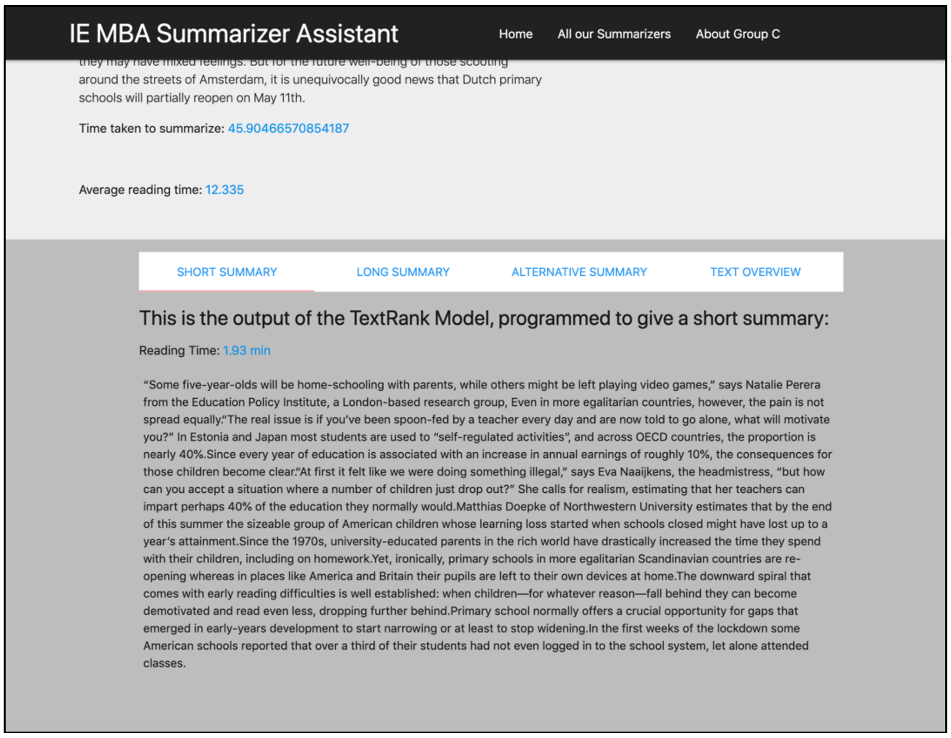
Just paste your text here, we'll do the work

Paste it here

ERASE INSERTED TEXT

LET'S SUMMARIZE!

The other window of the application gave access to different models:


IE MBA Summarizer Assistant

Home

All our Summarizers

About Group C

they may have mixed feelings. But for the future well-being of those scooting around the streets of Amsterdam, it is unequivocally good news that Dutch primary schools will partially reopen on May 11th.

Time taken to summarize: 45.90466570854187

Average reading time: 12.335

SHORT SUMMARY

LONG SUMMARY

ALTERNATIVE SUMMARY

TEXT OVERVIEW

This is the output of the TextRank Model, programmed to give a short summary:

Reading Time: 1.93 min

“Some five-year-olds will be home-schooling with parents, while others might be left playing video games,” says Natalie Perera from the Education Policy Institute, a London-based research group, Even in more egalitarian countries, however, the pain is not spread equally.“The real issue is if you’ve been spoon-fed by a teacher every day and are now told to go alone, what will motivate you?” In Estonia and Japan most students are used to “self-regulated activities”, and across OECD countries, the proportion is nearly 40%.Since every year of education is associated with an increase in annual earnings of roughly 10%, the consequences for those children become clear.“At first it felt like we were doing something illegal,” says Eva Naaijkens, the headmistress, “but how can you accept a situation where a number of children just drop out?” She calls for realism, estimating that her teachers can impart perhaps 40% of the education they normally would.Matthias Doepke of Northwestern University estimates that by the end of this summer the sizeable group of American children whose learning loss started when schools closed might have lost up to a year’s attainment.Since the 1970s, university-educated parents in the rich world have drastically increased the time they spend with their children, including on homework.Yet, ironically, primary schools in more egalitarian Scandinavian countries are re-opening whereas in places like America and Britain their pupils are left to their own devices at home.The downward spiral that comes with early reading difficulties is well established: when children—for whatever reason—fall behind they can become demotivated and read even less, dropping further behind.Primary school normally offers a crucial opportunity for gaps that emerged in early-years development to start narrowing or at least to stop widening.In the first weeks of the lockdown some American schools reported that over a third of their students had not even logged in to the school system, let alone attended classes.

The “Short Summary” can be found in the “TextRank_our_model.py” script and is the implementation of the TextRank algorithm described in the notebook and programmed to return 7 sentences. The “Long Summary” can be found in the “TextRank_our_model_2.py” script and is also the implementation of the TextRank algorithm described in the notebook and programmed to return 10 sentences. The “Alternative Summary” is the M1: Basic Model also explained in the script and is put here to provide an alternative way of summarizing in case the other models did not give a proper result. And the “Text Overview” tab is the Sumy library mentioned above, implementing a smaller version of the TextRank algorithm.

Our plan was to not only stop at text summarization in the application but continue and build on the output we have. We wanted to implement a feature that would use the keywords of our texts to either be able to answer questions on them or provide links for further information based on pieced of information in the studied text.
These were not implemented due to time constraint, but we plan on continue working on this project.

Note on other Use Cases

There are many different applications of this technique. Text Summarization is a very relevant topic in the recent years due to the evolution of new models to optimize the way texts are summarized. We see so many different business sectors benefiting from this NLP technique:

A. Social Media Marketing: Companies produce long-form content, such as whitepapers, e-books and blogs, and might be able to leverage by using summarization to break down this content and make it sharable on social media sites like Twitter or Facebook.

B. Legal contract analysis: Large companies are constantly producing internal documentation, which frequently gets stored and under-used in databases as unstructured data. Summarization in big law firms can enable to quickly understand everything the company has already done in their case history, and quickly develop better strategies. Also, text summarizer adds value by condensing a contract to the riskier clauses, or to compare agreements.

C. Question answering and bots: With the increasing performance of personal assistants (Siri, Alexa), large-scale summarization could become a powerful question answering technique, instead of the actual approach of specific tasks. By collecting the most relevant documents for a question, a summarizer assembles a cohesive answer in the form of a multi-document summary.

D. Science and R&D: Academic papers typically include a human-made abstract is the summary of the document. However, when conducting research, it can become overwhelming to read every abstract. Text summarizer systems can process multiple input papers and compress abstracts to reduce even more the consultation time of documents.

References

• ALGULIEV, Rasim, and Ramiz ALIGULIYEV. Evolutionary Algorithm for Extractive Text Summarization . 2009.

• “An Intro to ROUGE, and How to Use It to Evaluate Summaries.” FreeCodeCamp.Org, 26 Jan. 2017, https://www.freecodecamp.org/news/what-is-rouge-and-how-it-works-for-evaluation-of-summaries-e059fb8ac840/.

• BBC News Summary. https://kaggle.com/pariza/bbc-news-summary. Accessed 3 May 2020.

• “Comprehensive Guide to Text Summarization Using Deep Learning in Python.” Analytics Vidhya, 10 June 2019, https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/.

•“Exploring Transfer Learning with T5: The Text-To-Text Transfer Transformer.” Google AI Blog, http://ai.googleblog.com/2020/02/exploring-transfer-learning-with-t5.html. Accessed 3 May 2020.

• Goutham, Ramsri. “Simple Abstractive Text Summarization with Pretrained T5 — Text-To-Text Transfer Transformer.” Medium, 17 Apr. 2020, https://towardsdatascience.com/simple-abstractive-text-summarization-with-pretrained-t5-text-to-text-transfer-transformer-10f6d602c426.

• Huong Thanh , Le, and Le Tien Manh. An Approach to Abstractive Text Summarization . 2016.

• jcharistech. “Text Summarization Using SpaCy and Python.” JCharisTech, 31 Dec. 2018, https://jcharistech.wordpress.com/2018/12/31/text-summarization-using-spacy-and-python/.

• Machine Learning :: Cosine Similarity for Vector Space Models (Part III) | Terra Incognita. http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/. Accessed 3 May 2020.

• Panchal, Akash. “Text Summarization in 5 Steps Using NLTK.” Medium, 23 Feb. 2020, https://becominghuman.ai/text-summarization-in-5-steps-using-nltk-65b21e352b65.

• “Text Summarization Using TF-IDF.” Medium, 19 Jan. 2020, https://towardsdatascience.com/text-summarization-using-tf-idf-e64a0644ace3.

• Ratia, Tomas. “20 Applications of Automatic Summarization in the Enterprise.” The Frase Blog, 17 July 2018, https://blog.frase.io/20-applications-of-automatic-summarization-in-the-enterprise/.

• Santopietro, Vincenzo. “Diving into Abstractive Text Summarization — Part 1.” Medium, 14 Feb. 2019, https://medium.com/intel-student-ambassadors/diving-into-abstractive-text-summarization-part-1-e8570d370021.

• Sharma, Himanshu. “Text Summarization in Python With SpaCy Library.” Present Slide, 8 Aug. 2019, https://www.presentslide.in/2019/08/text-summarization-python-spacy-library.html.

• Singhal, Ashish. “NLP: Building Text Summarizer — Part 1.” Medium, 18 Nov. 2019, https://medium.com/datapy-ai/nlp-building-text-summarizer-part-1-902fec337b81.

• “NLP: Building Text Summarizer — Part 2.” Medium, 18 Nov. 2019, https://medium.com/datapy-ai/nlp-building-text-summarizer-part-2-db9204dea8bf.

• Vanetik, Natalia, and Marina Litvak. Query-BasedsummarizationusingMDLprinciple. 2013.

• Yogan Jaya , Kumar, et al. “A Review on Automatic Text Summarization Approaches .” 03.24.2016.