Capstone Project: Text Summarization using Machine Learning techniques

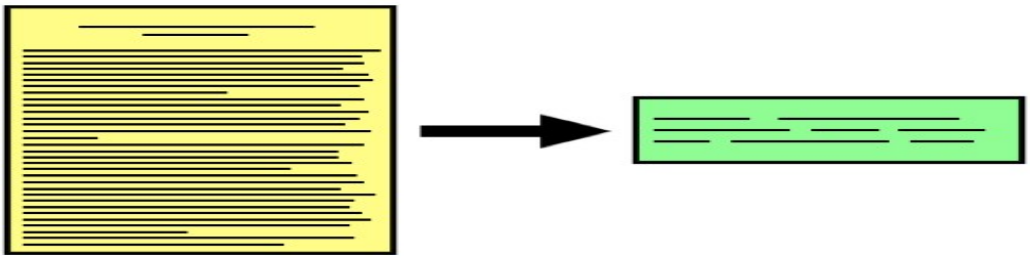
I. Definition

Project Overview

With the rise of information technologies, globalization and Internet, an enormous amount of information is created daily, including a large volume of written texts. The International Data Corporation (IDC) projects that the total amount of digital data circulating annually around the world would sprout from 4.4 zettabytes in 2013 to hit 180 zettabytes in 2025 [1]. Dealing with such a huge amount of data is a current problem where automatization techniques can help many industries and businesses.

For example, hundreds or thousands of news are published around the world in a few hours and people do not want to read a full article for ten minutes, So, the development of automatic techniques to get short, concise and understandable summaries would be of great help to many global companies and organizations.
Another use case is social media monitoring, many companies or organizations need to be notified when tweets about their products or brand are mentioned to prepare an appropriate and quick response to them. Other fields of interest are legal contract analysis, question answering and bots, etc.

Our main goal in this project is to analyze and build a text summarizer using some basics techniques based on machine learning algorithms. Given a long and descriptive text about some topic we will create a brief and understandable summary covering that same topic. As any other supervised techniques, we will use a dataset containing pairs of texts and summaries. There are some public datasets we can use like the CNN / Daily Mail dataset, the Gigaword dataset or the one we will mainly cover, a news summary from the Kaggle website [2]. This dataset contains news extracted from Inshorts, that publish many news articles from Hindu, Indian times and Guardian. We will describe this data extensively in a following chapter.



Source: https://sflscientific.com/data-science-blog/2016/11/17/text-summarization-in-natural-language-processing

Problem Statement

Text Summarization is a challenging problem these days and it can be defined as a technique of shortening a long piece of text to create a coherent and fluent short summary having only the main points in the document.
But, what is a summary? It is a "*text that is produced from one or more texts, that contains a significant portion of the information in the original text(s), and that is no*

longer than half of the original text(s) [3]. *Summarization clearly involves both these still poorly understood processes, and adds a third (condensation, abstraction, generalization)".* At this moment is a very active field of research and the state-of-the-art solutions are still not so successful than we could expect.

The problem, we are facing in this project, is to produce a summary (about 20-word length) from a single document much longer, 200-400 words. It will be a single document summarization, just one document will be summarize at a time, and the output could be of type *extractive* (containing pieces of the source text) or *abstractive* (a new text is created) from the original text.
We are interested in analyzing and compare the results of an extractive solution versus an abstractive one, given a benchmark, measure the quality of the outputs and identify strengths and weaknesses to highlight some conclusions.

The steps we are taking to meet our goal are:

- Decompose and analyze the data
- Identify, process and prepared the data to feed the model
- Build and train a summarizer using an extractive approach (using sentence clustering)
- Model and train an abstractive algorithm using a sequence to sequence model, recurrent neural network.
- Evaluate the results from both solutions and conclude by pointing out some possible improvements

Metrics

To evaluate the model, we will use the ROUGE metric:

"ROUGE, or Recall-Oriented Understudy for Gisting Evaluation, is a set of metrics and a software package used for evaluating automatic summarization and machine translation software in natural language processing. The metrics compare an automatically produced summary or translation against a reference or a set of references (human-produced) summary or translation.".[4]

This metric produces a value for recall (how much of the "real" summary is captured by the "predicted" summary) and precision (how much of the predicted summary is relevant). Both provide us a measure of how good our model output is. There are some variations of this metric: ROUGE-N, ROUGE-S and ROUGE-L For example, ROUGE-1 refers to overlap of unigrams between the system summary and reference summary. ROUGE-2 refers to the overlap of bigrams between the system and reference summaries [5].

We will consider ROUGE-2 and ROUGE-L metrics, measure the benchmark model, and then compare the results of our both algorithms to get an evaluation score. But we are also going to show and compare the outputs "visually", using a "human readable" measure.

II. Analysis

Data exploration and visualization

When searching for information and data about text summarization we found hard to obtain a "good" dataset. Some of the most popular data sets are intended for research use, containing hundreds of thousands of examples and gigabytes of data that require high computational capacity and days or weeks to train. But we are interested in a

dataset that could be trained faster, in a few hours, where we can experiment and develop easily.
We will use a dataset from Kaggle, [2] that contains to set of examples. One that consists in 4515 examples of news and their summaries and some extra data like *Author_name, Headlines, Url of Article, Short text, Complete Article*. This data was extracted from Inshorts, scraping the news article from Hindu, Indian times and Guardian. And another one including 98,000 rows containing shorter source texts and summary.
An example:

- Text: *"Isha Ghosh, an 81-year-old member of Bharat Scouts and Guides (BSG), has been imparting physical and mental training to schoolchildren ..."*
- Summary: *"81-yr-old woman conducts physical training in J'khand schools"*

This type of problem only focuses on two variables of textual type: the text of the article and the summary. Other variables are discarded because they are not used to create a summary from a text.

To analyze the composition and characteristics of the texts we create a bunch of new features:

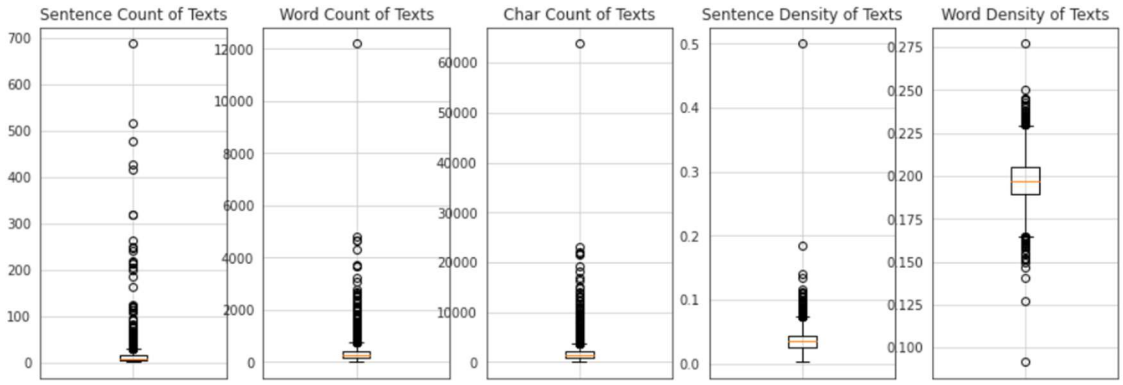
- Sentence Count - Total number of sentences in the text
- Word Count - Total number of words in the text
- Character Count - Total number of characters in the text excluding spaces
- Sentence density - Number of sentences versus the number of words
- Word Density - Number of words versus the number of characters
- Punctuation Count - Total number of punctuations
- Stopwords Count - Total number of common stopwords in the text

We start our analysis on the text variable, using the new features we can obtain the following descriptive statistics:

	text_sent_count	text_word_count	text_char_count	text_stopw_count	text_punc_count
count	4341.000000	4341.000000	4341.000000	4341.000000	4341.000000
mean	12.970974	337.573140	1713.334716	132.756047	58.908546
std	23.600000	343.945806	1754.699138	136.731540	81.742324
min	1.000000	1.000000	3.000000	0.000000	0.000000
25%	5.000000	185.000000	935.000000	68.000000	28.000000
50%	9.000000	281.000000	1429.000000	109.000000	44.000000
75%	15.000000	407.000000	2060.000000	162.000000	68.000000
max	687.000000	12202.000000	63845.000000	4498.000000	1889.000000

- 12 sentences, 337 words and 1713 characters per text are the mean values
- But we observe that the maximum number of sentences (687) or chars (more than 63,000) are far away from the mean values, indicating that there are some registers with values out of range or outliers.

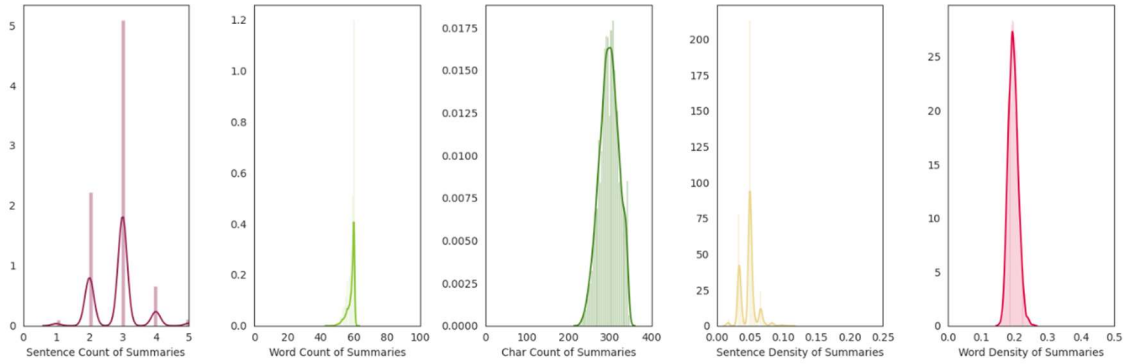
In the next figure we can check the presence of outliers, some points with extremely high values.



Now we repeat the same step using the summary variable, showing the following results:

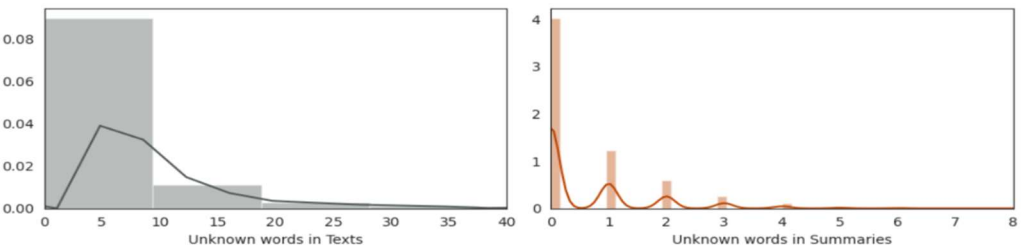
	sum_word_count	sum_char_count	sum_sent_count	sum_sent_density	sum_word_density
count	4341.000000	4341.000000	4341.000000	4341.000000	4341.000000
mean	58.297858	297.426860	2.824234	0.047648	0.196331
std	2.304367	23.049156	0.673544	0.011304	0.014742
min	44.000000	227.000000	1.000000	0.016393	0.152738
25%	57.000000	282.000000	2.000000	0.036364	0.185759
50%	59.000000	298.000000	3.000000	0.049180	0.195440
75%	60.000000	314.000000	3.000000	0.050847	0.205674
max	62.000000	346.000000	6.000000	0.111111	0.258772

We can observe that the summary variable is equally distributed or at least there are just a few outliers. Therefore, we can infer that the distributions will be less skewed:



Half of the summaries have around 60 words and 2 or 3 sentences but unlike with the text variable, the number of words and sentences stays around central values, no outliers.

Another topic to evaluate is **the presence of unknown words** in our texts. These words could have a huge impact in our experiments so we will count them and study how frequent they are:



- In our source texts the mean is 5 and 70% of the sentences are below 5. *About 8% of the words* in the texts are unknown it is high enough to be very careful on its treatment.
- The figures are 0.5 and 1 in the case of the summary variable, they are not relevant.

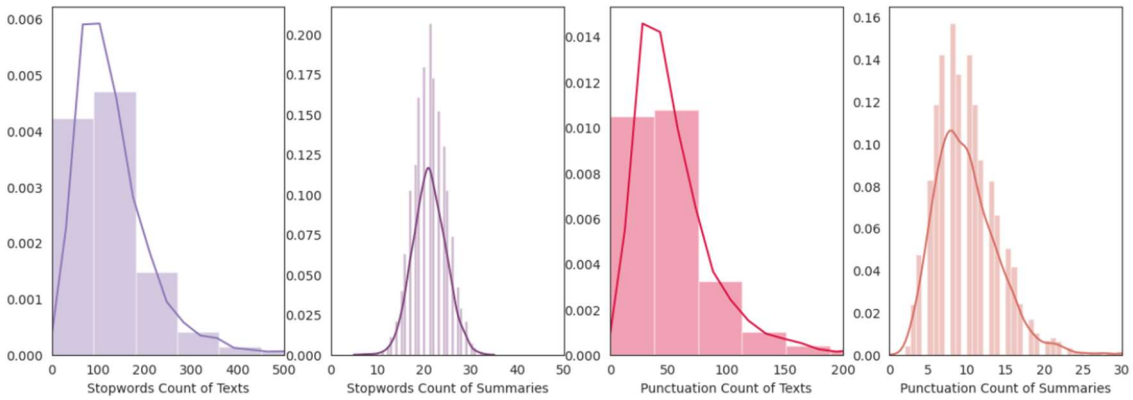
Most of the unknown words are names, surnames, or misspelled words.

A **Wordcloud** (or Tag cloud) is a visual representation of text data. It displays a list of words, the importance of each beeing shown with font size or color (the bigger the more frequent). This format is useful for quickly perceiving the most relevant terms on a document or set of documents.

Wordcloud of Text variable | Wordcloud of Summary Variable

As it could be expected they both contain almost the same words like said, people, india, film, asked, year. Except *goverment* and *time* that are very frequent in the source text while added and reportedly appears a lot of times in the summary.

Other topic we need to cover is the presence of **stop-words and punctuation** in the texts, they are usually removed in natural language processing tasks because they do not provide any useful information. We plot the histograms for both variables:

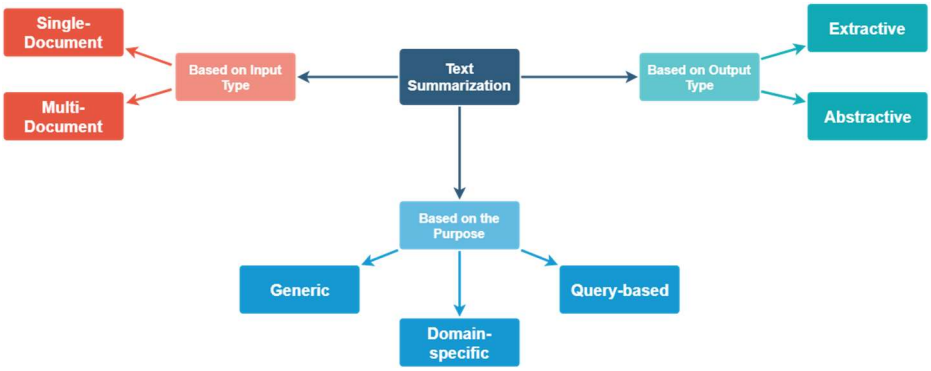


Both plots show a behavior similar to the previous ones, the text variable is a right-skewed distribution with a very long tail, but the shape of the summary distribution looks like a normal or gaussian one. Almost half of the words in the text variable are stop words or punctuation.

Algorithms y techniques

"Text summarization is the process of distilling the most important information from a source (or sources) to produce an abridged version for a particular user (or users) and task (or tasks)." [7]

We can classify the text summarization methods based on various aspects:



We will analyze single-document, domain specific (news articles) based on output type extractive and abstractive:

- Extractive — These approaches select sentences from the corpus that best represent it and arrange them to form a summary.
- Abstractive — These approaches use natural language techniques to summarize a text using novel sentences.

Our **extractive approach** technique will apply an unsupervised clustering algorithm on the sentences previously embedded using pretrained vectors. Now we describe the main steps involved:

1. **Data preprocess and cleaning**.
2. **Sentence embedding using pretrained vectors**

Word embeddings are a type of word representation that allows words with similar meaning to have a similar representation. Word embeddings are in fact a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned based on the usage of words. This allows words that are used in similar ways to result in having similar representations, naturally capturing their meaning. A well-trained set of word vectors will place similar words close to each other in that space. The words oak, elm and birch might cluster in one corner, while war, conflict and strife huddle together in another.

GloVe, algorithm is an extension to the word2vec method for efficiently learning word vectors, is an unsupervised learning algorithm that constructs an explicit word-context or word co-occurrence matrix using statistics across the whole text corpus.[9]

3. **Sentence clustering**

Once the sentences are embedded, we apply a K-Means clustering method to group the sentences in k clusters. Each cluster of sentence embeddings can be interpreted as a set of semantically similar sentences whose meaning can be expressed by just one candidate sentence in the summary.

4. **Select the most representative sentence of each cluster**

The candidate sentence chosen in every cluster is the one whose vector representation is closest to the cluster center.

5. **Ordering the sentences to form the summary**

Candidate sentences corresponding to each cluster are then ordered to form a summary. The order of the candidate sentences in the summary is determined by the positions of the sentences in their corresponding clusters in the original document

Our **abstractive approach** is based on a machine translation model, using an encoder-decoder with attention that has become popular for summarization.

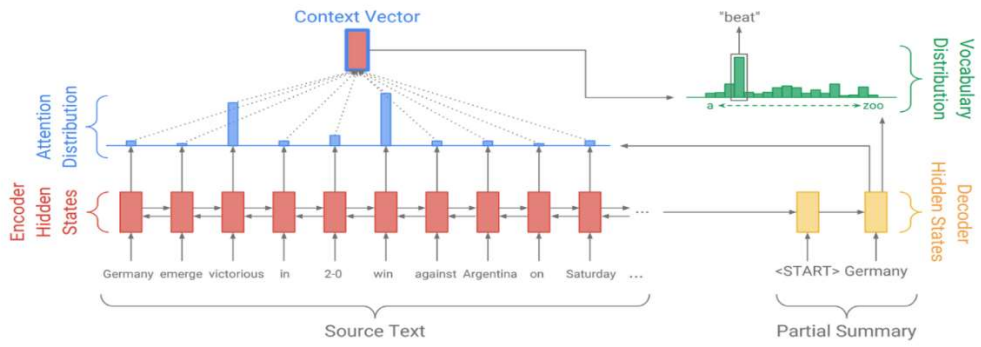


Figure 1 Sequence2Sequence attention model

The encoder-decoder model for recurrent neural networks is an architecture for sequence-to-sequence prediction problems. It comprised two parts:

- Encoder: The encoder is responsible for stepping through the input time steps, read the input words one by one and encoding the entire sequence into a fixed length vector called a context vector.
- Decoder: The decoder is responsible for stepping through the output time steps while reading from the context vector, extracting the words one by one.[10]

The trouble with seq2seq is that the only information that the decoder receives from the encoder is the last encoder hidden state which is like a numerical summary of an input sequence. So, for a long input text, we expect the decoder to use just this one vector representation to output a translation. This might lead to catastrophic forgetting.

To solve this problem, the attention mechanism was developed. Attention is proposed as a method to both align and translate. It identifies which parts of the input sequence are relevant to each word in the output (alignment) and use that relevant information to select the right output (translation). So instead of encoding the input sequence into a single fixed context vector (reason for the mentioned bad performance), the attention model develops a context vector that is filtered specifically for each output time step [11]. Attention provides the decoder with information from every encoder hidden state. With this setting, the model can selectively focus on useful parts of the input sequence and hence, learn the alignment between them.

We will apply **Teacher forcing**, a method for quickly and efficiently training recurrent neural network models that use the ground truth from a prior time step as input, that is, using the actual or expected output from the training dataset at the current time step y(t) as input in the next time step X(t+1), rather than the output generated by the network

Benchmark

The purpose of this project is not to define a state-of-the-art solution to this problem because it would require a long research and a very large dataset to train for many days. But we want our model to be compared to a baseline model, a module for summarization from the genism library. The gensim implementation is based on the popular “TextRank” algorithm [13].
To apply TextRank, we first need to build a graph associated with the text, a vertex is added to the graph for each sentence in the text and finally every vertex is scored based on the “vote” from its linked vertex.[15]

It is good enough to set a level of performance to compare with.

III. Methodology

Data preprocessing

Once we have analyzed the content of our text variables, we have identified some preprocess task needed to be applied and, based on other NLP problems, we will apply some other common cleanings techniques.

As we saw in the data exploration section, first we need to **remove examples where the text variable is null or duplicated** from our dataset. Then we select the two variables of interest: ctext as the text variable and text as the summary variable. Now we have a dataset containing two variables, the Text and Summary columns.

Now we can apply the following task:

1. Expand contractions
2. Remove URL and HTML tags.
3. Remove non-character symbol.
   Exploring the dataset we find some "special" characters that we need to remove
4. Remove the line break character
5. Remove hashtags and mentions.
6. Remove multiple space.
   We find very often the presence of more than one space between words, so we will remove them and keep just one space.
7. Remove punctuation
   We saw in our exploratory analysis that there were many punctuations and some of them in a wrong position. These symbols are usually removed because they do not provide interesting information, but when working in language modelling tasks they are necessary. We will remove them.

Now, we will tokenize our texts to apply other preprocessing jobs. *"**Tokenization**: Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called tokens, perhaps at the same time throwing away certain characters, such as punctuation."* [15]. Here is an example of tokenization:

Input: Friends, Romans, Countrymen, lend me your ears;

Output: | Friends | Romans | Countrymen | lend | me | your | ears |

A token is our useful semantic unit for processing and every sentence in our documents will be splitted in tokens using the NLTK library. Next, we will go through each token of our vocabulary and we will apply the following transformations:

1. Lowercase
2. Remove digits
3. Remove tokens with length less than 2
4. Remove stop words
   Some extremely common words which would appear to be of little value in our task. We use the stop word list for English included in the NLTK library.
5. Apply steaming or lemmatization
   The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.[15]. We prefer to apply steaming to avoid the presence of non-actual words that stemming usually produce. *If confronted with the token saw, stemming might return just s, whereas lemmatization would attempt to return either see or saw depending on whether the use of the token was as a verb or a noun* [15]

At the end of this cleaning pipeline we have a cleaning text and summary ready to feed our machine learning models, but previously we need to split our dataset in two group of examples:

- The **training dataset**: Contains the examples used to train the model, whose features will be learnt.
- The **validation or test set**: the model will be evaluated o these examples to get an score or accuracy mark.

When working with machine learning algorithm this task is very important, we want to make sure that the test set simulates as much as possible the real or productive environment where the inputs to the algorithm have never been seen by the model. Therefore, we must separate our set of examples and we will use the Sci-kit Learn library, defining a train set containing 80% of the examples, and a 20% in the test set. Finally, we save both dataset in different filename that our algorithm will load and work on them.

Implementation

In the development of this project we worked on the Google Cloud Platform, storing the data on Google Cloud Storage buckets (GCS), and running the code on AI-Platform notebooks. We also have developed and trained our abstractive model on Kaggle notebooks, to get access to GPU resources.

Sentence Clustering

This is an unsupervised algorithm and the output of every example is independent from any other examples, so there is no real "training". We take every document, apply the steps, and create a summary. It does not matter what the texts of the previous or following examples are.

We apply the following method:

- Install and import the libraries: We move the data from GCS to local disk and import the libraries.
- Loading the train and validation datasets: Load the data on pandas dataframes.
- Loading the embedding vectors.
  Rather than training our own word vectors from scratch, we will leverage on GloVe, as we discussed previously, we will use "Wikipedia 2014 + Gigaword 5" trained on a corpus of 6 billion tokens and contains a vocabulary of 400 thousand tokens. The version we have chosen is *"glove.6B.100d.txt"* so our vocabulary contains 400,000 words of a hundred values, dimensions are 400Kx100.
  We create a Python Class to load the embeddings vector, store the vocabulary and provides a function to transform a sentence into an embedded vector.
- Sentence embeddings
  Now, we have our word representation, a vector for every word in our vocabulary. But we need to deal with full sentences so we need to create a sentence embedding, basically we need a vector that represent the whole sentence and every feature in the vector will be based on the subjacent word embeddings. There are many possibilities, but we apply a very simple method: the ith value in the sentence embedding will be the mean of the ith values in the word embedding of all the words in the sentence.
  Given a document we transform each of its sentences into an embedding vector.
- Clustering the embedded sentences
  Once all the sentences in the document are vectors, we can apply the k-means clustering algorithm (grouping similar vectors/sentences together because of

certain similarities). We set the parameter *k*, number of clusters, to the square root of the number of sentences that correspond to the number of sentences desired.
- Select the candidate sentence in every cluster
  Now, for every cluster we select one sentence, the closest to the cluster centroid.
- Order the selected sentences.
  Finally, we order the clusters based on the mean of the positions of their sentences in the original document and select the summary sentences in that order.
- Evaluate the results.
  Using the rouge library, we calculate the rouge metrics for very summary predicted, but this will be extensively described in a later chapter.

Sequence-to-Sequence model

A sequence-to-sequence encoder-decoder model consist in an encoder that receives the input data and produce a vector representation of that input and a decoder that receive the output from the encoder and produce the next word or token for the output sequence. We will try to explain briefly how it has been implemented in our exercise.
- Create a Python Class to store the vocabulary and tokenize the text.
  We create an input vocabulary containing every token in the input dataset, transform it to an integer value and stores this mapping. Then every text in the training data its transform into a vector of integers.
  We repeat the process with the validation dataset, defining a new vocabulary on it.
- Identify the maximum length of an input or output sequence
  We go through all the records of the training set, calculate the length in tokens and finally establish the maximum value of the length of a sequence

Encoder

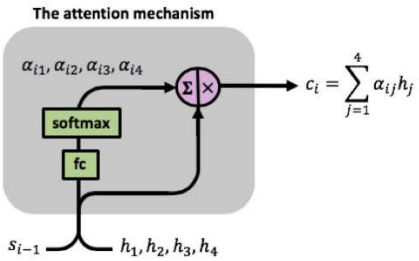
We define an encoder model, as a Class called EncoderRNN, including:
An embedding layer: it will learn an embedding for all the words in the training dataset. Receive an input, of input size, and convert it to its embedding, vector representation. The output is of size equals to the hidden layer size (hidden_size). In this model we are not going to use pre-trained vectors, like Glove, this layer is initialized with random weights.

A GRU layer: Gated Recurrent Unit is an RNN created to solve the short-term memory problem that RNNs suffer. They use an update gate to *"decide what information to throw away and what new information to add"* [16] and a reset gate to *"decide how much past information to forget"*. The input size corresponds to the output size of the embedding layer, hidden_size, and the size of the hidden state is defined as hidden_size too. Only create a single layer.

The encoder returns the output from the last layer of size hidden_size for each time step and the hidden state of the last time step.
For a more detailed explanation on how LSTM and GRU works check the article [16].

Decoder

The Class AttDecoderRNN encapsulates our decoder component:
An embedding layer: It will create the embeddings for the output vocabulary. Generates an output of size hidden_size.
Attention mechanism: first, calculate the *attention weights* with a fully-connected layer that receives the concatenation of the decoder states in the previous time steps and the encoder outputs and the output is passed through a softmax function computing the attention weights. Second, generate the *context vectors*, a weighted sum of the encoder outputs and the attention weights.[17]



A RELU activation receives the output from the attention cell (context vectors) and return an output of the same size (hidden_size) after applying the element-wise operation, max(0,x).
A GRU layer: like the previous one described. Receives the output from the Relu activation (hidden_size) and returns the outputs from the last layer and the hidden state.
A linear dense layer: transform the output from the GRU in the last time step to an output of size output_size, the size of the output vocabulary.
A LogSoftmax layer: receives the output from the linear layer and returns a vector of the same size that the elements lie in the range [-inf,1].

During training, *we apply teacher forcing with a probability of 0.5*. So, half of the times the expected output from the training in the current time becomes the input to the decoder in the next time step.[18]
Later, in the Refinement section, we will announce that due to the poor results obtained, **we opted for a new implementation**.

Refinement

In this section we will describe, for both models developed, how we try to improve their performance in different "parameters" or methods.

Sentence Clustering

This algorithm is an unsupervised technique that, as we said previously, it does not require a training phase but there are some points we can study to get more optimized results.

- Word representation or **embeddings** of the text
  First, we try to build our own vector representation for our "language", the texts we are working with, using the fasttext library. FastText is an extension to Word2Vec proposed by Facebook in 2016, Instead of feeding individual words into the Neural Network, FastText breaks words into several n-grams and after training we will have the embeddings for all the n-grams in the train dataset. This methodology allows rare words to be represented appropriately but it may not generalize as well as the Glove vectors because it is trained only on our dataset. And it takes a long time to train on a large dataset.
  Finally, we decided to use the glove vectors to obtain a better generalization capacity.

- Sentence embeddings
  In our problem a sentence consists in a group of vectors of number (every word is a vector) and we need to generate the vector of the whole sentence. To get this vector we study the use of the Euclidean distance between the words and then obtain the center point, but we discard this option because it does not provide any noticeable difference from obtaining the mean. It would only increase the computing cost to get just a linear combination of the data.

The use of the mean values of the vectors is good enough to measure the distances between sentences.

Sequence-to-Sequence model

This approach has many points or parameters to deal with, but the initial results confirm us that we need more than just a parameter tuning. In the next section we will discuss the results in detail, but for now just mention that the outputs from our initial model contains a lot of repeated words providing poor summaries. Then we decided to research for an optimized model, and we found a well-documented solution**, a Pointer Generation network**.

Based on the paper *"Get To The Point: Summarization with Pointer-Generator Networks"* [19], Pointer generator networks solve the repetition problem by calculating "generating probability" which represents the probability of generating a word from the vocabulary versus copying the word from the source. It is actually a hybrid network, a combination approach combining both extraction (pointing) and abstraction (generating).

We calculate an *attention distribution* and a *vocabulary distribution*. However, we also calculate the *generation probability*, which is a scalar value between 0 and 1. This represents the probability of *generating* a word from the vocabulary, versus *copying* a word from the source.

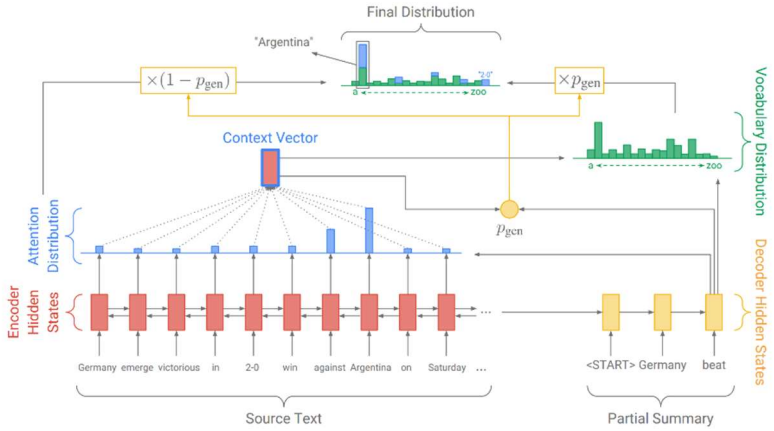


Figure 2. Diagram showing the Final Distribution

Later, we will show the results and performance of this approach.

We need to mention that we also tuned manually the most common hyperparameters in this kind of models like the hidden size of the encoder and decoder, the batch size and the learning rate and also the optimizer (SGD or Adam). Because this model takes too much time to train, we use a subsample of the training and validation dataset to tune the hyperparameters and once we obtain a good combination, we apply it on the whole dataset in our final implementation

IV. Results

Model Evaluation and Validation

After analyzing the data, designing a solution, and developing the models, it is time to measure the results and evaluate the performance of the algorithms implemented. Then we will compare them with our benchmark and analyze how good they are against different kind of inputs.

We will start showing the values of the defined metrics on the validation dataset. We are using the dataset containing a short summary, one sentence with about 15 words,

because encoder-decoder architecture requires a lot of time to train and produce large outputs and it would be difficult to compare the results:

	rouge2-f	rouge2-p	rouge2-r	rougel-f	rougel-p	rougel-r
gensim	0.113189	0.115086	0.143432	0.230512	0.226352	0.287895
sentence_clustering	0.123572	0.138182	0.137387	0.261064	0.279020	0.287121
seq2seq_model	0.043019	0.049889	0.038690	0.268255	0.477630	0.196017
pointer_generator	0.124713	0.088080	0.193559	0.339516	0.244984	0.442852

Rouge metrics f: F-score, p: Precision and r: recall

Values for Rouge2 metrics seems to be very poor in the abstractive approaches, which mean that on average there is a very low overlap of 2-grams between our summaries and the references. Another noteworthy observation is that for the extractive method, the differences between f-score, precision and recall are very small, so the "stability" of the results in this method is higher than for the abstractive approach. In the case of the pointer generator we can observe that the Rouge-L recall is 44%, meaning that 44% of the 2-grams in the reference summary are also present in the generated summary but only 24% of the 2-grams in the generated summary are also present in the reference summary.

Using the Rouge-L F-score as the selected metric, the pointer generator reaches the best score but the extractive approach to the problem seems to be an appropriate solution, because it provides high values on the metrics and it is much faster than any other abstractive solution.

A better perspective of the results can be obtained by visually examining the texts generated for each case. We will print some examples and the summaries returned by every model to evaluate the results by ourselves. First, we will explore examples with a very high Rouge-L f-score:

	summary	pred_summary	rougel-f
Clustering	government seek data deposits made individual demonetisation period part income tax returns assessment year 201718. according reports government introduced new column obtaining details deposits . ... effort ascertain deposits made demonetisation period official said .	taxman seek data deposits made assessee demonetisation period part new income tax returns itrs would notified soon assessment year 201718. officials said tax department understood introduced new column obtaining details deposits made individual entity demonetisation period november 8 december 30 last year .	0.63
Seq-2-Seq	flipkart parent invests 1431 crore india wholesale arm	flipkart invests flipkart crore india	0.66
Pointer	dhoni becomes oldest player captain india odi	dhoni becomes oldest player captain india odi led asia cup final exit wc tweet digging	0.63

But when we look for some examples, we find many repeated words in the encoder-decoder model including those with a relatively large F-score:

summary	pred_summary	rougel-f
demonetisation made homes affordable youth pm modi	pm modi pm modi pm modi	0.44
zarina wahab barkha bisht sengupta join pm modi biopic cast	biopic biopic pm modi biopic biopic	0.46
congress orop rahul priyanka amit shah	rahul shah one one one shah shah shah	0.44

As we mentioned in a previous section, most of the predicted summaries of the encoder-decoder model consist in just 2 or 3 words repeated continuously, so it is a model that does not meet the objectives from the point of view of a user. This problem was the reason why we analyzed alternatives until we found a solution in the Pointer Generator Network.

Some other examples with lower values in our Rouge metrics are:

	summary	pred_summary	rougel-f
Clustering	group men arrested allegedly threatening trying barge cricketer mohammed shami	group young men allegedly threatened tried barge india	0.30
Seq-2-Seq	people said bikini shoots anyway hansika private pics leak	people people pics online online	0.33
Pointer	border lead sunny deol pays tribute 1971 war hero kuldip singh	sunny deol portrayed brigadier kuldip singh 1997 borde	0.30

We said previously that extractive solution had been for a long time the only option to solve the summarization problem. And it is very appropriate when the source text is composed by many sentences where every text contains 12-15 sentences, 400 words. In this scenario we can select two or three sentences to extract the main ideas on the text, but when we have a shorter text or we need a summary or headlines with only 15 words, it is more difficult to compress all the meaning of the text by selecting only one sentence in the source text. For evaluation purpose, we forced our extractive algorithm to create a prediction just using 1 or 2 sentences, now, we are interested in evaluating this model using more sentences. We increased the number of sentences in the summary to 5 or 6 but the value of our F-score metrics were very similar:

Mean Rouge-2 FScore: 0.14365355377531439 Mean Rouge-L FScore: 0.2927782819325711

We must consider in this evaluation the training time where the extractive approach is clearly the best option, you can summarize about 10,000 examples in 4-5 minutes but for a encoder-decoder model it takes about 6-8 hours to get an acceptable model.

Now we know how our models behave with respect to data from our dataset**, but we must understand how good it would predict data from other datasets** and probably with different word distributions. We will use some examples from the CNN DailyMail dataset, it includes long texts and summaries. But these examples are also about news so it might be that they both share a similar vocabulary and grammatical compositions.

First, we will calculate the rouge metrics for 3,000 examples from the CNN dataset:

	rouge2-f	rouge2-p	rouge2-r	rougel-f	rougel-p	rougel-r
Sentence Clustering	0.058	0.033	0.340	0.121	0.070	0.070
Pointer Generator	0.035	0.035	0.035366695	0.136	0.149	0.149

The values for rouge-2 metric are very low, meaning that occurrences of 2-grams in the predicted summaries and the target summaries are not similar. The metric is not a perfect score so we cannot confirm that the results are not good enough to consider the model as a bad predictor.

Maybe we can explore some predicted summaries from the pointer generator to get a better intuition about the results, to check if the results differ much from the previous ones (we do not show prediction from the clustering model because the results from the extractive method are just a small part of the original source):

source test	target summary	predicted summary
attention facebook ipo people wondering world eduardo saverin cofounder facebook whose falling ceo mark zuckerberg immortalized film social network moved us singapore also recently revealed saverin born brazil gave us citizenship months ago became permanent singapore resident singapore according spokesman saverins move practical plans invest companies strong interests entering asian markets accordingly made sense use singapore home base one glaring possibility singapore capital gains tax	eduardo saverin cofounder facebook moved singapore recently renounced us citizenship permanent singapore resident saverins spokesman	saverin born brazil gave us citizenship months us singapore report report report report spokesman <UNK>
longdistance swimmer attacked great white shark near pier manhattan beach california los angeles county fire department said fisherman pier hooked shark saturday trying reel struggling fish victim swam lafd spokesman rick flores said agitated swimmer got close bit flores told cnn victim part group swimmers training waters near pier see shark late victim bit torso sustaining wound flores described moderate shark bit swimmer released said	great white shark bites swimmer waters manhattan beach california shark struggling end fishing line victim	swimmer attacked great white shark saturday beach fire department said said said said said said

The results show the same deficiencies many repeated words and short responses, but the words are a good selection from the source text and target. It looks like it has learnt

how to select the words from the vocabulary and the original text to form a representative summary, but usually incoherent.

Justification

We have mentioned previously that our goal was to search for two different types of solution to the text summarization problem and try to analyze and compare their results for a better intuition on how to solve the problem.
In the previous section we show that both developed models have outperformed the results of the benchmark model, the gensim summarizer, on the rouge-L f-score but just for a very small margin. In the case of the rouge-2 f-score the results are very similar. It is difficult to say that the pointer generator network is the best solution with the experiments we have done.

The process of training and adjust a encoder-decoder network is very consuming, requires a lot of computational resources (our training on a medium size dataset just for 5 epochs takes about 4-5 hours) and we could not face an exhaustive analysis of hyperparameters tuning. But we think that we are on the way to build a summarizer good enough to provide a short summary containing the main topics but with grammatical errors and repeated words.

At this moment, choosing the "sentence clustering" algorithm would be a safer solution but we can not discard the pointer network when we need a summary in just a few words.

V. Conclusion

It is time to think about how we have faced this hard to solve problem that today is still under research, we can not state that the best solution has been found.

Our first step was to find a dataset, a collection of source text and summary to work with. Our selection has an appropriate size and content but we have to admit that it was collected by scraping a website and the topics was focused on India, it contains a lot of names and places not very common and many misspelled words were found. Then we need to apply a cleaning process, removing punctuation, hashtags, and also removing stop words that we were not sure how could impact the results of our abstractive solution. We analyze the distribution of stop words, nouns, verbs, etc.… both in the source text and summaries to confirm that they were consistent, basically the shape of the distributions in the feature (the source text) and the target)the summary) should look similar.

Once we have a dataset ready to use, we create the benchmark model because it would be our guide on how good the next models would be. It was simple but even so it helped us to deepen our knowledge of the data and the complexity of the problem.

Then we build the sentence embeddings method, we found some resources on the internet that help us to understand clearly how it works. Our first relevant decision appeared, which embeddings we would use to transform our text in an array of numbers. We thought about creating our own embeddings using fasttext or selecting a well-known set of embeddings that usually works fine in many problems. Finally, we decided to use the Glove vectors, with 100 number per token, a larger embedding (300) would increase the processing time and the benefits, we think, were not going to be remarkable. When we were building this model, we realized the importance of determining the size of the summary text, how many sentences it should contain. This point already indicated that for short summaries it might not be the best alternative.

But the most important problem, and the one that made us realize the magnitude of the problem, appeared when we saw the first results of the encoder-decoder model. They were very discouraging, we found texts where 1 or 2 words were repeated continuously, resulting in completely incorrect solutions. Consequently, we had to ask ourselves whether we should stick with that proposal and point out its serious drawbacks or whether to find a new solution

Therefore, we began a process of research to find new improvements to the model ([20], [21]) and we decided to give the pointer generator a try. It was hard to find components to build our new model, we need to change part of our previous work and we finally managed to develop and build a new algorithm, train it and analyze the results. The predicted summaries were better, we almost solved the repetition word problem, but they were good enough to be considered a solution for the problem. We need to admit that not all the issues were solved, we need more computational resources to tune the hyperparameters and the use of GPU was very poor. It did not even work in some workspace like Google Cloud containers and Azure Data Science Virtual Machines.

But we have learned many lessons to consider in the future and the results achieved can be considered remarkable, given the complexity of the problem we are facing. There are many improvements to consider and we will discuss them in the next section.

Improvement

We have mentioned that this topic, text summarization, in the last two or three years has been under an intensive research process. Many papers have been published and discuss, so many improvements or even new models are still in progress.

But we will focus, in the next lines, in some points we can consider in the case of the pointer generator network:

- Include more training data: We have other sources of news data that with some adjustments could be included as examples in our dataset. This would extend our vocabulary and could allow us to train our model for longer time.
- Train the model for longer, more epochs: with a higher computing capacity and more examples we can train for longer and probably get some improvement.
- Make more efficient use of GPU resources, and solve some issue that our actual solution suffers.
- Hyperparameter tuning: We did some manual tuning but a more intensive study could guide us to a much better solution.
- Review and adapt the cleaning text process: for example we could consider not to remove stop words, if we are working with a large corpus and a long time of training it might learn how to use this “special” words.
- Implement additional linguistic features embeddings: we can search for other alternatives to build our embeddings, including some extra features.
- A new scoring metric of function: we could test BLEU metric instead of Rouge, we know that rouge is not a perfect metric. Sometimes it provides high value to a predicted summary that it can hardly be readable.
- Try beam search during inference: this kind of model operates by generating probability distributions across the vocabulary of output words and sample the probability distributions to generate the most likely sequences of words. Instead of greedily choosing the most likely next word as the sequence is constructed,

the beam search expands all possible next steps and keeps the *k* most likely. And select the word from that small group.

In the extractive approach, which not include a real training step, there are just a few issues to analyzed:

- Mapping between the cleaned sentence and the original one: The model works with the cleaning text, it is better when we want to focus on relevant words. But when the model chooses a sentence to include in the output summary, we can replace it with the original one, building a summary with less grammatical errors.
- Build a custom set of embeddings: we could try to train our own embedings or test with some other alternatives.

Links

[1]- IDC:Expect 175 zettabytes of data worldwide by 2025 https://www.networkworld.com/article/3325397/idc-expect-175-zettabytes-of-data-worldwide-by-2025.html
[2] - Kaggle Dataset, https://www.kaggle.com/sunnysai12345/news-summary
[3] - Hovy, E. H. Automated Text Summarization. In R. Mitkov (ed), The Oxford Handbook of Computational Linguistics, chapter 32, pages 583–598. Oxford University Press, 2005
[4] - https://en.wikipedia.org/wiki/ROUGE_(metric)
[5] - What Is ROUGE And How It Works For Evaluation Of Summarization Tasks? https://rxnlp.com/how-rouge-works-for-evaluation-of-summarization-tasks/#.Xr5aq2j7RPY
[6] CNN Daily Mail dataset - https://cs.nyu.edu/~kcho/DMQA/-
[7] - Seconds from Disaster: Text EDA and Analysis (Kaggle notebook) https://www.kaggle.com/shivamb/seconds-from-disaster-text-eda-and-analysis
[8] - Advances in Automatic Text Summarization, 1999. The MIT Press Cambridge, Massachusetts
[9] - Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. https://nlp.stanford.edu/pubs/glove.pdf
[10] - How to Develop an Encoder-Decoder Model with Attention in Keras
https://machinelearningmastery.com/encoder-decoder-attention-sequence-to-sequence-prediction-keras/
[12] - How Does Attention Work in Encoder-Decoder Recurrent Neural Networks
https://machinelearningmastery.com/how-does-attention-work-in-encoder-decoder-recurrent-neural-networks/
[11] - Neural Machine Translation by Jointly Learning to Align and Translate, 2015. arXiv:1409.0473.
[13] - https://rare-technologies.com/text-summarization-with-gensim/
[14] -TextRank: Bringing Order into Texts https://web.eecs.umich.edu/~mihalcea/papers/mihalcea.emnlp04.pdf
[15] - Introduction to Information Retrieval, By Christopher D. Manning, Prabhakar Raghavan & Hinrich Schütze, https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html
[16] - Illustrated Guide to LSTM's and GRU's: A step by step explanation by Micheal Phi,
https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
[17]- Attention in RNNs by Nir Abel, https://medium.com/datadriveninvestor/attention-in-rnns-321fbcd64f05
[18]- What is Teacher Forcing for Recurrent Neural Networks?, https://machinelearningmastery.com/teacher-forcing-for-recurrent-neural-networks/
[19]- Get To The Point: Summarization with Pointer-Generator Networks by Abigail See, Peter J. Liu, Christopher D. Manning, arXiv:1704.04368 .
[20] - Modern History for Text Summarization by Pengfei Liu,
http://pfliu.com/Historiography/summarization/summ-eng.html
[21] - Searching for Effective Neural Extractive Summarization: What Works and What's Next by Pengfei Liu,
http://pfliu.com/paper/interpretSum.pdf