On the very first day we were given multiple research papers on **Automatic Text Summarizer** to study and assess. In that week conferences had been scheduled to arrange introductory classes for a few of the interns and all of us had been made acquainted with the subject and the assignment at hand..which turned into increasing a version of automated textual content summarizer. We divided ourselves into companies of four
humans and researched numerous algorithms and strategies to increase our version.

My group worked on the TextRank Algorithm and coded it to assess its performance.
When it comes to the dataset to train the model with TextRank algorithm, The advantage of using the TextRank algorithm is no need of huge corpus for training.Thus the dataset we chose was a large text from wikipedia.
We applied several pre-processing techniques on it like removal of stopwords, tokenization. We calculated the frequency of every token and normalized them.

Then a "sentence score" was calculated to pick sentences to form a summary. However, the sequence of sentences was a bit distorted in the summary from the original text. Thus we discarded
our model in favor of the common model that was agreed upon by all teams. The dataset finalized for the common model was 'Amazon Fine Food Reviews' dataset.

In this dataset we try to make efficient model with about 6,00,000 records titled 'Amazon Fine Food Reviews'

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

The dataset consisted of the following 10 columns:

- Id
- ProductId
- UserId

- ProfileName
- HelpfulnessNumerator
- HelpfulnessDenominator
- Score
- Time
- Text
- Summary

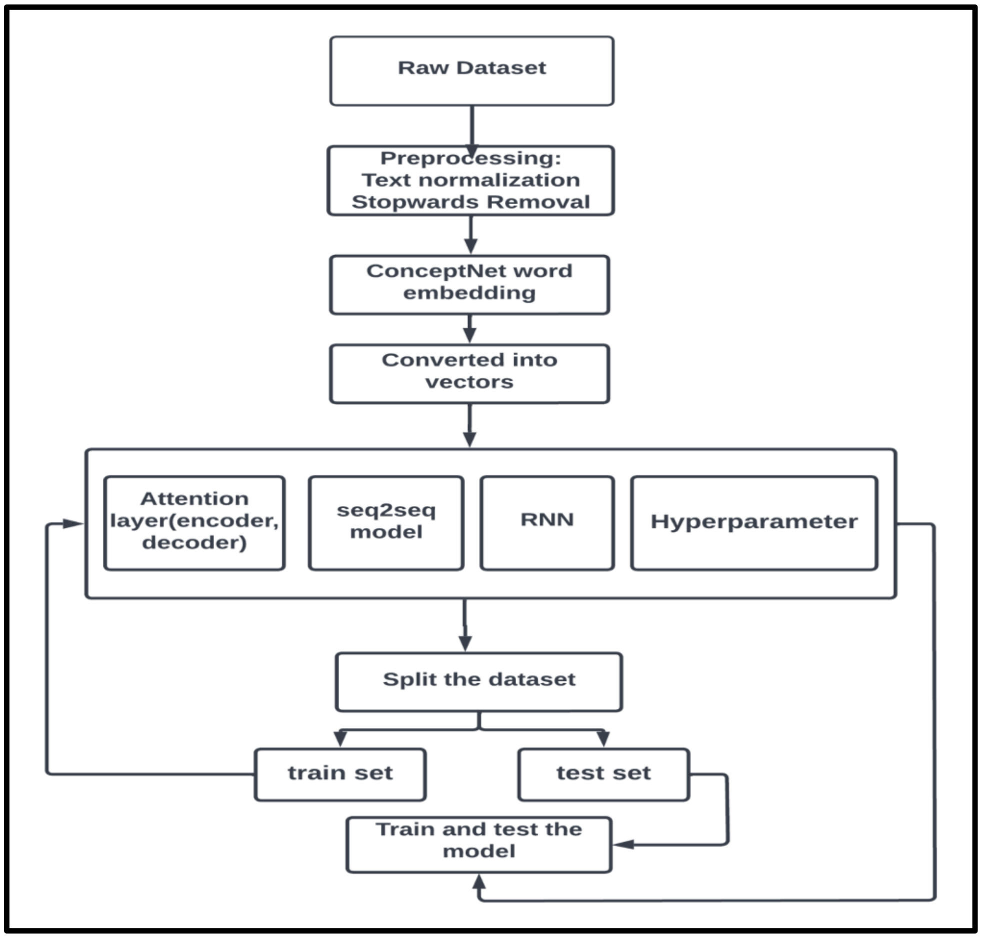
Out of these 10 columns, we eliminated 8, keeping only ‘Text’ and ‘Summary’.

Objective

Our objective here is to generate a summary for the Amazon Fine Food reviews using the abstraction-based approach we learned about above.We chose the ‘Amazon Food Reviews ‘ dataset to test and train our model.We split the dataset into an 80:20 ratio, trained the train set on the algorithm, and then tested the model on the test set.

****Below is the depicted flowchart in which format we have worked and trained our model to give the perfect and crisp summary.**

FlowChart



****Then we have done the following preprocessing techniques on the dataset.**

Preprocessing

Performing basic preprocessing steps is very important before we get to the model building part. Using messy and unclean text data is a potentially disastrous move. So in this step, we will drop all the unwanted symbols, characters, etc. from the text that do not affect the objective of our problem.

a) Text Cleaning

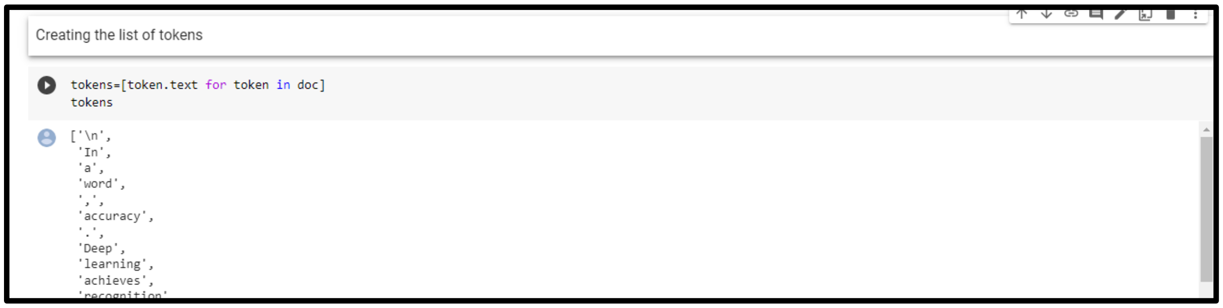
Let’s look at the first 10 reviews in our dataset to get an idea of the text preprocessing steps:

We will perform the below preprocessing tasks for our data:

- Convert everything to lowercase
- Remove HTML tags
- Contraction mapping
- Remove (‘s)
- Remove any text inside the parenthesis ( )
- Eliminate punctuations and special characters
- Remove stopwords
- Remove short words

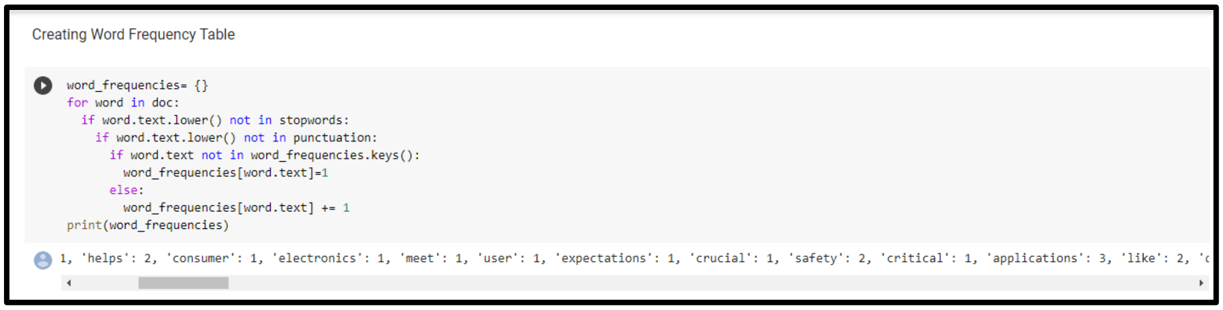
So, here in preprocessing technique we performed:

- To Pre-process the data we intend to remove the **stopwords** which are predefined and imported from the **spacy** package. And the punctuations that can be ignored from the text are also removed.
- Tokenization: the tokenization involves converting the whole text into the meaningful words (**word tokenization**) or Sentences (**sentence tokenization).**
- The text that is considered is tokenized, which means converting the whole text into meaningful words.
- The words and sentences which were tokenized **are ranked by taking** score on the basis of the frequency of the words and sentences.


Creating the list of tokens

```
tokens=[token.text for token in doc]
tokens
```

```
['\n',
 'In',
 'a',
 'word',
 ',',
 'accuracy',
 '.',
 'Deep',
 'learning',
 'achieves',
```


Creating Word Frequency Table

```
word_frequencies= {}
for word in doc:
  if word.text.lower() not in stopwords:
    if word.text.lower() not in punctuation:
      if word.text not in word_frequencies.keys():
        word_frequencies[word.text]=1
      else:
        word_frequencies[word.text] += 1
print(word_frequencies)
```

```
1, 'helps': 2, 'consumer': 1, 'electronics': 1, 'meet': 1, 'user': 1, 'expectations': 1, 'crucial': 1, 'safety': 2, 'critical': 1, 'applications': 3, 'like': 2,
```

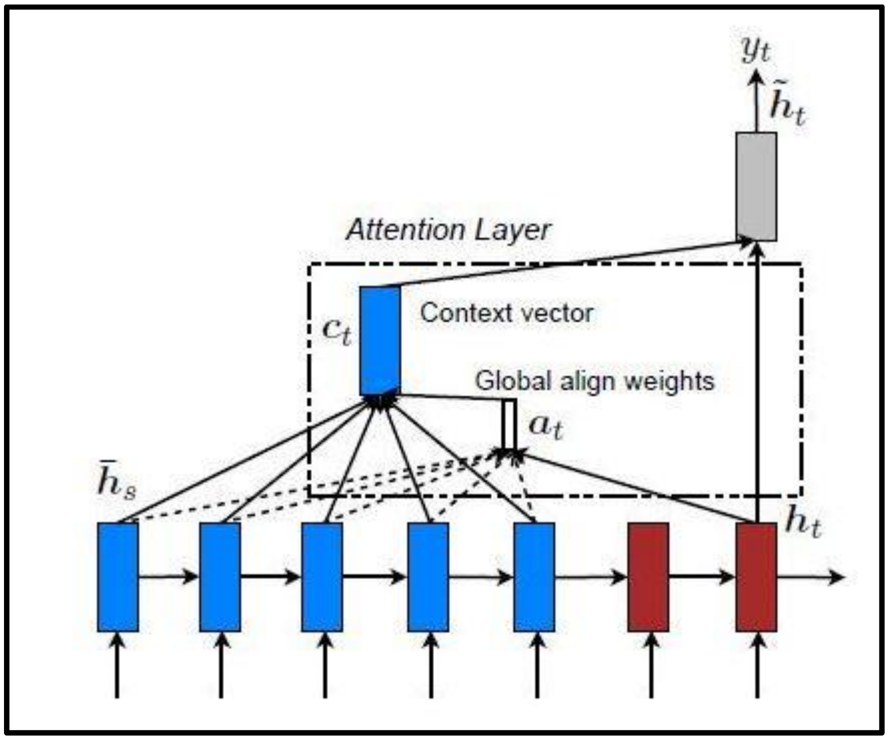
Attention Mechanism

How much attention do we need to pay to every word in the input sequence for generating a word at timestep? Instead of looking at all the words in the source sequence, we can increase the importance of specific parts of the source sequence that result in the target sequence. This is the basic idea behind the attention mechanism.

Algorithm

1. Encode the entire input sequence and initialize the decoder with internal states of the encoder
2. Pass <start> token as an input to the decoder
3. Run the decoder for one timestep with the internal states
4. The output will be the probability for the next word. The word with the maximum probability will be selected
5. Pass the sampled word as an input to the decoder in the next timestep and update the internal states with the current time step

6. Repeat steps 3 – 5 until we generate <end> token or hit the maximum length of the target sequence



Customer reviews can often be long and descriptive. Analyzing these reviews manually, as you can imagine, is really time-consuming. This is where the brilliance of Natural Language Processing can be applied to generate a summary for long reviews.

We will be working on a really cool dataset. Our objective here is to generate a summary for the Amazon Fine Food reviews using the abstraction-based approach we learned about above

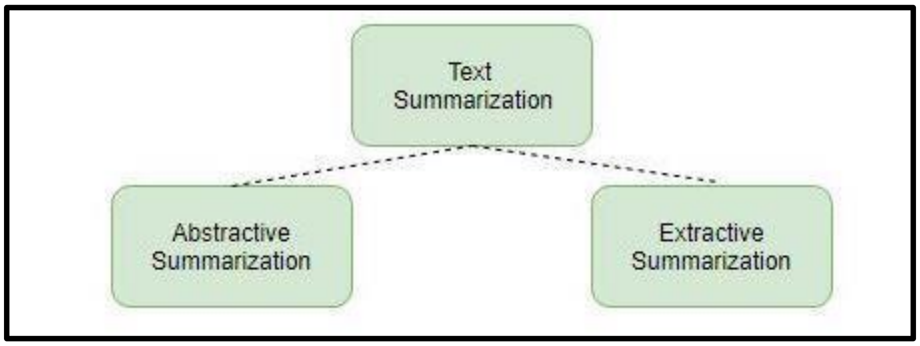
This dataset consists of reviews of fine foods from Amazon. The data spans a period of more than 10 years, including all ~500,000 reviews up to October

2012. These reviews include product and user information, ratings, plain text review, and summary. It also includes reviews from all other Amazon categories.

We’ll take a sample of 100,000 reviews to reduce the training time of our model. Feel free to use the entire dataset for training your model if your machine has that kind of computational power.

There are broadly two different approaches that are used for text summarization:

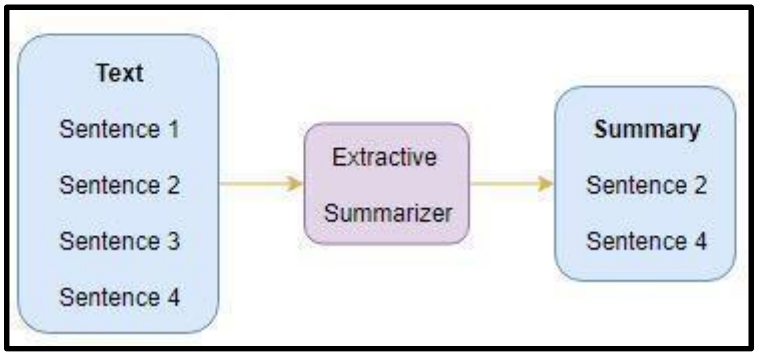
- Extractive Summarization
- Abstractive Summarization



Let’s look at these two types in a bit more detail.

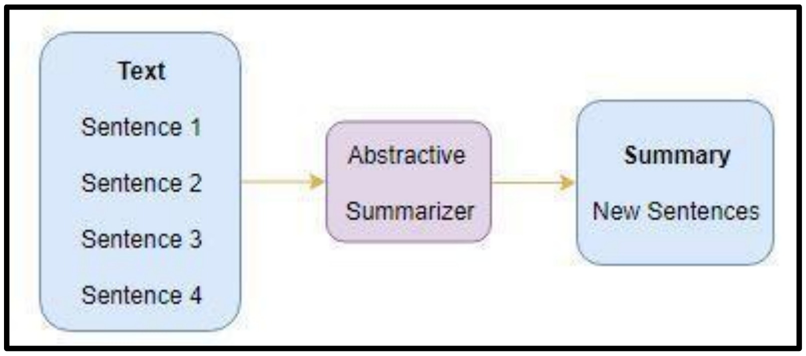
Extractive Summarization

The name gives away what this approach does. We identify the important sentences or phrases from the original text and extract only those from the text. Those extracted sentences would be our summary. The below diagram illustrates extractive summarization:



Abstractive Summarization

This is a very interesting approach. Here, we generate new sentences from the original text. This is in contrast to the extractive approach we saw earlier where we used only the sentences that were present. The sentences generated through abstractive summarization might not be present in the original text:



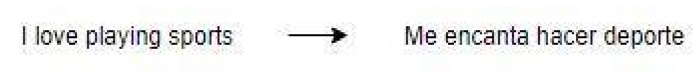
You might have guessed it – we are going to build an Abstractive Text Summarizer using Deep Learning in this article! Let’s first understand the concepts necessary for building a Text Summarizer model before diving into the implementation part.

****Here we are gonna make Abstractive Summarization, so for that we have taken suitable algorithm into consideration**

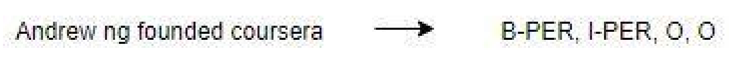
Sequence-to-Sequence (Seq2Seq) Modeling

We can build a Seq2Seq model on any problem which involves sequential information. This includes Sentiment classification, Neural Machine Translation, and Named Entity Recognition – some very common applications of sequential information.

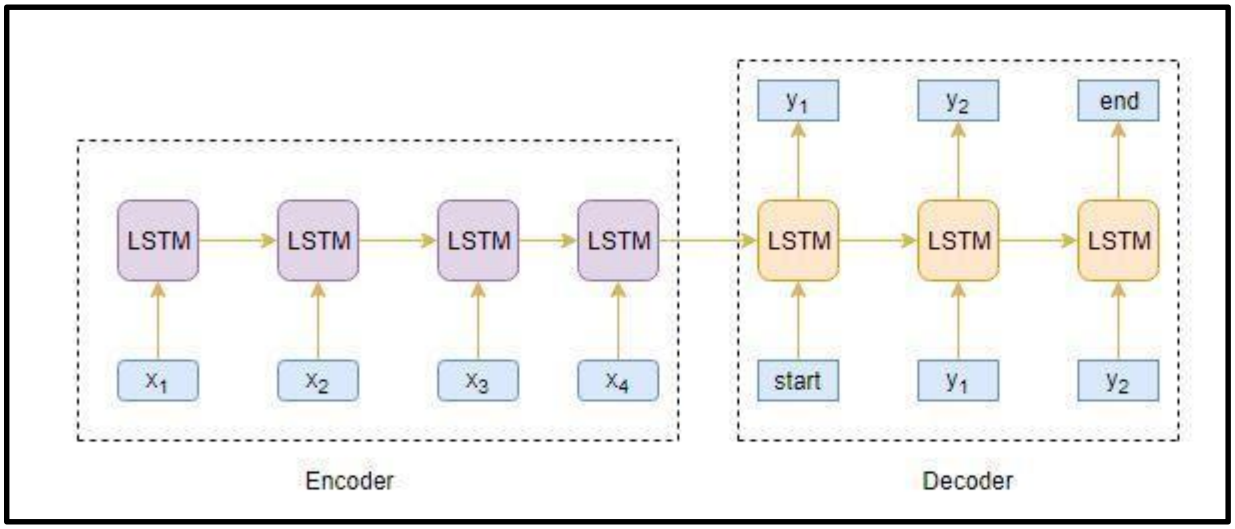
In the case of Neural Machine Translation, the input is a text in one language and the output is also a text in another language:



In the Named Entity Recognition, the input is a sequence of words and the output is a sequence of tags for every word in the input sequence:



Our objective is to build a text summarizer where the input is a long sequence of words (in a text body), and the output is a short summary (which is a sequence as well). So, we can model this as a Many-to-Many Seq2Seq problem. Below is a typical Seq2Seq model architecture:

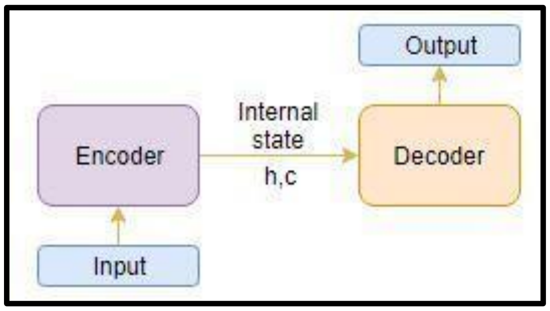


There are two major components of a Seq2Seq model:

- Encoder
- Decoder

Understanding the Encoder-Decoder Architecture

The Encoder-Decoder architecture is mainly used to solve the sequence-to-sequence (Seq2Seq) problems where the input and output sequences are of different lengths.



Generally, variants of Recurrent Neural Networks (RNNs), i.e. Gated Recurrent Neural Network (GRU) or Long Short Term Memory (LSTM), are preferred as the encoder and decoder components. This is because they are capable of capturing long term dependencies by overcoming the problem of vanishing gradient.

We can set up the Encoder-Decoder in 2 phases:

- Training phase
- Inference phase

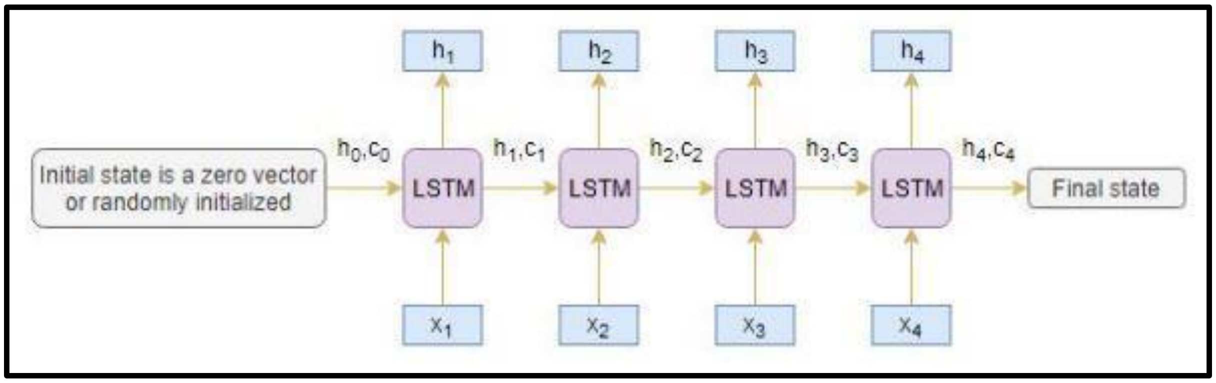
Training phase

In the training phase, we will first set up the encoder and decoder. We will then train the model to predict the target sequence offset by one timestep. Let us see in detail on how to set up the encoder and decoder.

Encoder

An Encoder Long Short Term Memory model (LSTM) reads the entire input sequence wherein, at each timestep, one word is fed into the encoder. It then processes the information at every timestep and captures the contextual information present in the input sequence.

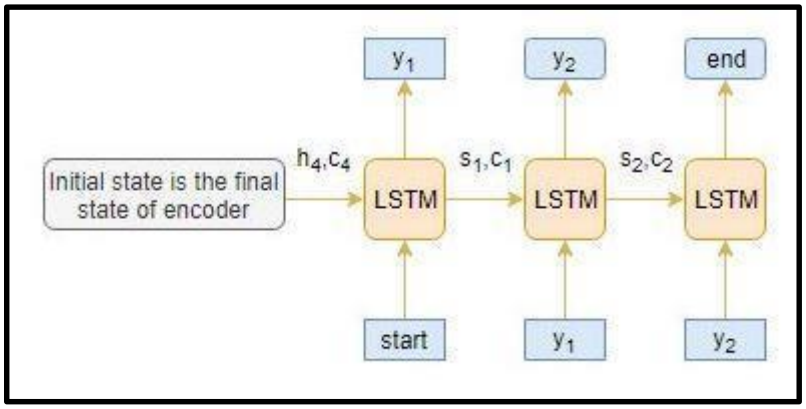
I’ve put together the below diagram which illustrates this process:



The hidden state (hi) and cell state (ci) of the last time step are used to initialize the decoder. Remember, this is because the encoder and decoder are two different sets of the LSTM architecture.

Decoder

The decoder is also an LSTM network which reads the entire target sequence word-by-word and predicts the same sequence offset by one timestep. The decoder is trained to predict the next word in the sequence given the previous word.

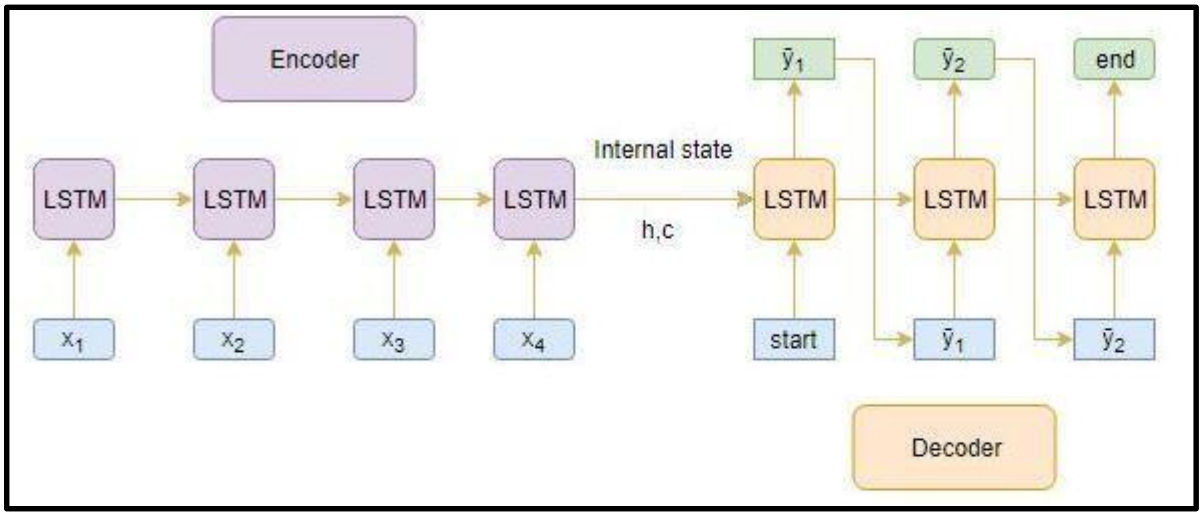


<start> and <end> are the special tokens which are added to the target sequence before feeding it into the decoder. The target sequence is unknown while decoding the test sequence. So, we start predicting the target sequence by passing the first word into the decoder which would be always the <start> token. And the <end> token signals the end of the sentence.

Pretty intuitive so far.

Inference Phase

After training, the model is tested on new source sequences for which the target sequence is unknown. So, we need to set up the inference architecture to decode a test sequence:



Limitations of the Encoder – Decoder Architecture

As useful as this encoder-decoder architecture is, there are certain limitations that come with it.

- The encoder converts the entire input sequence into a fixed length vector and then the decoder predicts the output sequence. This works only for short sequences since the decoder is looking at the entire input sequence for the prediction
- Here comes the problem with long sequences. It is difficult for the encoder to memorize long sequences into a fixed length vector

"A potential issue with this encoder-decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sentences. The performance of a basic encoder-decoder deteriorates rapidly as the length of an input sentence increases."

Recurrent Neural Networks (RNN)

The Recurrent Neural Network (RNN) – a type of neural network that can perform calculations on sequential data (e.g. sequences of words) – has become the standard approach for many Natural Language Processing tasks.

The Group that I am a part of also worked with Text Rank Algorithm taking web content as a dataset to rank sentences and generate their summary.

Here is a brief about that

Text Summarizer using Text Rank algorithm:

Text Rank is a text summarization technique which is used in Natural Language Processing to generate Document Summaries. Text Rank uses an extractive approach and is an unsupervised graph-based text summarization technique.

Step 1

Extract all the sentences from the text document, either by splitting at whitespaces or full stops, or any other way in which you wish to define your sentences.

Step 2

A Graph is created out of the sentences extracted in Step 1. The nodes represent the sentences, while the weight on the edges between two nodes are found by using a Similarity function, like Cosine Similarity or Jaccard Similarity.

Step 3

This step involves finding the importance (scores) of each node by iterating the algorithm until convergence, i.e., until consistent scores are obtained. (It can be said that this step involves application of the PageRank algorithm to the document, with the only difference being that the nodes are sentences instead of web pages).

Step 4
The sentences are sorted in a descending order based upon their scores. The first k sentences are chosen to be a part of the text summary.

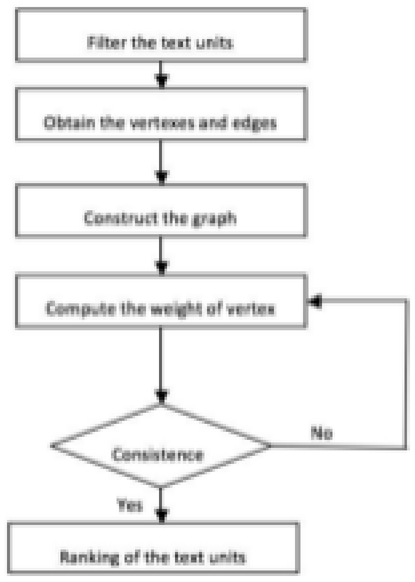
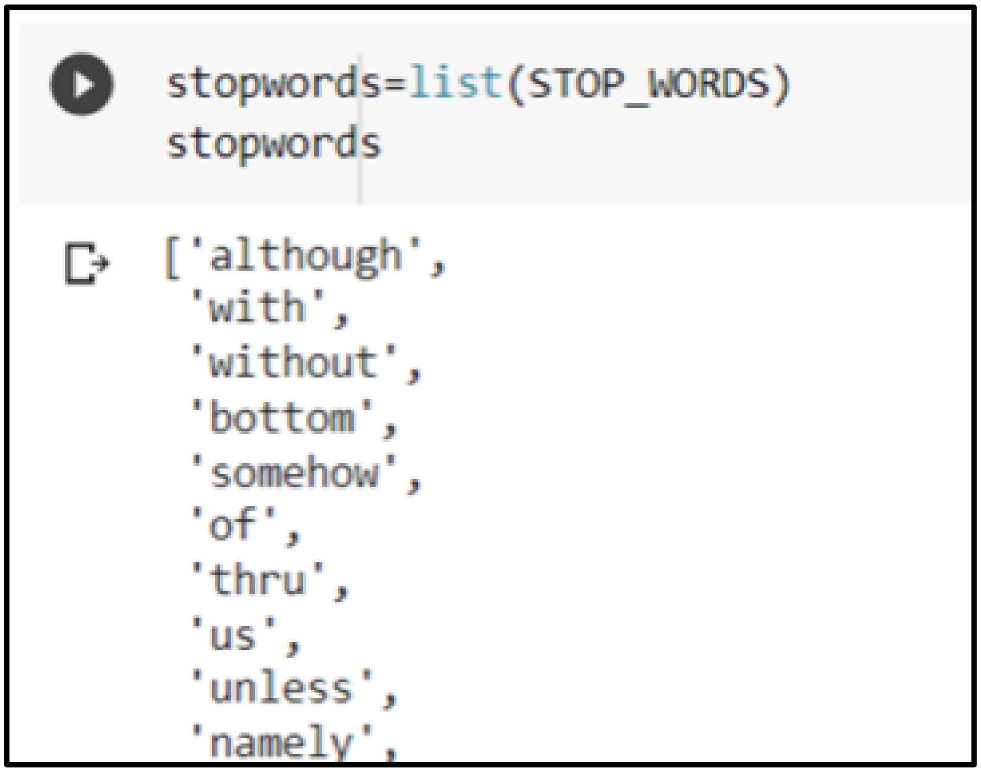

Filter the text units
Obtain the vertexes and edges
Construct the graph
Compute the weight of vertex
Consistence
No
Yes
Ranking of the text units

Figure 1. Classic TextRank algorithm workflow

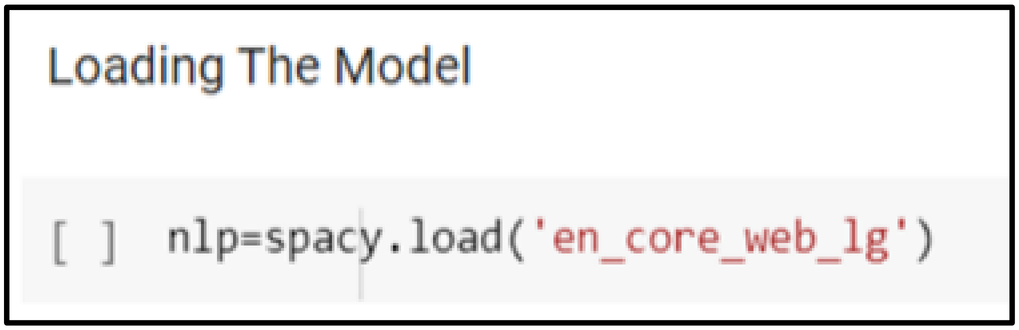
The advantage of Text Rank is that it is an unsupervised learning algorithm in no need of huge corpus for training. It makes it easy to be adopted for handling other text resources in an efficient way.
In the model that we have coded, we have deployed many NLP Techniques like
1. Removal Of Stop Words- Stop words are commonly used words excluded from searches to help index and parse web pages faster.



```
stopwords=list(STOP_WORDS)
stopwords
```

```
['although',
 'with',
 'without',
 'bottom',
 'somehow',
 'of',
 'thru',
 'us',
 'unless',
 'namely',
```

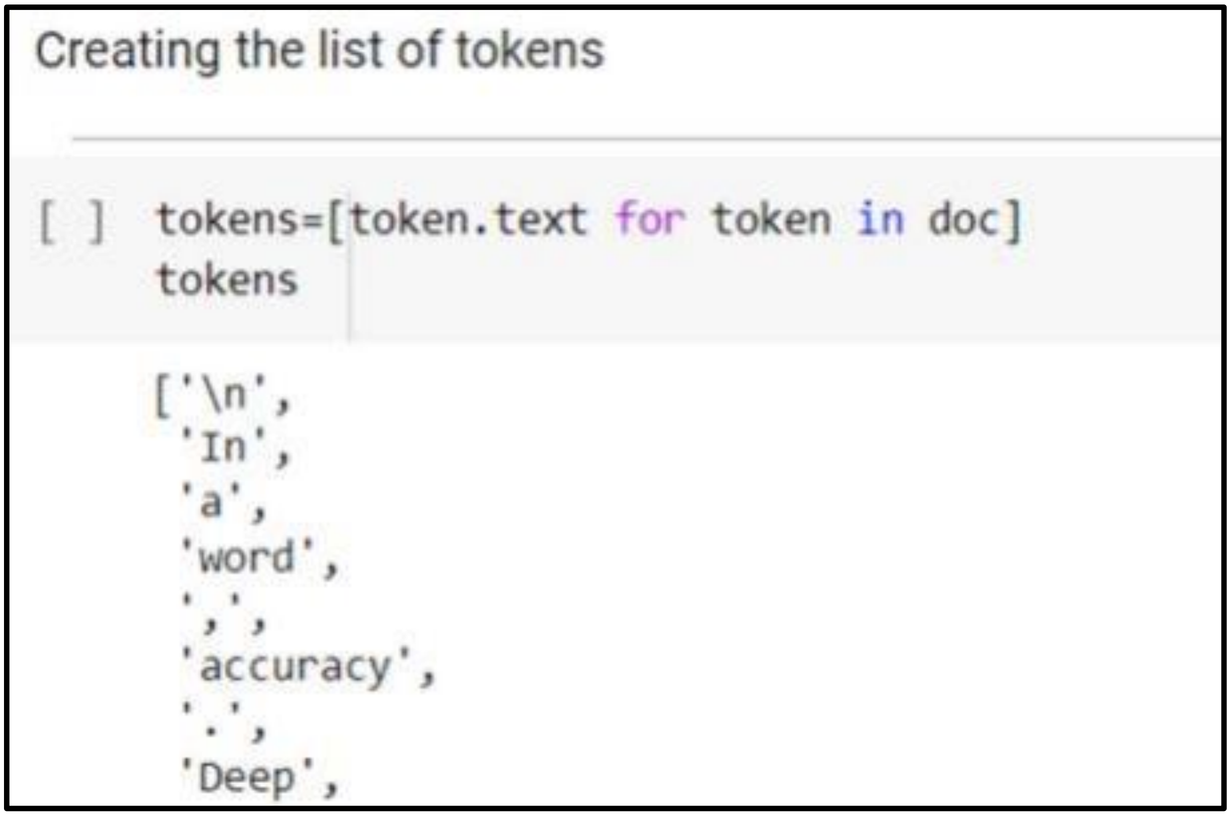
2. We are using the large version of core English Web Model available to us via the spacy library



Loading The Model

```
[ ]  nlp=spacy.load('en_core_web_lg')
```

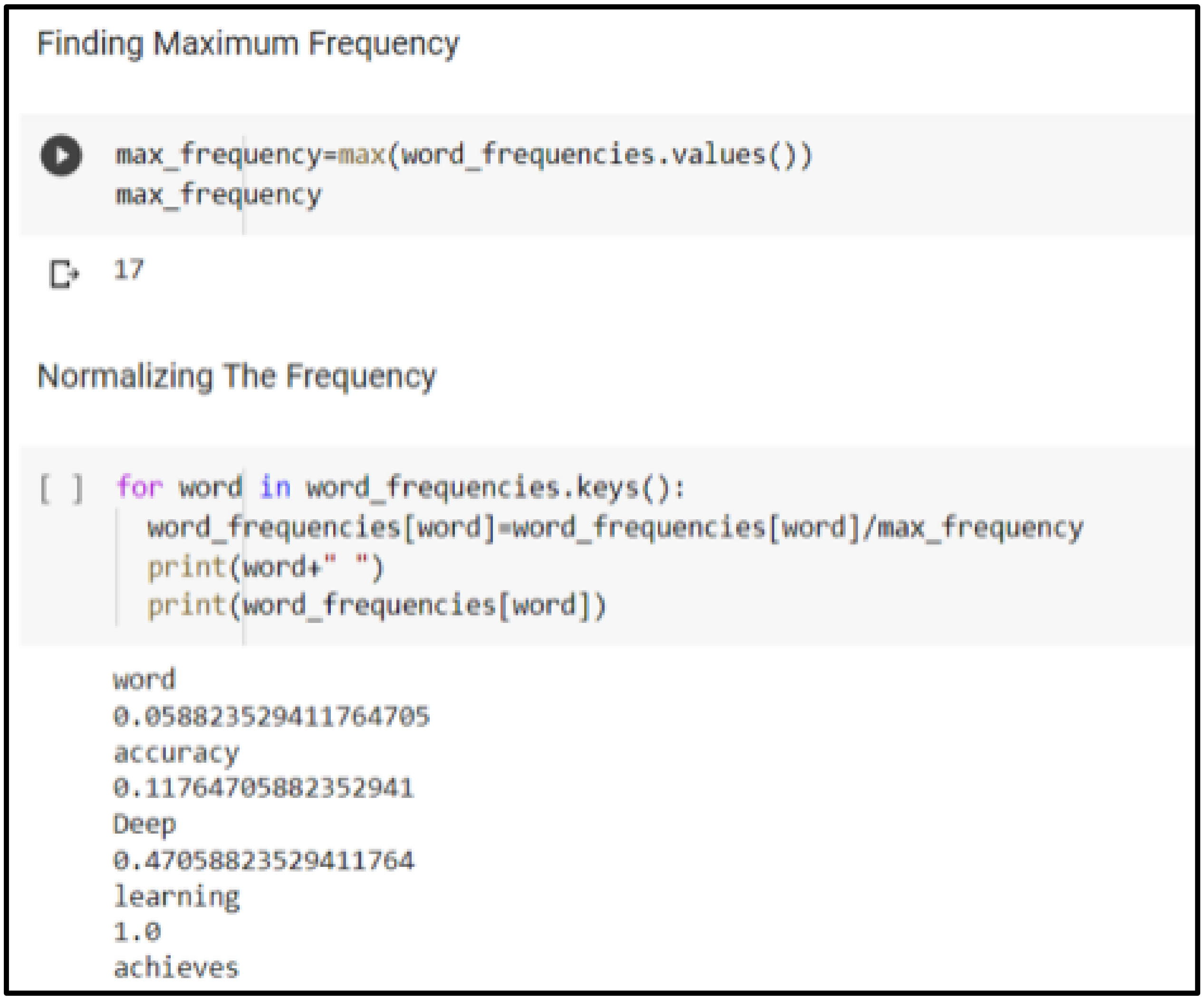
3. Tokenization: - Tokenization is a way of separating a piece of text into smaller units called tokens.


Creating the list of tokens

```
[ ]  tokens=[token.text for token in doc]
     tokens

     ['\n',
      'In',
      'a',
      'word',
      ',',
      'accuracy',
      '.',
      'Deep',
```

4. We calculated the frequency of each token and normalized the frequencies by dividing them by the maximum frequency.


Finding Maximum Frequency

```
max_frequency=max(word_frequencies.values())
max_frequency
```

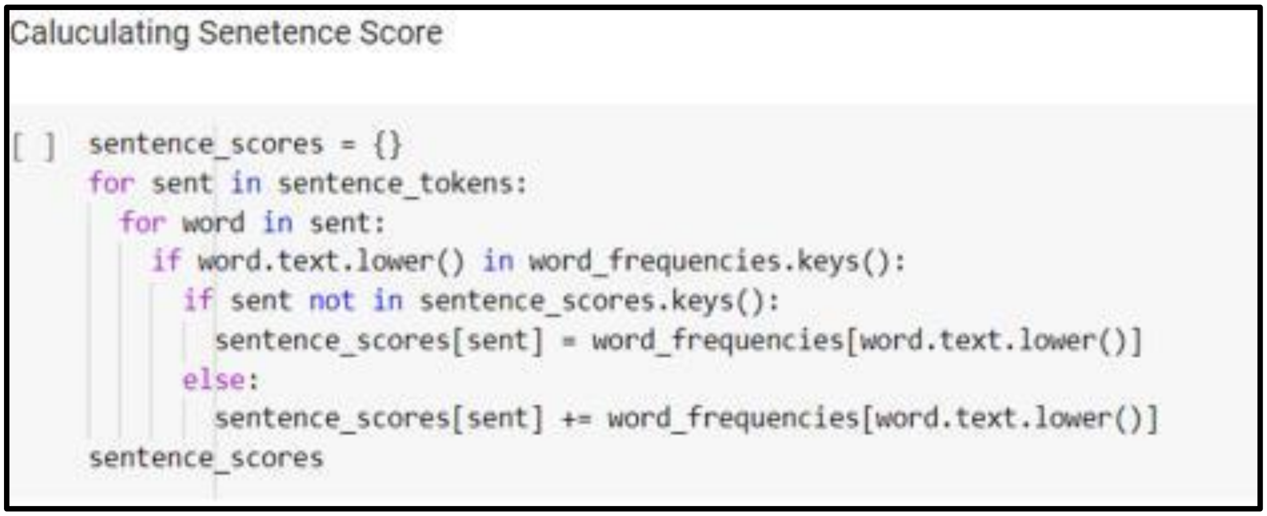
```
17
```

Normalizing The Frequency

```
for word in word_frequencies.keys():
  word_frequencies[word]=word_frequencies[word]/max_frequency
  print(word+" ")
  print(word_frequencies[word])
```

```
word
0.058823529411764705
accuracy
0.11764705882352941
Deep
0.47058823529411764
learning
1.0
achieves
```

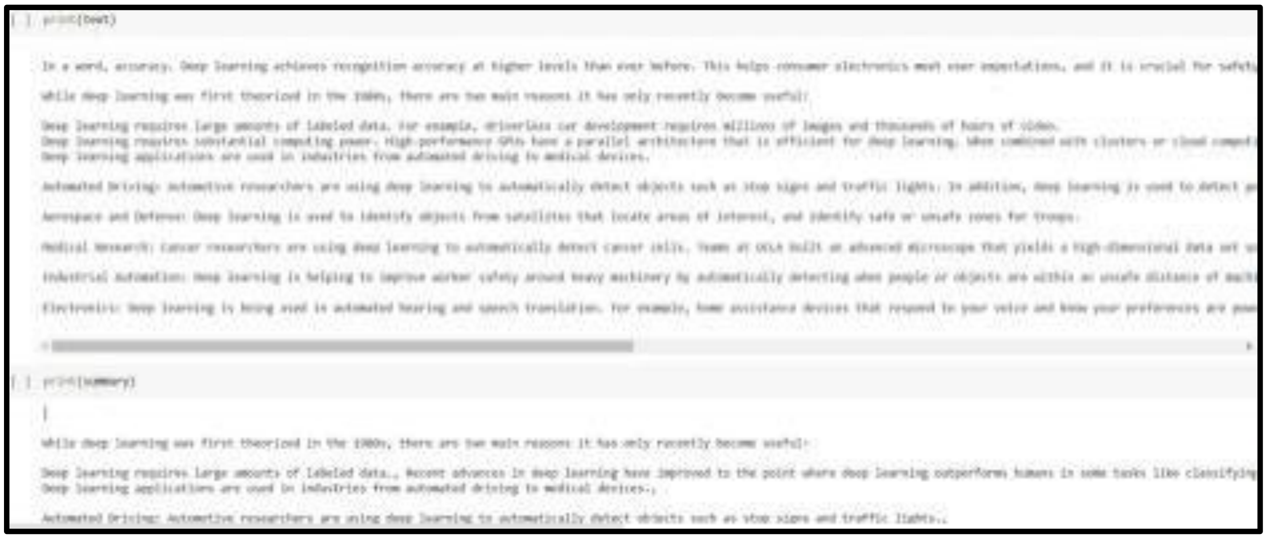
**5. After that we calculated the sentence score and ranked the sentences
for our summary accordingly**


Caluculating Senetence Score

```
sentence_scores = {}
for sent in sentence_tokens:
  for word in sent:
    if word.text.lower() in word_frequencies.keys():
      if sent not in sentence_scores.keys():
        sentence_scores[sent] = word_frequencies[word.text.lower()]
      else:
        sentence_scores[sent] += word_frequencies[word.text.lower()]
sentence_scores
```

The dataset that we used for this model was taken from a Wikipedia web page in accordance with the model that we were using.

Output (Result)



```
[ ] print(text)

    In a word, accuracy. Deep learning achieves recognition accuracy at higher levels than ever before. This helps consumer electronics meet user expectations, and it is crucial for safet

    While deep learning was first theorized in the 1980s, there are two main reasons it has only recently become useful:

    Deep learning requires large amounts of labeled data. For example, driverless car development requires millions of images and thousands of hours of video.
    Deep learning requires substantial computing power. High-performance GPUs have a parallel architecture that is efficient for deep learning. When combined with clusters or cloud computi
    Deep learning applications are used in industries from automated driving to medical devices.

    Automated Driving: Automotive researchers are using deep learning to automatically detect objects such as stop signs and traffic lights. In addition, deep learning is used to detect pe

    Aerospace and Defense: Deep learning is used to identify objects from satellites that locate areas of interest, and identify safe or unsafe zones for troops.

    Medical Research: Cancer researchers are using deep learning to automatically detect cancer cells. Teams at UCLA built an advanced microscope that yields a high-dimensional data set us

    Industrial Automation: Deep learning is helping to improve worker safety around heavy machinery by automatically detecting when people or objects are within an unsafe distance of machi

    Electronics: Deep learning is being used in automated hearing and speech translation. For example, home assistance devices that respond to your voice and know your preferences are powe

[ ] print(summary)

    |

    While deep learning was first theorized in the 1980s, there are two main reasons it has only recently become useful:

    Deep learning requires large amounts of labeled data., Recent advances in deep learning have improved to the point where deep learning outperforms humans in some tasks like classifying
    Deep learning applications are used in industries from automated driving to medical devices.,

    Automated Driving: Automotive researchers are using deep learning to automatically detect objects such as stop signs and traffic lights.,
```