NLP: Text Summarization in Machine Learning

Introduction:

Shortening a set of data computationally, to create a summary that represents the most important or relevant information within the original content, this is how Wikipedia describes Text Summarization. However, essentially text summarization means keeping the essence of the required text and make it easier and faster to read.

In today's world of rapid internet growth we breath in and out data and information, much of it in the form of text (this can include our texts to each other as well as a well fashioned blog). Occasionally this data needs to be understood and analyzed for various commercial applications, however the sheer amount of it makes it fairly improbable that we will be able to do so. This is where Text Summarization comes into the picture. We can summarize our text in a few lines by removing unimportant text and converting the same text into smaller semantic text form. This enables us to parse through the same data set quickly while preserving the apparent meaning conveyed.

The process of text summarization:

1. Preprocessing:

The first step in the process of text summarization is preprocessing. **n most cases for NLP**, preprocessing consists of **removing non-letter characters** such as "#", "-", "!", **numbers** or even **words that do not make sense** or are not part of the language being analyzed.

- We first initialize the **stopwords** which are words that appear frequently in the data but rarely add any meaning to it and are just present for language formation purposes.
- Then **words** from some English Dictionary eg: the nltk library
- Finally a **lemmatizer** which allows us to preserve the root of the given word eg: words like drink and drinking as also the words like neighbour and neighbourhood are considered as one as they largely convey the same meaning and the apparent difference in them is just to suit the sentence.
- Then the process of Stemming which is the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma, is carried out, stemming and lemmatizing essentially accomplish one combined task
- This pre-processing function does the following things:
    1. Removes Punctuations
    2. Removes Numbers
    3. Transforms sentences into a list of words as the stopwords connecting them are eliminated
    4. Lemmatizes the words to the root words.
    5. Changes capital letters to small ones, as this has no effect on the overall meaning of the dataset.

2. Initial Algorithms for Text summarization

Here are some of the algorithms that I stumbled upon in my research:

- **Luhn’s Algorithm:** Luhn's algorithm first filters terms in the document using a stop-list of closed-class words such as pronouns, prepositions, and articles. Next, terms are normalized based on aggregating together orthographically similar terms: Pairs of terms (e.g., "similar" and "similarity") are matched at each character position. Once a mismatch occurs, the number of non-similar subsequent letters of both terms is counted. If this count is less than 6, these terms are aggregated together. The frequencies of these aggregate terms in the document are noted, and low frequency terms are removed. Sentences are then weighted using the resulting set of "significant" terms and a term density measure. Each sentence is divided into segments bracketed by significant terms not more than four non significant terms apart. Each segment is scored by taking the square of the number of bracketed significant terms divided by the total number of bracketed terms. The score of the highest scoring segment is taken as the sentence score.Luhn describes several possible extensions to the basic algorithm, such as varying the length of the abstract, and giving an added premium to words in a domain-specific word list (ie., "bonus" words). He also mentions the possibility of applying the algorithm to foreign languages in order to create abstracts, which could then be translated to avoid translating the full document. Finally, he also suggests using these techniques to generate index terms for information retrieval. Luhn's basic idea of a statistical approach to text summarization, relying on ingredients such as term frequency and term normalization, has had a considerable influence on the field. In addition, the problem of text segmentation continues to be of interest. Subsequent progress in the field, however, allows one to identify some aspects of his approach that are no longer ideal. The particular type of term aggregation and normalization Luhn uses has since been supplanted by the use of stemming (Frakes 1992) or (especially for languages with rich morphology) morphological analysis; see also papers by Hovy and Lin and Aone et al. in Section 2 for further varieties of term aggregation. In information retrieval, absolute term frequency by itself is recognized as being less useful than term frequencies which are normalized to take into account document length and frequency in a collection (Harman 1992). Finally, thematic features such as term frequency by themselves are somewhat less useful in summarization compared to other features such as title or location, as shown in Edmundson's paper in this section, and also in the papers by Kupiec in Section 2 as well as Teufel and Moens in Section 3.

- **Edmundson’s Algorithm:** The next paper by Edmundson (1969) extends earlier work to look at three features in addition to word frequencies: cue phrases (e.g., "significant," "impossible," "hardly"), title and heading words, and sentence location. Edmundson created programs to weight sentences based on each of the four methods: cue phrase, keyword (i.e., term frequency based), location, and title. He then evaluated each of the programs by comparison against manually created extracts. He used a corpus-based methodology, dividing his set of articles into a training and test set. In the training phase, he used feedback from evaluations to readjust (by hand) the weights used by each of the programs, which were then tested and evaluated on the test data. Edmundson found that the three additional features dominated word frequency measures in the creation of better extracts. He also found that the combination of cue-title-location was the best, with location being the best individual feature and keywords alone the worst-performing algorithm.

Both of these algorithms are essentially from the very beginning of NLP and are therefore not too advanced on how they implement their objectives. But they are of correct complexity for a text summarization project although not without their misfires…

To combat them here are some of my ideas that I think would be able to improve the existing process of text-summarization.

As I read up on the process of text summarization, I realized how tricky it actually is. The language that we use on a day-to-day basis is never as simple as words having one-to-one meanings which could simply be condensed. Our language is largely dynamic ie: it keeps changing according to the times and relevancies. Even after the above process of pre-processing, most of the words that actually do have meaning rarely have the same amount of meaning. Additionally certain words are very important in certain contexts and depending on the way they are used can change the meaning of the sentence entirely eg: Negations will completely contradict the meanings of sentences when used. This makes our choice of the appropriate algorithm extremely important.

When I get to design an algorithm its main focus will be the maximum available character to information ratio, as this approach will enable the algorithm to achieve the best results ie: shorter and more comprehensive summaries. The following points can help the algorithm achieve this:

1. **Analyzing clusters of words together along with analyzing them separately.**
   After the initial pre-processing, the data is generally dense in what it means however, words can have meanings according to where they are used in a sentence and the pre-processing above will largely destroy such meaning. Instead I would propose an additional step after pre-processing with compares the remaining words with the words that are to discarded, thus extracting the meaning based upon their positions. A limited no. of words thus an be added or the order of the words can be changed such that the data remains condensed while having additional information about the retented words.

2. **Replacing sets of words with synonyms that might convey a better meaning.**
   If we find words that can be replaced with certain synonyms or similar words that might convey the meaning better, we would be better of replacing them in our dataset.

3. **Analyzing colloquial speech/writing patterns and working on them.**
   More often than not the words that we write never actually convey their exact same meaning, because of usage of idioms, phrases and sayings which certainly add to the conversation however objectively might not add meaning to the article itself. Most of the words involved in them will not be eliminated as stopwords as the words themselves are not meaningless however a collective set of their usage probably is. Eg words like 'you', 'sow', you', 'reap' in immediate succession are probably a part of the ages old saying however, if we are able to see through these patterns and words we would be able to extract the meaning as the rest of the written dataset surely does convey the same meaning.

4. **Extracting the main words from the sentence/phrase.**

   Extracting the most important word in the sentence/phrase can come in handy and sure will increase our information about the rest of the dataset. More often than not the most important words of a sentence are the subjects of the sentence themselves. All of this is done to summarise and assist in the relevant and well-organized organization, storage, search, and retrieval of content.
   There are numerous keyword extraction algorithms available, each of which employs a unique set of fundamental and theoretical methods to this type of problem. This I think can be of great help because the machine can be programmed to keep the subject as the focus and insert connectors and words according to the subject of the data. One way of doing this can be counting the frequency with which words appear in the dataset and the one having unusually high occurences is bound to be the Subject.
5. **Analysing the sentiments displayed in the given sentences.**
   Analysing the sentiment might be the most important of all of the proposed ideas. We can use a system that might contain three options like positive, negative or neutral options and the sentences can therefore be assigned any of these three values based upon the words involved in them. Many words like did, didn't , haven't have unfortunately been discarded before however this technique can be applied prior to that and they can factor into the large meaning of the summary.

These are the changes that I propose to the existing model of text-summarization. Now, I do realise that many of these require pre-existing information or might be difficult to implement. However, according to me they would be very helpful in keeping and enhancing the message conveyed in the original information, which I think the Algorithms mentioned before missed to fully achieve.

```
import pandas as pd
import numpy as np
data = "In its most proficient form i feel that MACHINE LEARNING is surely capable of
running the entire world! Not that it does not run a huge chunck of it now. From what I
understand ML is essentially teaching a machine to learn the world around it using
examples of how the current world works. So Imagine a case where we can TEACH the MACHINE
to essentiallly TEACH MACHINES!!! I surely understand that the above mentioned Endgame is
years away and I would love to contribute to this...



"
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords

def solve(text):
  stopwords1 = set(stopwords.words("english"))
words = word_tokenize(text)
freqTable = {}
```

```
for word in words:
  word = word.lower()
if word in stopwords1:
  continue
if word in freqTable:
  freqTable[word] += 1
else :
  freqTable[word] = 1

sentences = sent_tokenize(text)
sentenceValue = {}
for sentence in sentences:
  for word, freq in freqTable.items():
    sentence = sentence.lower()
  if word in sentence:
    if sentence in sentenceValue:
      sentenceValue[sentence] += freq
    else :
      sentenceValue[sentence] = freq
    sumValues = 0
for sentence in sentenceValue:
  sumValues += sentenceValue[sentence]
average = int(sumValues / len(sentenceValue))

summary = ''
for sentence in sentences:
  if (sentence in sentenceValue) and(sentenceValue[sentence] > (1.2 * average)):
    summary += "" + sentence
return summary
```