Memorial University

An Evaluation of Modern Text Summarization Methods

Joshua Peddle

COMP 4750

2022-12-01

Text summarization is the act of automatically shortening text while trying to retain meaning and information. The design and resulting output from a text summarization system depends on the use-case for the system, but some observations can be made. For human consumption, sentences should be human readable and make sense to the average reader in that language. Summaries should try to follow grammatical and syntactic rules that humans would expect from written text, although they need not be perfect as humans have some ability to parse imperfect text. Machine learning systems also find use in text summarization. Consider an automatic sentiment analysis system whose runtime scales with the size of the input text. By preprocessing the text with a suitable summarization system, the sentiment could be kept while shortening the execution time. Systems designed with machines analysis in mind may not place such requirements on grammatical and syntactic rules and focus on keeping more informative words.

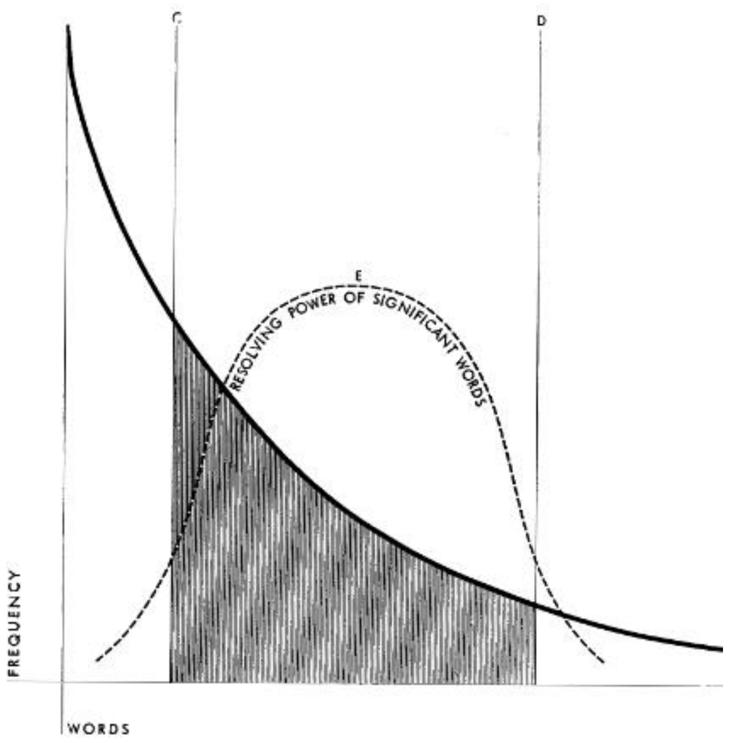
Since its rise in popularity in the 1970s following the work of Hans Peter Luhn many techniques have been theorized and applied to the field of text summarization. There are two clear ways in which these systems shorten a document. The first is through "sentence selection". The document(s) are broken up into sentences and some method of scoring the sentences is used to select a subset of the "best" sentences from the whole. These systems are the simplest to implement in practice since no care needs to be taken to consider grammar, syntax, or sentence structure. If each sentence is grammatically correct in the source, it will also be in the summary. The second method is sentence compression. These techniques are more difficult to implement, but the potential is higher than that of sentence selection. Words are removed from the source, usually by scoring the words of the text and dropping the lowest scoring ones. If the system need not be grammatically correct this simplifies the process. When shortening sentences for human consumption, the system must be careful not to remove words that could make the sentence unreadable. By removing the wrong words, the system could introduce unwanted ambiguity, lose meaning, or make it exceedingly difficult to understand. The technique proposed in “Modelling Compression with Discourse Constraints” by James Clarke and Mirella Lapata in 2007 uses various Sentential Constraints to ensure certain words are not removed. Their technique is explored more fully starting on pg. 7 of this paper. In practice both sentence selection and sentence compression can be used together to fully remove uninformative sentences and remove undesired words from the remaining informative sentences. “Multi-Document Summarization by Maximizing Informative Content-Words.” by Yih, Wen-tau et. al in 2007 use sentence selection by default but propose a method to simplify sentence that eliminates “various syntactic units based on predefined heuristic templates, such as removing noun appositives, gerundive clauses, nonrestrictive relative clauses, or intra-sentential attributions.” (Yih, Wen-tau et. al, 2007. pg 1779). Their method is explored in more detail on pg. 7.

Three software systems will be explored in the next section of this paper. All were implemented using Python version 3.6.9. The first two systems were implemented using only the standard library (math, random, string, json). The last method, “Modelling Compressing with Discourse Constraints” uses some third-party libraries.

The first is a sentence selection method proposed by H. P. Luhn in 1958. Readers may argue that this paper should not include such an old method but in practice systems like SumBasic are still in use today and perform very well while using a similar approach. This system attempts to assign a significance factor to each sentence.

Figure source (Luhn 1958. Pg 161)

Figure 1 **Word-frequency diagram.**
Abscissa represents individual words arranged in order of frequency.

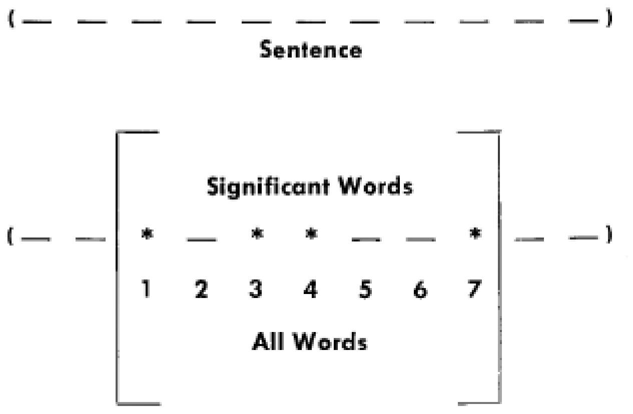


"The 'significance' factor of a sentence is derived from an analysis of its words. It is here proposed that the frequency of word occurrence in an article furnishes a useful measurement of word significance." (Luhn 1958. pg 160)

Luhn proposed that word frequency was a good indicator of significant words but found that there was an issue. Words that appeared the most were likely to be noise; Words like 'It' 'and' 'the' that provide no information. To deal with this, a method was proposed to eliminate frequently used words by comparing them to a common-word list. Luhn also briefly mentions another method that establishes a high and low frequency cutoff marked as lines C and D in the above figure. Not contained in the space between C and D would be discarded. For the software implementation, a common word list was crafted by adding English propositions and other frequently used words to a 'common_words.txt' file that is read in at the start of execution.

One interesting thing about this method is that it does not try to differentiate between word forms. "Similar" words would be considered the same word when evaluating frequency data. Although this can be seen in other summarization methods, the approach used was more primitive than the approaches used today. "A letter-by-letter comparison of pairs of succeeding words in the alphabetized list. From the point where letters failed to coincide, a combined count was taken of the non-similar subsequent letters of both words. When this count was six or below, the words were assumed to be similar notions; above six, different notions.” *(Luhn 1958. pg 162)* This method was used in the software implementation but a hyperparameter ‘non_similar_threshold=6’ was created to allow the user to change this value. As one can assume, this system is fallible and non-similar words in practice are assumed to be similar. Any prefix attached to the word renders the system useless; It is only capable of identifying words that differ by prefix. For example, the system would identify ‘President’ ’Presidents ‘Presidential’ as similar, not a bad outcome. On the other hand, the system would consider ‘force’ and ‘former‘ to be similar words; Not an excellent choice.

Figure source (Luhn 1958. Pg 162)



Portion of sentence bracketed by and including significant words not more than four non-significant words apart. If eligible, the whole sentence is cited.

Using the word-frequency data, sentences are given a significance score “which reflects the number of occurrences of significant words within a sentence and the linear distance between them” (Luhn 1958. pg 161). For the software implementation, the user has a hyperparameter ‘word_dist=4’ that allows them to select the maximum number of non-significant words aloud in-between significant words.

The sentences are assigned scores by counting the number of significant words in the bracket and dividing it by the square of the total number of words in the bracket. The highest score for each sentence is that sentences final significance score. Using the sentences and their significance score, a set of the best scoring sentences can be selected to make up the summary.

ORIGINAL

Bad weather dashed hopes of attempts to halt the flow during what was seen as a lull in the lava's momentum. Experts say that even if the eruption stopped today, the pressure of lava piled up behind for six miles would bring debris cascading down on to the town anyway. Some estimate the volcano is pouring out one million tons of debris a day, at a rate of 15ft per second, from a fissure that opened in mid-December. The Italian Army yesterday detonated 400lb of dynamite 3,500 feet up Mount Etna's slopes. (Clarke and Lapata 2007. pg 4) (88 words)

ABSTRACT

Bad weather dashed hopes of attempts to halt the flow during what was seen as a lull in the lava's momentum. (0.2) Experts say that even if the eruption stopped today, the pressure of lava piled up behind for six miles would bring debris cascading down on to the town anyway. (0.2) (50 words)

SIGNIFIGANT WORDS

Lava, today, debris, flow, rate, seen

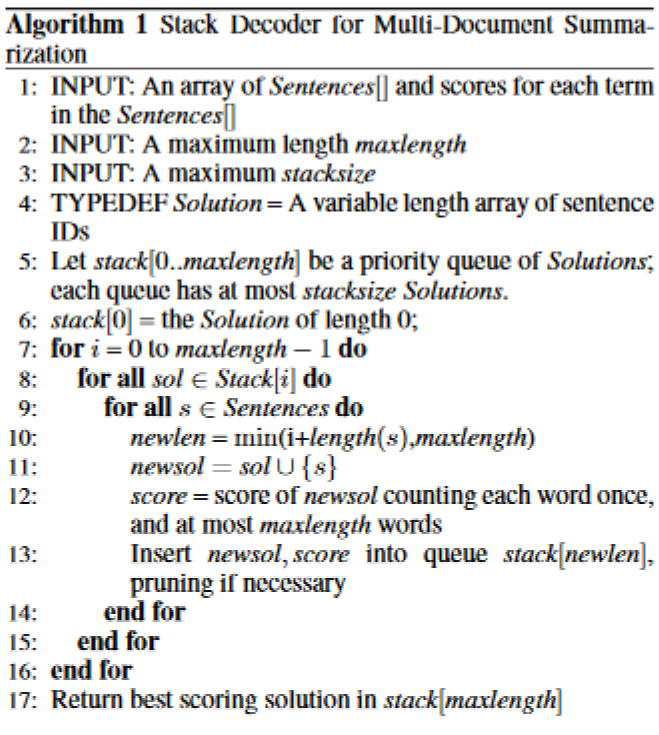
word_dist=4 | non_similar_threshold=6 | min_freq_threshold=2 | percent_to_retain=0.4

The second comes from a paper titled "Multi-Document Summarization by Maximizing Informative Content-Words" (Yih, Wen-tau et. al 2007). This paper extends the word frequency model by also considering the position of the words while scoring. The researchers were trying to find other sources of information that could help improve machine generated summaries. They looked at many properties "including capitalization, word length, sentence length, and others" (Yih, Wen-tau et. Al 2007. pg 1777). They found no significant difference between human and machine-based summaries in all these properties except for word position. The authors found that summaries that favored words that appeared near the beginning of the document and even that "a simple baseline system that takes the first l sentences as the summary out-performs most summarization systems in the annual DUC evaluation” (Yih et. al 2007. pg 1777).

To combine position and frequency, the software implementation uses the “Generative Scoring” method described in the paper. During execution, the model flips a biased coin to choose whether to score the term based on all the terms from every document in the document cluster, or only from the beginning of documents in the cluster. The researchers found that a bias of 0.5 worked best (Yih et. al, 2007. Pg 1777). The software implementation allows the user to adjust the bias as a hyperparameter where a higher bias makes the algorithm more likely to select from the beginning of documents.

Figure source (Yih et. al, 2007. Pg 1779)

To select a combination of sentences to make up the summary, a unique stack decoder method is used by the researchers. The algorithm uses size-limited priority queues to hold solutions. *maxlength* priority queues created; Each holding *stacksize* possible solutions. Starting from an empty string, the algorithm tries to extend the current solutions with every sentence. These combinations are scored using the coin flip scoring method and are added to the priority queue that matches the length of the solution.



Algorithm 1 Stack Decoder for Multi-Document Summarization

```
1: INPUT: An array of Sentences[] and scores for each term in the Sentences[]
2: INPUT: A maximum length maxlength
3: INPUT: A maximum stacksize
4: TYPEDEF Solution = A variable length array of sentence IDs
5: Let stack[0..maxlength] be a priority queue of Solutions; each queue has at most stacksize Solutions.
6: stack[0] = the Solution of length 0;
7: for $i = 0$ to $maxlength - 1$ do
8:     for all $sol \in Stack[i]$ do
9:         for all $s \in Sentences$ do
10:            newlen = min(i+length(s),maxlength)
11:            $newsol = sol \cup \{s\}$
12:            score = score of newsol counting each word once, and at most maxlength words
13:            Insert newsol, score into queue stack[newlen], pruning if necessary
14:        end for
15:    end for
16: end for
17: Return best scoring solution in stack[maxlength]
```

Using this method, it is possible to create sentences longer than *maxlength*. If the current solution's length is *maxlength*-1 and a sentence with length L is added the solution will have length (*maxlength*-1) + L. This is handled by the scoring function. The first *maxlength* words are considered in scoring and the remaining ignored. As well, as the sentence tries to extend each solution with every sentence, the same sentence could be added twice. This is also handled by the scoring function. During scoring each word is only considered once so a duplicate sentence provides 0 score increase.

ABSTRACT

BAD WEATHER DASHED HOPES OF ATTEMPTS TO HALT THE FLOW DURING WHAT WAS SEEN AS A LULL IN THE LAVA'S MOMENTUM. EXPERTS SAY THAT EVEN IF THE ERUPTION STOPPED TODAY, THE PRESSURE OF LAVA PILED UP BEHIND FOR SIX MILES WOULD BRING DEBRIS CASCADING DOWN ON TO THE TOWN ANYWAY. (50 WORDS)

beginning_words=50 | maxlength=50 | bias=0.5

The third is a method proposed by researchers from the University of Edinburgh; The paper titled "Modelling Compressing with Discourse Constraints" (Clarke and Lapata 2007) proposes a method of text compression using sentence compression. Out of the methods looked at so far, this method is much more complex. One issue with sentence compression in the words of the researchers is "compression is performed on isolated sentences without taking into account their surrounding context" (Clarke and Lapata 2007. Pg 1).

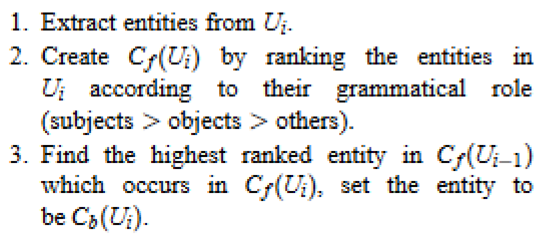
The authors states "Although an utterance in discourse may contain several entities, it is assumed that a single entity is salient or 'centered'" (Clarke and Lapata 2007). Each sentence has a list of forward-looking-centers. These nouns are used to determine the unique backward-facing-center of the next sentence.

The authors propose a two-step pipeline to generate these centers. First, names entity recognition is performed to identify all named entitles. Next coreference resolution is performed to avoid having multiple forward-looking-centers refer to the same entity.

The software implementation uses NLTK (Bird, Loper and Klein 2009) for named entity recognition. First, the data split into sentences by splitting the text on every period. Next, each sentence is tokenized using nltk.work_tokenize. This method uses a treebank word tokenizer along with a punkt sentence tokenizer to efficiently tokenize an input sentence. This tokenized data is used with NLTKs pos tagger to identify the most probable part-of-speech that token is fulfilling. The name entity chunker then uses this part-of-speech information to identify the named entities. Given input 'Army' the system outputs (ORGANIZATION Army/NNP).

Before writing this paper, I had no idea what coreference resolution was or how it was performed. The authors used LingPipe but that is unfortunately implemented in Java. Doing some research for an alternative that could run in Python, I came across Neuralcoref 4.0 (Huggingface Inc. 2018). Nerualcoref is a neural network based coreference pipeline that is capable of accurately finding coreference clusters given a document. Neuralcoref is not a standalone program but an extension for a popular machine learning library spaCy (Honnibal, Montani, Van Landeghem and Boyd (2020)). Using spaCys' model of the English language "*en_core_web"* an "English pipeline optimized for CPU. Components: tok2vec, tagger, parser, senter, NER, attribute_ruler, lemmatizer." (Honnibal, Montani, Van Landeghem and Boyd (2020)). Three versions of this model are available. en_core_web_sm, en_core_web_md and en_core_web_lg. These models get successively larger and therefore might result in more accurate coreference at the cost of time.

Figure source (Clarke & Lapata 2007. Pg 4)



1. Extract entities from U_i.
2. Create $C_f(U_i)$ by ranking the entities in U_i according to their grammatical role (subjects > objects > others).
3. Find the highest ranked entity in $C_f(U_{i-1})$ which occurs in $C_f(U_i)$, set the entity to be $C_b(U_i)$.

With NER and coreference resolution complete, we now have a list of nouns to build the forward and backward facing centers. A method is proposed called the ”Centering Algorithm” (Clarke and Lapata 2007. pg 3). All nouns occurring in a sentence are added as forward-looking-centers. At each step, the highest ranked noun from the previous sentence that also occurs in the current sentence is selected as the backward facing center.

To extract more discourse information, a lexical chain analysis is performed on the document. The authors describe a method of creating lexical chains using a method from researchers Michel Galley and Kathleen McKeown from Colombia University. Upon reviewing this paper, the implementation was tedious and ultimately out of scope of this paper. Instead of giving up on this method while being so close to completion, I decided to see if I could find a solution online. I found paper with theory and code by a graduate student from the Illinios Insitute if Technology Ana Maria Martinez Sidera. I decided to study and implement their method of lexical chaining as it was much simpler than the proposed version but still yielded useful results.

Now, this data must be used in some way to find the significant words in the given document. This is completed by using a series of constraints to assign either a 1 or 0 to each word. There are various constraints discussed by the researcher in the paper. Some of these such as the Sentential Constraints, I could not get working in my implementation. Therefore, I chose to implement the ones that I could figure out. Firstly, a constraint was placed on the backward-facing-centers. If a backward-facing-center was retained for any reason, it will also be retained in the next sentence if it exists. Next, a constraint is placed on the lexical chaining. If a word in a chain is retained, so are all other words from that chain.

ABSTRACT

Bad weather attempts to halt the flow during what was seen as a lull in lava's momentum. even if the eruption stopped lava six miles debris town anyway. the volcano one million tons of debris a rate 15ft per fissure that opened in mid December. The Italian Army yesterday 400lb of dynamite 3,500 feet up Mount Etna's slopes. (58 words)

greedyness=.564

References

Luhn, H.P. (1958) The Automatic Creation of Literature Abstracts. IBM Journal of Research and Development, 159-165. doi: 10.1147/rd.22.0159

Yih, Wen-tau & Goodman, Joshua & Vanderwende, Lucy & Suzuki, Hisami. (2007). Multi-Document Summarization by Maximizing Informative Content-Words. Proceedings of IJCAI. 1776-1782.

Clarke, James & Lapata, Mirella. (2007). Modelling Compression with Discourse Constraints. 1-11.

Sidera, Ana Maria Martinez. (2018). Lexical Chains. 1-3. Retrieved from https://github.com/amsidera/Lexical-Chains

Bird, Steven & Loper, Edward & Klein, Ewan. (2009) Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit [Software]. Retrieved from https://github.com/nltk/nltk

Honnibal, Matthew & Montani, Ines & Van Landeghem, Sophie & Boyd, Adriane. (2020). spaCy: Industrial-strength Natural Language Processing in Python [Software]. Retrieved from https://github.com/explosion/spaCy. doi: 10.5281/zenodo.1212303

Huggingface Inc. (2018). Neuralcoref [Software]. Retrieved from https://github.com/huggingface/neuralcoref