Document Summarization of Biological Articles

Leslie Manrique
ljm391@nyu.edu

Abstract

Abstracts are very helpful to researchers when searching for useful information to accomplish tasks. Such tasks can be for academic or recreational purposes. However, often times we have to read very lengthy articles in order to obtain the information that we need. Reading these articles can also be in vain, if the information is not found. We need abstracts or summarizations to realize ahead of time that the article is suited for in depth study. I approach the issue of creating automatic summarizations from text files by applying TF-IDF (term frequency – inverse document frequency). Using TF-IDF I segment the document into sentences and calculate the TF-IDF of each token. Afterwards, I sum up the TF-IDF's of each individual token per sentence. I compare the total TF-IDF's of each sentence to each other – thus extracting the most important sentences. The Idea is to find out if TF-IDF alone produces good summaries, when compared to human summarizations.

1 Introduction

Due to the internet revolution, there are terabytes of information that exist on the web. It can become overwhelming to search through all the information to find what you are looking for. As a student, you have to juggle a lot of different courses and your times is very valuable. Automatic document summarization can thus be a great tool to save someone some time. Automatic document summarization produces a truncated representation of a document that represents the most important information. Different methods are still being researched in order to improve overall efficiency. are different approaches to creating document summarizations. There are two general approaches to producing an automatic document summarization: extractive and abstractive.

Extractive based document summarizations extract important segments of a document, without manipulating the structures of the segment. This could mean to find the most important sentences of a document and output them in order. To produce, this will require statistical methods in order to identify what sentences are important, such as TF-IDF. Abstractive based document summarization, on the other hand, relies on natural language generation. It gathers important information from the documents, and outputs them as a machine representation of what natural language is. Basically, it generates a cohesive representation of the important segments of the document.

The approach I use for my automatic document summarizer is an extractive based document summarization because generating an abstractive based summarizer would require me to create a system that creates well structured sentences and is under a lot of research.

2 Corpus

I used CRAFT: The Colorado Richly annotated Full Text Corpus. CRAFT provides full peer reviewed journals that are annotated. Texts are available in both xml and .txt format. It's also annotated using Penn Treebank tags. [1] [2]

3 Term Frequency – Inverse Document Frequency (TF-IDF)

TF-IDF is an information retrieval (IR) tool used in order to treat documents as a bag-of-words ignoring syntax, meaning and ordering [3]. The TF-IDF is a weight given to a word in order to observe it's relevant importance with respect to the rest of the corpus.

3.1 Term Frequency (TF)

Term frequency describes the occurrence of a word within a single document. It's name say's it all: the frequency of a term or word in a document.

$$tf(t) = \#\ of\ times\ term\ appears\ in\ doc$$

3.2 Inverse Document Frequency (IDF)

The inverse document frequency is what measures the importance of the terms. This means that words that are frequent such as “to”, “from” … are seen as relatively unimportant with respect to the collection of documents. Higher weights will be given to words that appear more infrequently. Therefore, the fewer documents in which a term occurs, the higher it’s weight [3]. The lowest weight term can get is one, due to the logarithmic component.

$$idf(t) = \frac{total\ number\ of\ documents}{number\ of\ documents\ with\ t}$$

3.3 TF-IDF

TF-IDF weighting is finding the product of tf(t) with idf(t). The result is greater for terms that are frequent in the current document but infrequent in the collection of documents.

$$tfidf(t) = tf(t) * idf(t)$$

4 Method

Creating an extractive test summarization, require you to be able to find what sentences are important in a document. You have to answer the questions – what features, show that this sentence is important? Also, there needs to be a way to score the sentences based upon. I used Term Frequency – Inverse Document Frequency (TF-IDF) in order to compute the importance of sentences and generate an automatic document summarization.

4.1 Loading Documents

4.1.1 Document Collection from Corpus

In order to calculate the IDF values of the corpus I used beautifulsoup to extract the sentences and their tokens rather than using a tokenizer. This way, I could make sure that the terms of the document were accurately segmented.

4.1.2 Test Files

To calculate the TF scores of the test document, I couldn’t use the same approach as I did with the collection of documents. This is because, I wanted to be able to eventually create an interface where you could input a text file and the code will generate an output that is the automatic summarization. To do so, I would need use a tokenizer. I used NLTK sentence tokenizer and NLTK tokenizer to obtain the individual terms in the corpus.

4.1.3 Stop words

Filtering the stop words from TF-IDF score computation is necessary because we need to obtain a list of important words. Stop Words are so common in English vernacular that they are not important, such words are “to”, “from”, “a” …. Filtering out these stop words prevents interference when we add up the TF-IDFs scores in the sentences. You are also closer to a list of the most important words.
I used NLTK’s list of stop words in English. Also, I added numerical values such as digits and floating points as well as punctuation to the stop words list. TF and IDF is computed without interference from stop words.

4.2 Why TF-IDF?

Using TF-IDF is useful because obtaining the score shows the relative frequency of a word in based upon the entire collection of documents. In other words, it measures how important the word is to a document by comparing with a collection of other documents.

4.3 Sentence Scoring

To obtain the importance of a sentence, I calculated the TF-IDF scores of each word in the sentence. Reminder, the sentences here are tokenized and do not include stop words, as mentioned prior. I summed up the TF-IDF scores in each sentence. Then I divided the total score by the length of the sentence. Dividing by the length of the sentence, allowed me to account for the fact that some sentences are relatively longer than others. Dividing eliminates length from being a factor in importance. I then sorted the document sentences by their total sentence weight, in reverse order (from greatest to lowest weight).

5 Evaluation

Tests were performed on seven test documents. Meaning, I outputted the different summarizations of only seven documents in my total set. These test documents were also found in the document collection set, for computing the idf values. Summarizations were generated for more, but due to time constraints, I was only able to perform calculations on seven.

5.1 Top n words

In my system, I test if a score is more precise given lengths of summaries. I test 4 different lengths each given in percentage of document: 5%, 10%, 15% and 20%. "n" in Top n words is replaced by the given percentage of documents.

$$n = length(document) * Percent$$

The top n words are outputted as a text file in a folder called "Summaries" in the PythonROUGE directory for each percentage to be evaluated later on.

5.2 Abstracts

I had difficulties obtaining the abstracts from the documents using beautifulsoup. When I cleaned the files xml, there was a lot of formatting issues. I manually got the abstracts from the test documents, as craft also provides them in non xml, plain text format.
Some of the test files contained synopsis as well as abstracts, so I created two reference files for them to test my system on.

5.3 What is ROUGE?

For system evaluation, I used ROUGE which stands for Recall Oriented Understudy for Gisting Evaluations developed by the Information Science Institute at the University of Southern California. It is a tool that is used to evaluate automatic summaries from a system with reference summaries. In my approach, I used abstracts from journals as my reference summaries.

5.3.1 ROUGE-N: N-gram Co-Occurrence Statistics

Note: This description obtained from reference [4]
ROUGE-N is an n-gram recall between a candidate summary and a set of reference summaries. ROUGE-N is computed as follows:

$$\text{ROUGE-N} = \frac{\sum_{S \in \{ReferemceSummaries\}} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S \in \{ReferenceSummaries\}} \sum_{gram_n \in S} Count(gram_n)}$$

Where n stands for the length of the n-gram, gramn, and Countmatch(gramn) is the maximum number of n-grams co-occurring in a candidate summary and a set of reference summaries. [4]

5.4 Evaluations using ROUGE

Rouge computes recall, precision and f-measure scores that I used to evaluate my system. I outputted the average of these rouge scores per document length percentage.
I outputted scores using both bigrams and unigrams rouge calculations.

5.4.1 Precision

The precision is the parentage of n grams in the automated summaries that also occur in the provided human summaries.

5.4.2 Recall

The recall is the percentage of n grams in the abstracts or human generated summaries that I provided, that also occur in the automated summary.

5.4.3 F-Measure

F- Measure is also given by our ROUGE outputs. F-Measure is calculated using precision and recall scores:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

6 Results

I evaluated my system using rouge for each summarization length. Below you can observe the results average results for my test data. In my evaluations I will pay closer attention to Average recall measures.

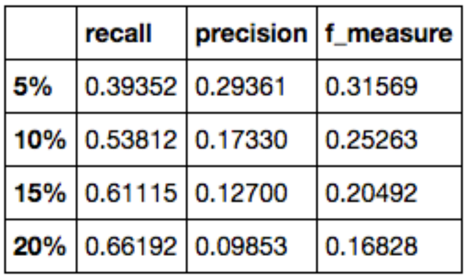
6.1 Average Bigram Scores (ROUGE-2)

	recall	precision	f_measure
5%	0.11532	0.09066	0.09567
10%	0.20973	0.06846	0.09972
15%	0.27753	0.06047	0.09699
20%	0.34463	0.05166	0.08820

According to both recall values, the bigram scores increase as you produce a larger summary. With precision, however, we have the opposite.

Recall is more important than precision when testing our system. This is because we want to know how much of the information from the abstract or human generated summary we have in our automated summary.

6.2 Unigram Scores (ROUGE-1)



	recall	precision	f_measure
5%	0.39352	0.29361	0.31569
10%	0.53812	0.17330	0.25263
15%	0.61115	0.12700	0.20492
20%	0.66192	0.09853	0.16828

For recall, Unigram scores did better than bigram scores. Meaning that more unigrams from the abstract appeared in the automated summary.

7 Conclusions

I can conclude that my system needs a lot of work. According to my recall scores, the percentage of the abstract bigrams that appeared in the automated summary was less than 50%. Unigram scores did a lot better, but bigrams would simulate a more cohesive natural language based summarization. Therefore, using TF-IDF alone to produce document summarizations would not provide you with the best summarization. Despite saying so, I could also do some improvements to my project.

One improvement I can do is to add a named entity recognizer. Using a named entity recognizer, I would be able to provide a better filter for important words at the beginning of my process. Currently, I am only filtering out the stop words which are provided by NLTK and also include numerical values and punctuation. If I have a named entity recognizer, I would only include in my sentence tokens, words that are named entities. A simpler approach could be to just filter out words that are not nouns.
Another improvement is to run ROUGE on more test data and include more human summarizations.

In my research, I came across various methods of document summarizations. One method used machine learning to summarize medical documents. They trained a system to recognize important sentences by ranking them as either "worthy", "moderately worthy" or "unworthy". They also used tf-idf along with cosine similarity measurements to reorder sentences in a document so information is not repeated. Given time, I will research more methods to improving my system.

References

[1] Bada, M., Eckert, M., Evans, D., Garcia, K., Shipley, K., Sitnikov, D., Baumgartner Jr., W. A., Cohen, K. B., Verspoor, K., Blake, J. A., and Hunter, L. E. Concept Annotation in the CRAFT Corpus. BMC Bioinformatics. 2012 Jul 9;13:161. doi: 10.1186/1471-2105-13-161. [PubMed:22776079]

[2] Verspoor, K.*, Cohen, K.B.*, Lanfranchi, A., Warner, C., Johnson, H.L., Roeder, C., Choi, J.D., Funk, C., Malenkiy, Y., Eckert, M., Xue, N., Baumgartner Jr., W.A., Bada, M., Palmer, M., Hunter L.E. A corpus of full-text journal articles is a robust evaluation tool for revealing differences in performance of biomedical natural language processing tools. BMC Bioinformatics. 2012 Aug 17;13(1):207. [PubMed:22901054]

[3] Jurafsky, Dan, and James H. Martin. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2000. Print.

[4] Lin, Chin-Yew. 2004a. ROUGE: a Package for Automatic Evaluation of Summaries. In Proceedings of the Workshop on Text Summarization Branches Out (WAS 2004), Barcelona, Spain, July 25 - 26, 2004.

[5] Sarkar, Kamal, Mita Nasipuri, and Suranjan Ghose. "Using Machine Learning for Medical Document Summarization." International Journal of Database Theory and Application 4.1: 31-48.

[6] Almeida, B Miguel, PythonRouge, 2012, Github Repository,https://github.com/miguelbalmeida/Python ROUGE