CS 297 Report

Experiments with and Implementation of a Context Sensitive Text Summarizer

By

Charles Bocage

Project Advisor: Dr. Chris Pollett

Department of Computer Science
San José State University
One Washington Square
San José, CA 95112

Table of Contents

Introduction

Text summarization is an important field in the area of natural language processing and text mining [Samei2014]. There is a vast amount (more than six billion websites) of information on the Internet and growing every day. This overload of information has lead us to be able to produce concise summaries of the information so we can process it. Text summarization is the ability to obtain the key ideas from a text passage using as little words as possible. In order to achieve a concise summary, one must break the summarizer into many modules, some of which are sentence segmenters, stemmers, stop word removers and term frequency computers. In this paper I experiment with storing the least amount of information to still be able to show the relevance of a given webpage. In other words, how good is the summary that is generated by the summarizer and can we apply some simple concepts to make the quality of the summarizer's content better.

This semester I worked heavily in the summarization arena doing various experiments and writing a fair amount of code. I compared the quality of the existing summarizers, worked with stemmers, worked with a new summarizer and modified one of the existing summarizers. This work was split into four deliverables. In the first deliverable, I got the Yioop search engine and the Recall-Oriented Understudy for Gisting Evaluation (ROUGE) software package working on my machine and the compared quality of the existing Basic (scrape) summarizer (BS) with the Centroid-Based summarizer (CBS). In the second deliverable, I created a stemmer for the Dutch locale and added it to the Yioop search engine. In the third deliverable, I created a new summarizer called the Graph-Based Summarizer. Finally, I experimented with adding an additional weight algorithm to the CBS.

Deliverable 1: Compare Basic Summarizer to Centroid-Based Summarizer using ROUGE

The goal of this deliverable was to compare the Yioop search engine's basic summarizer (BS) and content-based summarizer (CBS) results to human created summaries on a sample of web pages and report the results. The Yioop search engine has two summarizers. The BS and CBS. The BS grabs different parts of an HTML document in a fixed order until it has gotten to the limit of the number of allowed characters for a summary. The CBS uses a centroid (a set of words that are statistically important to the document) to get the main idea for the document. After that it uses text frequencies and cosine similarity to find the nearest sentence to main idea. The results were compared using the Recall-Oriented Understudy for Gisting Evaluation (ROUGE) software package, which as of now is the gold standard for calculating summarization metrics.

ROUGE uses various methods to calculate its metrics along with its Recall, Precision and F measures. The precision measure is the relevance of the retrieved documents, the recall measure is the relevance of the relevant documents and the F measure is a combination of both. Below are each method and a brief explanation:

• ROUGE-L: measures sentence-to-sentence similarity based on the longest common subsequence (LCS) statistics between a candidate translation and a set of reference translations (LinOch2004)

• ROUGE-S: computes skip-bigram co-occurrence statistics (LinOch2004)

• ROUGE-W: is an extended version of ROUGE-L. The only difference is ROUGE-W weights the LCS statistics and favors contiguous occurrences

• ROUGE-SU: is an extended version of ROUGE-S. ROUGE-SU considers skip-bigrams and unigrams, hence the addition of the U in the name

• ROUGE-N: is an N-gram recall between a candidate summary and the reference summaries. N is the length of the n-gram (Lin2004)

In order to generate the BS and CBS summaries I needed to install and perform an initial configuration of the Yioop search engine. Details on how to set up the Yioop search engine are covered on the seekquarry.com website. After I installed the Yioop search engine, I let it crawl the internet to generate an index that I could search. Creating an index is not needed for this deliverable. I mention it because I needed it to prove I had a working search engine. Now that my search engine was functional, I proceeded to select ten sample web pages for the experiment. The sample web pages were blog entries written for my CS 200W class. I figured they would be easy to summarize since I wrote them and they were not very long. After a few hours of carefully selecting the most important sentences from each blog entry, I used the Yioop search engine to generate both its BS and CBS summaries. Next I had to compare the BS and CBS summaries to my human summary. The ROUGE software package was used for this comparison. I was skeptical that I would be able to use it because I had to contact the developer to get it. Luckily he responded and I was able to start setting it up. To put it nicely, the setup was not straight forward. In short, ROUGE needs an input configuration file that tells the system what system generated files and human generated files to use. The system and human generated files must be in a specific html format with each sentence wrapped in an <a> tag. I will show an example in the results section. Once you have all of the pieces put together, you run it with various switches and review the output. I ran ROUGE using the recommended arguments in each test. In general the results for the BS (0.80587) were better than the CBS (0.76663) and they are very close.

Deliverable 2: Create a Dutch Stemmer for the Yioop Search Engine

The goal of this deliverable was to create a stemmer in Dutch that can be used in the Yioop search engine. Stemmers are used in Information Retrieval (IR) systems and used for natural language processing. Stemmers specifically for IR were first developed in 1968 by Julie Beth Lovins. Lovins ideas were so popular it spurred many other types of stemmers. Stemmers can classified into main types; truncating, statistical and mixed. A truncating stemmer does exactly what its name says. They remove the appropriate affixes (prefix or suffix) in a word. Statistical stemmers remove affixes based on statistical analysis like letter frequency. Finally, a mixed stemmer uses inflectual and derivation methods. In other words, the language syntax variations and the relations to the part of speech (POS) it comes from. More information can be found on this topic by reading the documents in the reference section.

The Dutch stemmer I worked on is a Truncating stemmer. It is based on the work of Martin Porter. In 1980, Porter presented a simple algorithm for stemming English language words [Willett2006]. Furthermore, with the help of Porter's website of the Dutch Stemmer I was able to start working on my Dutch stemmer. The language to write the stemmer in was chosen for me since the Yioop search engine is written in PHP. Now that I had my sudo code and programming language, I searched the internet to see what others had already done. I found the Simplicity Lab PHP Dutch Stemmer that claimed to be complete but it was not. I downloaded the code and started to take a look at it. It was not written in the format I would have liked so I the first thing I did was refactor it.

Next I ported it to the format needed for the code to run within the Yioop search engine. To see how good it was I ran it against the vocabulary words and stems provided by Porter's

website. To my amazement, it had tens of thousands of errors out of the 49000 plus words in the list. At this point I knew I was going to spend hours repairing the code I planned on using as my base. After may trials and errors the first smoking gun was the problems with the strtolower() and split() methods. The Dutch alphabet has characters with umlauts and accents and the strtolower() and split() methods cannot handle those types of characters. I changed the methods to the mb_strtolower() and preg_split respectfully. Next messaged the method that removes umlauts and accent characters.

There were some vowels that needed to be removed and some that needed to be added. After that method was complete, I had to add words to the no stem list and review each of the 5 steps (play with regular expressions) until the stemmer passed all tests. In addition to the stemmer, the Yioop search engine also has a configure.ini file used to translate static phrases presented on the user. There are over 1100 of them. This was no easy task either. I spent hours trying to figure out the best way to skin this cat and finally came up with a full proof way to do it. If you have ever opened an .ini file you know the settings are stored in a <setting> = <value> format. First I opened the configure.ini file with notepad. I replaced " = " with a comma and saved it to a .csv file. That allowed me to open the file with Excel for manipulation. Excel was able to give me a representation of the file that had the settings and values seperated in columns and select only the values from the file. I copied about 400 or so values at a time and pasted them into Google Translate and clicked Transate.

Once the words were translated, I copied the translations into the column next to the values. I repeated this process until all of the words were translated. I double checked a random sample of the words and I was confident the translations were correct. After all of that was done, I submitted an issue and a patch to the MantisBT repository manager (Dr. Pollett) for review.

Deliverable 3: Create a New Summarizer for the Yioop Search Engine

The goal of this deliverable is to find a paper that covers an algorithm used for summarization. Once I have found the right paper, the algorithm will be implemented in PHP and integrated into the Yioop search engine. To reiterate from my CS297 proposal, text summarization is the ability to obtain the key ideas from a text passage using as little words as possible. Here I will cover the new algorithm chosen, its results, the work that went into implementing it and the work getting it into the Yioop search engine. My first objective was to find a good summarizer to implement. I felt it had to be something completely different compared to the existing summarizers in the Yioop search engine. The existing summarizers are covered in Deliverable 1. I came across a worthy algorithm known as Luhn's algorithm. Dr. Pollett did not feel the same way, so I searched further. I eventually stumbled upon a graph based algorithm that looked promising. This time my summarizer suggestion passed Dr. Pollett's evaluation and I proceeded to dig into it.

Furthermore, in order to prove I understood the algorithm, Dr. Pollett had me create a short PowerPoint presentation summarizing the algorithm. Once the presentation was complete, Dr. Pollett and I discussed the presentation thoroughly. After the discussion, we both felt we had a good handle on the algorithm and what it would take to implement it. For our own sake we also calculated the order in which the algorithm ran. "The algorithm is $O(n^2)$ not counting the outer while loop / for loop" (Dr. Pollett). Now for the details on the Graph Based Ranking Summarizer I implemented. I will first describe the algorithm as it is stated in the paper Multi-Document Summarization Using Graph-Based Iterative Ranking Algorithms and Information Theoretical Distortion Measures to give more detail than what is in the 8 slide presentation. Next I will detail the changes made in our implementation and what I had to do to add the algorithm to the Yioop search engine plus I will showcase the ROUGE results of the new summarizer.

The Graph Based Ranking Summarizer uses and extraction based model. It constructs a weighted graph out of the text where the sentences are the nodes. Now let us talk about our implementation. The first thing we changed was filtering non-ASCII characters. Dr. Pollett and I felt, since the Yioop search engine has many locales (with stemmers and stop word removers including ones like Dutch that include non-ASCII characters), we can support them. The other deviation is how we chose to weigh the sentences. It was unclear how the sentences were weighted so we decided to normalize the frequency values and not weight them further.

Once the code was written for the Graph Based Ranking summarizer (GBS), it had to be integrated into the Yioop search engine. Thanks to a tip by Dr. Pollett of how to search the code base, I was able to find all of the places I needed to inject code to get the summarizer integrated. First I ran "php code_tool.php search .. centroid" from the Yioop repository bin directory. It found all of the places the word centroid was located in the code base that may or may not have applied to my objective. As I took a look at each line, I either added the appropriate code or left it alone. I ended up modifying fetcher.php, crawl_component.php, crawl_constants.php, graph_based_summarizer.php, phrase_parser.php, html_processor.php, page_processor.php, text_processor.php and the English locale configure.ini file. When I tried my new summarizer for the first time I had errors all over the screen. I started adding echo lines throughout the code, spending hours tweaking the code until the errors were gone. In order to find some of the hidden hooks, I walked the stack trace backwards. There were some places that could not be found by text searches because they were loaded dynamically. Once I was confident the code was ready, I submitted an issue and a patch to the Mantis BT repository manager (Dr. Pollett) for review. Hopefully this summarizer will make it into the Yioop search engine also.

Deliverable 4: Term Frequency Weighting in the Centroid-Based Summarizer

The goal of this deliverable was to implement Dr. Pollett's summarizer algorithm. Once the algorithm is implemented, I will test the algorithm using ROUGE and compare its results to the other summarizers. This deliverable is an extension of Deliverable 1 and Deliverable 3. Here I will cover Dr. Pollett's algorithm and its results. Before any code could be written, Dr. Pollett and I needed to discuss his algorithm. His idea is to add a weighting algorithm to the centroid-based summarizer in order to improve its results. In short if you increase the frequency you increase the weight. First, we needed to define how the terms will be weighted and what we will do with the additional weight values. For example, if the term is in an h1 (heading) tag we would add some numerical weight to that terms frequency. We also discussed weighting terms based on if they appeared in the beginning, middle or at the end of the document. Furthermore, we discussed looking at the sentence length of the sentence the term was found in and increase its weight that way.

After we had all of our ideas on the table, we decided to go with weighting by tag idea. I found two documents on the internet; Using the Structure of HTML Documents to Improve Retrieval and Semantic-Sensitive Web Information Retrieval Model for HTML Documents that looked promising. Their idea is to organize certain html tags into classes and weight them according to a hierarchy. Using the Structure of HTML Documents to Improve Retrieval mentioned that it used 6 classes of html tags. The first class contains only the A (anchor) tag. The second class contains the H1 and H2 heading tags. The third class contains the H3, H4, H5 and H6 heading tags. The fourth class contains the text emphasizing and list tags. They are

STRONG, B, EM, I, U, DL, OL and UL tags. The fifth class contains the TITLE tag and the sixth class is everything else. Terms that fall through to the sixth class gets no additional weight.

Now that we know how we planned on weighting the terms, we will get into what the weighting algorithm will do to get the new weight. Instead of having a hierarchy like in the paper Using the Structure of HTML Documents to Improve Retrieval, all class occurrences will be counted. Furthermore, the weight of each tag will be multiplied by its term frequencies in that tag and added up. In other words, the weight of a term is the sum of all weights multiplied by its respective frequency in that tag. Lastly, we needed to consider the efficiency of this algorithm. Dr. Pollett made it clear that the current algorithm is pretty slow and my weighting algorithm should not slow it down any more than it needs to.

After reorganizing the code, I finally started writing the code for the additional weighting algorithm. Although I have used regular expressions for some time, I struggled with them in this implementation. It took a couple of weekly meetings before we had it nailed down. The problems were stemming from the regular expressions ability to count multiple occurrences of the same term in a tag. We decided to take a performance hit for quality results. After the regular expression found a match, the substr_count() method would be used to count the term frequencies in that tag. In order to prove my code would perform correctly, I wrote unit tests.

Lastly, I needed to regression test the additional weighting algorithm to expose optimal weights that I may use when making the algorithm live. I took what I did with the unit test framework to automate the entire process of updating the code with some weighting value, generating a summary for my 10 test documents, formatted the output for ROUGE, ran ROUGE on the results and repeated the process for some limit. This may sound east but it was not. I was able to create such a system that did one class weighting value at a time and tested its results.

Conclusion

In conclusion, I feel the work I have done barely scratches the surface because there is a lot of work to be done in the text summarization environment. This paper covered most of the basic experiments needed in order to understand text summarization and what some others have done previously. For example, being able to use ROUGE to test the quality of a summarizer, understanding what a stemmer is and why it's important, reviewing the details of a non-standard (graph-based) approach and trying to improve on an existing idea will prepare me for the work to come in the next semester. I am proud of the work I have done here and cannot wait until I complete my experiments. Maybe we will publish something the next students of summarization can use to build their research on.

For next semester we want to do experiments that will generate a paper and get the paper done well before graduation. In order to accomplish that goal we will continue to get deeper into text summarization. First, we want to compare a few more summarizer methods. One of which is different than any that others that have been looked at so far. For example, a graph summarizer (which I already did), the centroid summarizer with weights and look into the famous summarizer Nick D'Aloisio wrote in the Summly app the he eventually sold to Yahoo for $30 million. In addition, we want to do more extensive experiments. For example, using an existing set of summaries and documents like in the Document Understanding Conference (DUC) 2002 data set. Using a larger data set will allow us to really determine how good the summarizers we test perform.

References

[LinOch2004] Automatic Evaluation of Machine Translation Quality Using Longest Common Subsequence and Skip-Bigram Statistics. Chin-Yew Lin and Franz Josef Och. Association for Computational Linguistics. 2004.

[Lin2004] Looking for a Few Good Metrics: Automatic Summarization Evaluation How Many Samples Are Enough?. Chin-Yew Lin. NTCIR. 2004.

[Lovins1968] Development of a Stemming Algorithm. Julie Beth Lovins, Mechanical Translation and Computational linguistics. 1968.

[Porter1980] An algorithm for suffix stripping. M.F. Porter. Program: Electronic Library and Information Systems. 1980.

[Willett2006] The Porter stemming algorithm: then and now. Peter Willett. Program: Electronic Library and Information Systems. 2006.

[Porter2001] Snowball: A language for stemming algorithms. M.F. Porter. Snowball. 2001.

[Samei2014] Multi-Document Summarization Using Graph-Based Iterative Ranking Algorithms and Information Theoretical Distortion Measures. Borhan Samei, Marzieh Eshtiagh, Fazel Keshtkar and Sattar Hashemi, The Florida AI Research Society. 2014.

[Agrawal2014] A Graph Based Ranking Strategy for Automated Text Summarization. Nitin Agrawal, Shikhar Sharma, Prashant Sinha, Shobha Bagai, DU Journal of Undergraduate Research and Innovation. 2014.

[Langvielle2006] Google's PageRank and Beyond: The Science of Search Engine Rankings. Amy Langville and Carl Meyer.Princeton University Press.2006.

[Cutler1997] Using the Structure of HTML Documents to Improve Retrieval. Michal Cutler, Yungming Shih and Weiyi Meng. Proceedings of the USENIX Symposium on Internet Technologies and Systems. 1997.

[Bassil 2012] Semantic-Sensitive Web Information Retrieval Model for HTML Documents. Youssef Bassil and Paul Semaan. European Journal of Scientific Research. 2012.