

# Semi-supervised Document Summary

Bobby Dorward and Ryan Wilson

December 12, 2016

if you want to leave notes to me use this command to make it red

## 1 Introduction

In our final project, we looked at the problem of single-document summarization. That is, we attempted to write a program to read in a document, and output a short summary of the document. Our summaries are general summaries rather than query-based summaries, and we generate extracts rather than abstracts for our summaries (meaning that our summaries are made using specific informative sentences extracted from our document, rather than being fully self-generated using the extracted raw information).

Since summarization is a complex task, it poses a particularly interesting natural language processing problem. For a human, summarization requires a moderately complex understanding of a document; critical information must be understood and prioritized, combined into a cohesive set of concepts, then put into words in a way that will both maintain the meaning of the high-priority concepts, and make sense in terms of the rules of the language in which the summary is written. These steps are modified, of course, by probabilistic approaches that simplify the nature of each of these problems.

We consulted the Jurafsky & Martin textbook, specifically Chapter 23, entitled *Question Answering and Summarization*, as well as the article *A Statistical Approach for Document*

*Summarization* from the Department of Computer Engineering at the Fr. C. Rodrigues Institute of Technology. Chapter 23 of the textbook discusses multi- and single-document summarization and gives a brief discussion of different ways to solve different subproblems of the summarization problem. The article mentioned above attempts to accomplish its own specific task (query-based abstract summarization based on a given query, generated using two or more web documents, which are selected based on the provided query). We consider their techniques and alternative modified approaches to the problem formulated by ourselves throughout the course of the project, as well as their results, to learn more about the nature of these algorithms and how to further refine our own approach.

## 2 Implementation

Document summarization can be broken down into 3 subproblems; content selection (extracting summary-worthy material), information ordering (putting together the selected content), and sentence realization (?cleaning up? the ordered information such that it is concise, cohesive, and sensical). For content selection, we start by assigning each sentence in the document an ?information content? score. Then, we rank these sentences by information content, and finally select the top X% of ranked sentences, where X is a parameter set in our code (10% by default).

The first step is content selection, selecting sentences from the document to extract. To achieve this goal, we used a semi-supervised approach recommend by Jurafsky and Martin. There are two main ideas to this approach. The first is that we can think of sentences as bags of words, vectors in  $n$ -dimensional space, where  $n$  is the number of unique words in our document  $d$ . The document is then a cluster and a sentence is worthy of being extracted if it is near the center of the cluster. In order for this approach to work, we need a measure of distance between sentences, which brings us to the other main idea of the method. We want to weight certain words in our sentence as more important than others based on two

criteria. The first is that words that appear very frequently in the document are probably very important to the document. This is the *term-frequency* measure

$$tf(w) := \text{number of occurrences of word } w \text{ in } d.$$

However, lots of words appear frequently without contributing any content. Some examples are articles, prepositions and other structural words. Therefore, we also weight words by whats called the *inverse document frequency*

$$idf(w) := \log \left( \frac{N}{N_w} \right),$$

where  $N$  is the total numbers of documents in our corpora and  $N_w$  is the number of documents in the corpora that contain  $w$ . A nice feature of the *idf* is that it deals with stop words. If a word is in every document (e.g., “the”) then

$$idf(the) = \log(1) = 0$$

and the word is not included in our estimates. We use these functions to create a feature vector  $v_s = (v_1, v_2, \dots, v_n)$  for each sentence  $s$ , where

$$v_i = \begin{cases} tf(w_i)idf(w_i) & \text{if } w_i \in s \\ 0 & \text{otherwise.} \end{cases}$$

We then can define our distance, which is really a measure of similarity because we want to choose sentences with a large distance from other sentences. The distance Jurafsky and Martin recommend is the cosine distance

$$tf-idf-cosine(s_1, s_2) := \frac{v_{s_1} \cdot v_{s_2}}{\|v_{s_1}\| \|v_{s_2}\|}.$$

put the bad summary example here, reference it by using \ref{badsummary}

Figure 1: put the caption here

put the good summary example here, reference it by using \ref{goodsummary}

Figure 2: put the caption here

With our distance function we can define a measure of how central a sentence is to our cluster, which is just the average distance to all other sentences in the document

$$centrality(s) := \frac{1}{|d|} \sum_{s' \in d} tf - idf - cosine(s, s'),$$

where  $|d|$  is the number of sentences in  $d$ . Thus, a sentence is good for our summary if  $centrality(s)$  is large relative to the other sentences in the document. Therefore, we compute the centrality of each sentence and choose the best  $X\%$  of sentences in  $d$ , where we currently have the parameter  $X = 10$ , though this could be changed depending on the desired summary length. These best sentences form the extract summary.

Next, we implemented information ordering. Standard practice for this is very simple—most often, sentences from such a summary are simply ordered in the same relative order they had in the original document. However, one of our goals that developed as we worked on the project was to experiment with slightly different approaches. We did one such experiment with information ordering. We attempted to use an n-gram approach to look for higher-probability sentence transitions, i.e. we attempted to slightly rearrange sentence ordering to reflect sentences to accommodate any sentence transitions which, according to our n-gram approach, are likely to appear. The intuition behind this approach was that some sentence endings and beginnings have meaning that directly relates to the progression of ideas; for example, the word “however” is often used in the beginning of a sentence to indicate a counterexample or contrasting statement to the sentence before. Therefore, if we have a “sentence transition n-gram”, meaning an n-gram that includes words from two different sentences, that is “... death penalty. However, ??”, and we have two sentences in

our summary: "There was a great deal of support for the death penalty." and "However, a new study indicates that the death penalty is less humane than previously anticipated?", we might want to put those two together. Since there is a logical progression from the first sentence to the second, and this is reflected by the structure of the n-gram transition, we hope that making small adjustments to the ordering of the sentences based on these n-gram transitions would yield a more sensible and logical progression of ideas.

Finally, we have the problem of sentence cleaning. For this problem, we attempted to create a new implementation using only minimal information on the algorithms commonly used to solve the problem. We knew that parse trees were usually used to accomplish this, short of the use of complex semantic analysis that is beyond the scope of this project. So, using this information, we set out to find our own way to prune sentences. Concluding that parse tree-based pruning would best be accomplished by weighting and pruning subtrees from our parse tree, we used that as the basis of our algorithm. Initially, we simply used the information content of our initial scoring, applying those as weights at the lexical level of the parse tree and weighting subtrees by averaging the

### 3 Data

The corpus that we use for the training of all parts of our algorithm (and to find documents for summarization) in the final version is the Open American National Corpus (a.k.a. the OANC; we use files in Graph Annotation Format). This corpus consists of about 9,500 articles on a variety of different subjects, of lengths usually on the order of a few dozen paragraphs.

Initially, we intended to use a variety of different genres of document, so to speak, and train only on documents of the same category when summarizing a given document. However, as we developed our own approach to the problem, namely our modification of the sentence ordering subproblem, we decided it would be more pertinent to use all of the documents at our

disposal to train on. For starters, we concluded that the logical progression of ideas that are hopefully represented by sentence transition n-grams would be the same regardless of genre. Specifically, we felt that the structural meaning of words like ?however? or ?additionally? would be the same in any English document. However, as we will discuss when we analyze our results, sparseness of data for sentence transition n-grams also played a part in our decision to stick to training on all of our data at once.

Prior to our final implementation, we also tried using the Reuters Corpus, which consists of nearly all articles ever published on reuters.com. However, this corpus was much smaller than the OANC, and as a result was thrown out. We also considered using the Penn Treebank, prior to discovering the cost, and we also considered using Project Gutenberg. However, Project Gutenberg does not have a single centralized corpus that is available to the public, and we also decided that novels would not be the most useful tests of our algorithm, given their length. So, we finally settled on the OANC, and it has proven successful for our purposes.

## 4 Results

Given that document summarization is an unsupervised learning problem, it was particularly difficult to evaluate the ?success? of our program. However, we did have 2 different methods for evaluation in mind. First we have ROUGE, a program built by a USC grad student which provides a measure of how ?good? a summary is. We acquired this program and installed it on my laptop, but upon inspection of Chin-Yew Lin (the creator of ROUGE)’s paper ROUGE: A Package for Automatic Evaluation of Summaries, we realized that ROUGE was made for the evaluation of abstract rather than extract summaries, and under the hood it operated using an n-gram model. Since we use sentences copied (albeit somewhat edited) from the original document, this method would be confounding and produce poor evaluation results.

We also considered a process by which one of us writes questions about a document, while the other one writes a summary. We would then randomly assign some human subjects either a human-made summary or a summary assembled by our program, and pose the questions that were written about that document. We would look at the the accuracy of their responses for both the human- and computer-generated summaries and compare the two. However, given the task of assembling a meaningful number of human subjects on which to run this test is outside of the scope of this project.