sumMary: Generic Text Summarization with Sentence Disambiguation

https://github.com/arberx/sumMarize

Arber Xhindoli axhindol@umich.edu
Dillon Kesto dkesto@umich.edu
Caroline Saab ginasaab@umich.edu
Trevor Rees tjrees@umich.edu

Abstract

These days, the quantity of available information on any given topic is increasing at an incredible rate. At the same time, people are finding themselves with much less time to gather information on topics they want to research. For any target topic, there may be thousands of related articles on the Internet, but nobody can afford to take the time to read through all of them to determine which article gives the most relevant information. Search engines do help narrow the search, but it is ultimately up to the user to decide which article or page works best for them. If there was a system that could summarize an article in just a few concise sentences, then users would only have to read those to determine if an article is useful for them. Reducing the amount of necessary reading for users could help make research (both casual and professional) much quicker.

sumMary seeks to address this issue by parsing articles and summarizing them using the top few sentences from the article itself. That way, the user is receiving the text from the article that most accurately describes what it is about. sumMary is a processing model that uses standard preprocessing, evaluation, and ranking methods, rather than machine learning (ML) or natural language processing (NLP). As a result of this choice, sumMary will not generate its own sentences to summarize the articles or papers it analyzes; rather, it will use sentences that already exist in the document.

In this paper, we will introduce sumMary, explain the underlying algorithms that it uses, and describe evaluation methods used and results obtained.

1 Introduction

Text summarization involves taking a piece of text and shortening it to provide a summary of the text. This summary can be constructed one of two ways: either by having a program return sentences in the original article, or by having a program that constructs a original summary. These are the main methods used in field of text summarization, called extraction and abstraction respectively.

Extraction is the process of using the existing sentences in the text, to create a summary. Thereby returning the top N sentences that best represent the text. This method usually employs weighting schemes to decided the most popular or 'best' sentences. One of these schemes is standard term frequency measure, where unique terms are counted, and the weight of a sentence is the sum of weights for each unique term in the sentence. These algorithms seek to gain the meaning of the text, using the words in the text themselves. Certain issues arise with this method; most involve determining which features to weigh more heavily. For example, the choice to normalize the score of the sentence by the length of the sentence provides interesting debate in text summarization.

Abstraction is the process of using the existing text to generating new/original human readable sentences using Natural Language Processing techniques. These methods usually employ ML algorithms to help in the process of text generation.

Both these techniques can be used for many different summarization tasks. They are not limited to text summarization; they can be applied to both images and videos. In the case of sumMary, we use an extraction approach.

2 Related Work

There are a variety of approaches to text summarization. It has been a topic of increasing interest in recent years. However, a majority of text summarization has been focused on creating query-relevant summaries. One such ex-

ample is M. Sanderson's (Sanderson, 2000) proposed summarizer, which divides a document into equally overlapping passages. These passages are compared against an input query, and the passage that best matches the input will be returned as the summary of the document. Other flavors include summarizing generic text of multiple documents. One such example is D. Radev's Centroid-based summarization of multiple documents (Radev, 1999). His work describes several key parts, including Cluster-based sentence utility (CBSU), which refers to the degree of relevance of a sentence in regards to the entire collection of documents. There is also Cross-sentence information subsumption (CSIS) which refers to the fact that some sentences repeat information that is found in others. The core component is the centroid itself, which describes the tokens that are most indicative of the topic of a corpus of documents. These components are all used to calculate a score for each of the sentences extracted from the documents.

There is also work done on single document summarization. This includes manual summaries with local and global vocabularies. These are particularly difficult and usually involve more advanced machine learning methods such as deep learning. There are also several methods for single document generic text summarization that do not involve manual summaries. Similar to sumMary, these return parts of the document that are most relevant. One such case includes Y. Gong's work on summarization using relevance measure (Gong, 2001). The core algorithm starts with computing weighted term-frequency vectors for the document itself, as well as for all the sentences in the documents. The dot product between the sentence vectors and the document vector are computed and used a score for each of the sentences. Following this, the top sentence is selected and added to the summary. It is then removed from the document, along with all the tokens inside the document. The vectors are recalculated and the process is repeated until the number of sentences retrieved is satisfactory. This work is particularly interesting as it manipulates the document as sentences are extracted and added to the summary.

Text preprocessing is directly related to Generic Text Summarization. While tokenization, stopword removal, and stemming are fairly standard, sentence disambiguation can be achieved through a multitude of methods. Some simpler methods include rule based systems, while others are more complex with machine learning an deep learning. One such example is Conditional Random Fields specified by Tomanek et al (Tomanek, 2010).

The related work was pivotal in determining what methods to attempt in solving this problem, and provided insight into what pros and cons were involved when developing certain features of our system. One observation was the general diminishing returns that correlated with the complexity of a feature. One example is that machine learning techniques for splitting sentences were only marginally better than standard rule systems. This observation assisted in determining which feature of sumMary required the most work.

3 Approach and Methodology

With a body of text (e.g. an article, speech, or other medium sized text) as input, we sought to write our program to return N sentences from the text. These sentences would be those that our algorithm chose as the N most relevant to the overall message or subject of the article that is, the sentences that most effectively summarize the article. Rather than generating a natural language summary of the text, which would undoubtedly require our team to create extensive ML systems, we elected to design our system to summarize using already existing text. Our algorithms analyze the existing text of the document and compare its sentence content to the information gathered from the document as a whole. Our overall process of summarization follows these steps:

1. Split the document into sentences
2. Preprocess the text of each sentence (tokenize, remove stopwords, stem words)
3. Score each sentence based on the selected algorithm
4. Rank each sentence based on its assigned score
5. Return the top N sentences for the given document

3.1 Sentence Splitting

There are two levels of granularity with which we chose to analyze the text. The first and larger of these is the sentence level. In text summarization,

sentences within the text should be treated as separate objects that can be compared against one another. Due to the irregularity of punctuation use in the English language, not every instance of terminating punctuation (periods, exlamation marks, and question marks) should always denote the end of a sentence. This is due to the fact that not all of these mark the conclusion of a thought. Exclamation marks often follow short interjections that are often themselves followed by a sentence terminated with a period. Question marks are found at the end of questions, which are, in a text situation, usually followed by an answer for that same question. Furthermore, periods are used in other contexts, such as titles, abbreviations, acronyms, and initialisms. Based on these observations, we only delimited sentences with periods, and we chose not to split a new sentence if the period in question was part of a string that we determined to be non-terminating.

3.2 Text Preprocessing

The second, smaller level of granularity with which we analyzed text was tokens. Tokenization of text involves splitting text into individual alphanumeric representations. Common techniques involve splitting contractions into their source words (e.g. "you're" is tokenized to ["you", "are"]), removing commas (valid numbers are kept together but commas are removed), splitting hyphenated words, removing periods from the ends of sentences and from other strings like those listed in **3.1 Sentence Splitting**, and removing any other non-alphanumeric characters that might exist in possible tokens. There are many edge cases for tokenization, making it a challenging process. After sentences are split, each sentence is divided into a list of tokens that make up the content of the sentence. (Tomanek, 2010)

Not all tokens provide value to a piece of text. There are some that are so common that removing them entirely makes it easier to identify valuable information. These are called stopwords, and a crucial part of text preprocessing involves including a file with a list of stopwords and removing all stopwords from a n article's list of tokens. Doing this ensures that only valuable words will be considered in analysis. (Nedunchelian, 2008)

The final step in text preprocessing is to stem each token. Stemming is the process by which words are reduced to their root form, so that different tenses and conjugations will all appear as the same word. For example, "run" is the base word for "runs", "running", and "ran", so stemming will reduce all three of these strings to "run". This can potentially merge two words together that have different meanings. However we elected to use stemming anyway, as it has been shown to do more good than harm in IR systems. The algorithm we use for stemming in our system is the Porter Stemmer. Todo: Citation 3

3.3 Standard Term Frequency Scoring

After text has been preprocessed, the text will be in the form of sentences, which are each represented by a list of tokens. The first algorithm we use for sentence ranking uses the term frequency of words within a sentence to score a sentence. Initially, the algorithm counts the number of occurrences of each word (stopwords removed, stemmed) in the entire document. The score for each sentence is then the sum of the document-wide term frequencies for all words in the sentence divided by the length of the sentence. This algorithm is based on the idea that words appearing more often in a piece of text will be more important to the message of the text as a whole. Sentences that contain more of these words will, therefore, better summarize the text..

$$score(sent) = \frac{\sum_{i=1}^{w} tf_i}{|sent|}$$

We considered this algorithm to be a very simple, basic method for summarizing text. We were not able to find any previous work that used this method, as we imagined it would be too simple.

3.4 Vector Space Model and Centroid Summarization

The next method we used for summarization involved viewing each sentence as a vector of length V, where V is the vocabulary of the text. The weight of each term in a sentence vector is the term frequency of that term within the sentence. The algorithm would create an ideal, or centroid vector, which would represent the term vector for an ideal sentence. For a given document, this vector was constructed by taking the 14 most frequent words and giving each a weight of 1 in the centroid vector. Then each sentence was scored by its cosine similarity to the centroid vector. Cosine similarity measures the cosine of the angle between two vectors, to show how similar those vectors are in

V-dimensional space. The idea behind this algorithm was that the centroid vector would represent a sentence that contained the main "message" or "meaning" of a text document, and the sentences closest to that "meaning" would summarize the document most effectively.

$$sim(\textbf{sent}_j, \textbf{sent}_k) = \frac{\textbf{sent}_j \cdot \textbf{sent}_k}{|\textbf{sent}_j||\textbf{sent}_k}$$

$$= \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$$

$$score(sent) = sim(\textbf{sent}, \textbf{centroid})$$

(Radev, 1999) Creating an average or centroid vector to compare sentences to is a known technique that we found in multiple related works. Most, however, generated a vector for the document using all the terms and their corresponding weights, rather than a vector with the top few terms and fixed weights.

3.5 Naïve Bayes Based Probabilistic Summarization

Our final method of sentence scoring for text summarization was a probabilistic method based loosely on the Naïve Bayes text classification algorithm. Given training documents, test documents, and a set of classes, Naïve Bayes will use the training documents with predetermined classes to generate conditional probabilities for each word given each class. The test documents will be classified based on the class that their content has the greatest probability of appearing in. Not every part of this algorithm translates easily to text classification, as there is no true training data and there are not multiple classes in which to place the text. The algorithm for text summarization treats the provided article as its own class, and first generates probabilities of each word in its vocabulary appearing in the class. The probability for word x_k appearing in class c is

$$P(x_k|c) = \frac{n_k}{n}$$

where n_k is the number of appearances of x_k in the document, and n is the length of the document. The most common conditional probability formula for Naïve Bayes uses add-one smoothing to avoid zero counts, but for summarization, each word in every sentence is guaranteed to appear in the document, so we did not elect to use any smoothing. The score of each sentence is then the probability of that sentence appearing in the class.

$$score(sent) = \sum_{k=1}^{w} P(x_k|c)$$

Note that we do not divide the score by the length of the sentence. Doing so causes this algorithm to heavily favor very short sentences, which often do not give enough detail to effectively summarize text. We have not found related work that uses probability to summarize text, and we thought using Naïve Bayes would be a unique approach. However, given that there is only one class, the algorithm becomes much closer to one based on term frequency.

4 Data Sets

Evaluating our summarizer was one of the main challenges of the project. In general, evaluation requires the use of a 'golden standard'. This golden standard is usually created from human judgment and evaluation. In the case of a text summarizer, a human could read an article and then either manually write a summary of the text, or pick the top N sentences of the text to be used as a summary. For the evaluation of sumMary, the method we chose to employ was selecting the top N sentences. We describe our reasoning later in this section.

4.1 Available Data Sets

Text summarization is widely researched field, with many people working on outside-the-norm solutions to summarizing documents. With this much unique work, there needs to be a way of analyzing and comparing methods using a standard dataset. The **Document Understanding Conference** (DUC for short) is a conference that is focused on just that: analysis of text summarization techniques. Along with the conference is a standard dataset called TREC. This data set is extremely large and contains many text articles. These text articles could be individual articles, or a group of articles that consist of the same theme. The articles are then summarized by humans. The way in which they are summarized is as follows: a human reads the article or articles and writes a summary of them. This summary is one or two sentences, again it is manual. Therefore might, or might not include original text from the article/s.

Originally we tried to evaluate our program using the TREC dataset. Unfortunately, due to the structure of the TREC datasets and our program we ran into some issues. First, as explained above, the TREC dataset are short summaries of one to two sentences. While we could limit our program to only output this amount, we didn't think it would accurately capture the accuracy,precision and recall of our program. Second, and most importantly the TREC dataset were manual summaries, while our program output the top five sentences. Obviously these two methods were incompatible; resulting in us using and collecting our own datasets.

4.2 sumMary Data Sets

The datasets sumMary used to evaluate its algorithm were composed of 40 articles gather by the group members. These articles were then summarized by different humans. The process involved picking the top five sentences in the article that they thought best represented the text.

5 Evaluation of sumMary

We evaluated the effectiveness of the summarizer against human-made summaries using standard recall and precision. Due to the nature of the sentence splitting algorithm we used, and the challenges that sentence splitting posed, the evaluation process had some limitations, which may have produced artificially lowered results for macro and micro precision and recall.

5.1 Methods of Evaluating

We considered the sets of relevant sentences to be those in the manual summaries. The sets of retrieved sentences were the sentences found in the various generated summaries. There are three sets of evaluation data: one for the term frequency scoring, one for centroid scoring, and one for the Naïve Bayes-Based probabalistic scoring.

We considered several different perspectives using precision and recall. We found the micro average precision, the micro average recall, the macro average precision, and the macro average recall. Additionally, it was interesting to isolate the the summaries for which the precision and recall were nonzero to see how well sumMary performed when it was able to retrieve any of the relevant sentences. We wanted to visualize the distribution of precision and recall results because the data turned out to be very bottom-heavy.

$$Precision = \frac{|Manual\ Sentences \cap Generated\ Sentences|}{|Generated\ Sentences|}$$

$$Recall = \frac{|Manual\ Sentences \cap Generated\ Sentences|}{|Manual\ Sentences|}$$

5.2 Limitations and Challenges in Evaluation

The task of summarization lends itself to interesting and unique challenges in evaluation. Aside from the aforementioned difficulties in using natural summaries versus top-sentence summaries as a gold standard, the chosen sentence splitting algorithm can create the need for special considerations when comparing certain genres of text to the gold standard.

5.2.1 Sentence Splitting

A useful feature of the sentence splitting algorithm was that it was designed to try to leave quotes within the text unsplit as much as possible. Consider the following example:

She admitted, "I've never been very involved in the political scene." Her tone of voice left me questioning whether she even believed herself, "It's just one of those things you put on the back-burner when life gets busy."

While in certain contexts it may be useful to split such a syntactical pattern into

She admitted, "I've never been very involved in the political scene."

and

Her tone of voice left me questioning whether she even believed herself, "It's just one of those things you put on the back-burner when life gets busy."

we reasoned that the latter sentence would be out of context if along in a summary and less useful when not combined with its leading counterpart. There are, however, situations where the sequence

word1." Word2

is better split, namely when the Word2 begins a sentence that does not continue the previously given quote. We determined that it would be better to have one sentence of extra information resulting from the latter case than to wind up including

a piece of a quote without its context, so we left sequences like *word1." Word2* together as a single sentence.

During the evaluation we realized the major flaw this brought about: the top 5 sentence-rankings were not always interpreted as 5 sentences when quotes were involved. Imagine that the original article contains the following:

"This is the first string." This is the second string. This is the third. This is the fourth.

Our sentence splitter would parse this into:

sentence 1: *"This is the first string." This is the second string.*
sentence 2: *This is the third.*
sentence 3: *This is the fourth.*

Now suppose that the manual summarizer determines that a good manual summary is the following:

"This is the first string." This is the third. This is the fourth

Of course now the chosen sentences have different neighbors than they did in the original full text. The issue we encounter is that when we run the manual summary through the sentence splitter, the result is:
sentence 1: *"This is the first string." This is the third.*
sentence 2: *This is the fourth.*

The problem that arises is that **sentence 1** of the manual summary does not occur anywhere in the sentence representation of the article. This means that this "relevant" sentence has a 0% change of appearing in the generated summary, which only contains sentences from that sentence representation of the article. The end result is that even if sumMary successfully includes a particular quote, the evaluation methods used would not identify the set overlap between the retrieved sentences and the relevant sentences, thus artificially lowering the evaluation measurements.

A work around for this issue had we detected it sooner would have been to write a separate sentence parser for the manual summaries, such as a parser that split sentences at newlines, with all manual summaries being given as 5 sentences, separated by newlines. Another work around would be to modify the sentence splitter to look ahead into the sentence following a quote to see if the quote continues or not, and to split only if the second sentence does not contain a quote.

5.2.2 Data Availability

The data readily available was highly limited given our relatively short time frame for completing the sumMary project. We had considered the aforementioned TREC dataset from the Document Understanding Conference, however this was not practical for our evaluation needs due to the reasons described in **4.1 Available Data Sets**.

5.3 Evaluation Results

The centroid scoring method of summarization outperformed both of the other methods. Initially, due to the use of some unhandled unicode characters in the manual summaries as well as the anomaly in consideration of quotes in sentence splitting, which is described in section **5.2.1 Sentence Splitting**, several of the 40 manual summaries were considered by our sentence splitter to contain fewer than the expected 5 sentences. This lead to skewed recall and precision, as pictured below:

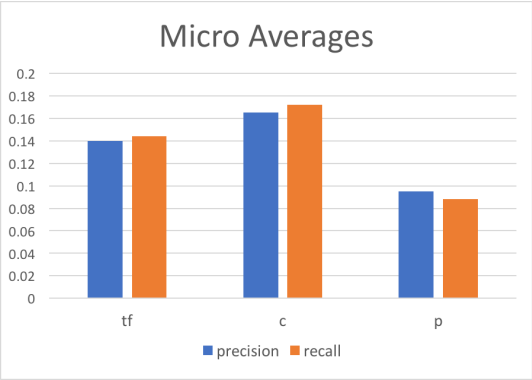

Figure 1: Skewed precision and recall micro averages with a summary length of 5 sentences. This was supposed to be the R-precision, but clearly the recall and precision are not the same due to the unanticipated issues with quotation parsing
.

We realized that the data was incorrect because we wanted to calculate R-precision and recall, yet the values given in the chart above do not match. After replacing slanted quotes with regular quotes as well as adding relevant sentences to manual summaries that were insufficiently long, we found the R-Precision for R=5 pictured below. The probabilistic scoring is also higher because we decided

to modify our algorithm slightly. In the updated results, normalization by sentence length was removed so that shorter sentences receive less priority. We also removed the add1-smoothing because every token will appear in the article at least once.

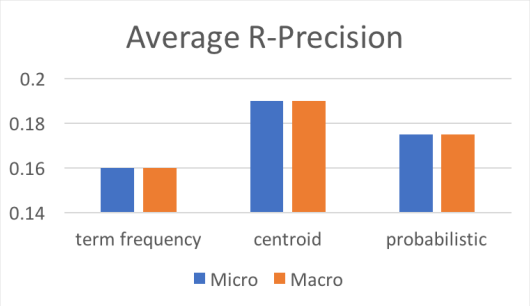

Figure 2: Micro and Macro average R-Precision for each scoring method

As mentioned, the data was generally very bottom heavy, with a significant number of 0 precisions.

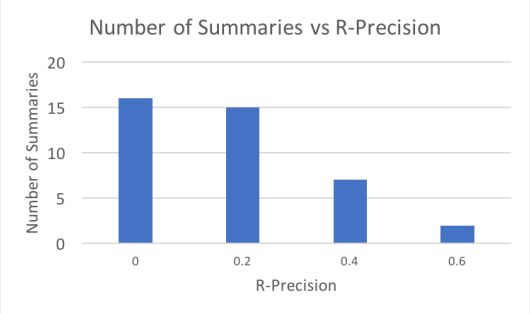

Figure 3: The graph above shows how many summaries were evaluated at each respective R-precision

For most of the articles that it did retrieve relevant sentences for, the most common precision was 1/5 sentences. In the future, if we had time to obtain longer manual summaries, the results might be better. For longer articles, it became less and less likely that the generated summary would select the exact 5 relevant sentences chosen for the manual summary, simply because there was more for it to choose from. So, it performed best on medium to small length articles.

6 Further Considerations and Improvements

Further considerations can be made across all aspects of this project, starting with preprocessing (specifically sentence splitting). There are many

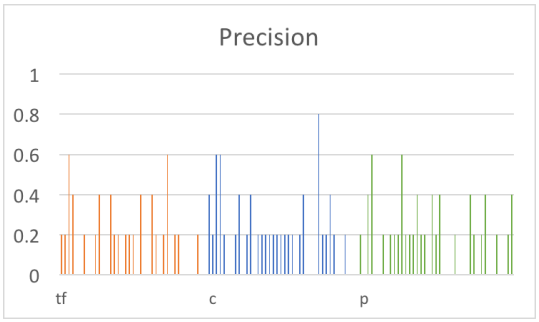

Figure 4: Each bar represents the R-Precision of an individual article, with each scoring method shown in a different color. The density at 0.2 demonstrates that for the overall majority of articles, one out of the 5 relevant sentences was retrieved.

ways to consider the end of a sentence. As previously mentioned, only the period (".") was considered in our case, but we did allow for expansion with other punctuation. Our rule based system could be replaced by a more effective machine learning mechanism. This would improve sentence splitting only marginally, but could help significantly for specific types of articles. sumMary targeted standard medium length news articles, which did not usually contain ambiguous tokens such as initials and taxonomic names. Other document types such as research papers and scientific articles could contain difficult sentences to split. Perhaps an ML approach to sentence splitting would allow sumMary to broaden the types of articles it can effectively summarize.

The majority of the further considerations and improvements can be made in regards to the algorithms chosen to score sentences. There are many questions that can be asked for each of the algorithms, and whether or not they would improve performance.

For standard term frequency scoring, we considered normalizing the term frequencies with the maximum term frequency. Another significant consideration was computing an "inverse sentence frequency" (isf) for each term. This is similar to inverse document frequency, but instead of measuring the occurrence of words in documents, it measures the occurence of words in sentences. In the end, we determined that although idf is important when dealing with a corpus, isf would not accurately measure the importance of a word in a sentence. This is because words that are important

in an article are likely to appear in multiple sentences in an article, rather than just a few times.

The centroid algorithm also provides a variety of tweaks that could be implemented and tested. The optimal sentence vector could be determined in many different ways. As previously stated, the optimal vector in sumMary is chosen using the N (14) top most popular words. However, we could have used the entire document as a vector, similar to Y. Gong's work. If this isn't desired, there is also the option of choosing a scaling N amount of words as the length of the optimal vector. This value would change depending on the lengths of the sentences in the document.

There are also considerations that could be applied regardless of the algorithms. One such example is dealing with small sentences that have high scores. These small sentences are not very good for summarization, as they often need the context from the sentences surrounding them to make sense. As such, should they be removed? Or perhaps they could be combined with the sentences surrounding them.

Other improvements could be made to sumMary to further the practicality of the system. One useful feature would be the ability to provide a link to an article that a user would like to summarize. This would be an improvement over copy and paste methods that could include strange symbols or headlines that parsers have difficulty with. With a link, an HTML parser can properly extract the text that sumMary would summarize.

Better data sets would definitely improve our ability to analyze sumMary's performance. The data set that we were able to use to test our system and evaluate our output only consisted of 40 documents, each with a 5 sentence "golden standard" summary. The data set itself was rather small; a larger, more diverse data set would have allowed us to do more testing to fine-tune our algorithms. The static number of sentences in the "golden standard" summary causes issues scaling with respect to document length. Very short documents will generally have better values for precision and recall, as the 5 sentences used for the summary make up a large percentage of the total number of sentences in the document. Inversely, very large documents with many sentences will have worse precision and recall values, as 5 sentences will make up a very small percentage of the document's total. An ideal data set would have its ideal summaries proportional in length to the articles they summarize.

7 Individual Contributions

Initially, we discussed the general algorithm then all wrote an implementation of the summarizer with term frequency scoring. Each team member had their own preprocessing from EECS 486, so the significant challenge was to come up with a creative algorithm for sentence splitting.

After writing our own implementations, we discussed each person's algorithm, and selected the one that seemed to have the best results. We tested the individual summarizers on the Wikipedia article for "Mr.Bean" as well as on news articles to compare.

We chose Dillon Kesto's summarizer as the base for sumMary with term frequency scoring and discussed improvements that could be made based on the unique features each of our summarizers offered. Dillon Kesto's summarizer was particularly unique because of the quote handling that he incorporated into his design. We later decided to use Trevor Rees's preprocessing functions due to how thorough they were.

Primarily: Arber Xhindoli made the web application used to interface with the summarizer along with a script to run all our algorithms on the training data; Dillon Kesto wrote the sentence splitter, term frequency scoring, and centroid scoring; Caroline Saab wrote the functions for evaluation; and Trevor Rees implemented the probabilistic scoring method as well as preprocessing.

Everyone chose 10 articles each from a popular news outlet and sought out volunteers among family and friends to create the top-sentences summaries.

Everyone contributed to the report and the poster, focusing more so on the same areas that they narrowed in on during the code design and implementation.

Overall, contributions by all members are mutually agreed to be even throughout all aspects of the project. We are all satisfied with one another's work and feel that work was appropriately and fairly distributed across team members.

Acknowledgments

We would like to thank the University of Michigan Electrical Engineering and Computer Science department, as well as the EECS 486 staff for making

our work on this project possible.

References

Yihong Gong. 2001. *Generic Text Summarization Using Relevance Measure and Latent Semantic Analysis*, 1(1):1–25.

Canasai Kruengkrai. 2003. *International Conference on Web Intelligence*, 24(1):1–6.

Prof. R. Nedunchelian. 2008. *Centroid Based Summarization of Multiple Documents Implemented using Timestamps*, 1(1):480–485.

Dragomir R. Radev. 1999. *Centroid-based summarization of multiple documents: sentence extraction, utility-based evaluation, and user studies*, 1(1):1–8.

Mark Sanderson. 2000. *Accurate user directed summarization from existing tools*, 1(1):1–5.

Katrin Tomanek. 2010. *Sentence and Token Splitting Based On Conditional Random Fields*, 1(1):1–8.