Multi-document Text Summarization

Tracy Rohlin
University of Washington
tmrohlin@gmail.com

Karen Kincy
University of Washington
kincyk@uw.edu

Travis Nguyen
University of Washington
travin@uw.edu

Abstract

In this paper, we provide a description of our multi-document summarization system. The documents are parsed using several modules, the sentences of the documents are clustered and scored, and the scores of the sentences are used in conjunction with a pre-set word count threshold to produce summaries that are then analyzed by the ROUGE evaluation toolkit.

1 Introduction

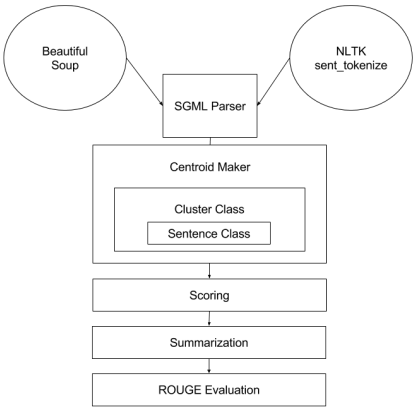
Multi-document summarization has become a burgeoning field in natural language processing where multiple documents are collected and parsed in order to provide a concise and readable summary. There have been many approaches constructed to tackle the problem, including graph models, rule-based methods using hidden Markov models, and statistical methods. One of the seminal approaches in multi-document summarization devised by researchers has been to use a centroid approach. This process, where the sentences closest to the centroids in topic clusters are extracted into the summary, was first described by Radev and other researchers in Radev, Jing, and Budzikowska (2000) and Radev, Jing, Stys, and Tam (2004). We applied this simple and effective approach to the AQUAINT data set provided by the National Institute of Standards and Technology (NIST). Afterwards the sentences are ordered chronologically according to an algorithm described by Bollelaga, Okazaki, and Ishizuka (2011). What follows is a description of this approach. Section 2 describes the general architecture of our system with a more detailed description in Section 3. The results of our system are described in Section 4 with a follow up discussion in Section 5.

2 System Overview

The system is comprised of three major parts: a Standard General Markup Language (SGML) parser that links the document IDs found in the guided summary file to the document SGML files in the corpus folders, a centroid-based algorithm that scores the sentences within each cluster, and a knapsack algorithm that chooses the highest scoring set of sentences constrained by the word count limit of the summary.

2.1 System Architecture

The following graph displays the architecture of our system:



3 Approach

3.1 Content Selection

The SGML parser uses Python's subprocess library to grep for each document ID found in the guided summary file. Once found, the exact file is read and parsed using BeautifulSoup and the Natural Language Toolkit (NLTK). This was done to avoid loading all 3,000 files in the corpus folder into memory before linking the document IDs from the summary files to the document files, thereby reducing the risk of running out of memory. The text of each document is retrieved

using BeautifulSoup and pre-processed by removing extraneous tags, duplicate headlines, and the document IDs themselves from the text. The text is then tokenized into sentences using NLTK and saved into a JSON object, where the keys are the topic IDs and the values consist of the tokenized documents along with the document headlines. This file is then loaded into the next program in the pipeline, where it is to be processed even further.

Next, the centroid-based algorithm loads the JSON file into memory and further processes the text with a series of regular expressions. The module parses the processed text to initialize a list of Cluster instances. Unlike the MEAD algorithm outlined by Radev et al. (2004), our system does not automatically perform any clustering; since each cluster produces its own summary, we defined a cluster as a set of documents with the same topic ID. This allows our system to produce one summary per topic, avoiding multiple summaries for one topic ID.

Each Cluster object contains a list of instances of the Sentence class, which encapsulates all the data necessary to rank a sentence. The algorithm assigns four scores to each sentence: (1) centroid score, (2) position score, (3) first sentence overlap score, and (4) topic focus score. In addition, a redundancy penalty is subtracted from the sum of these scores. More detail about the redundancy penalty follows after a discussion of the scoring mechanism.

First, the centroid-based algorithm calculates TF*IDF for each of the terms in this cluster, using the "news" subset of the Brown corpus for inverse document frequency. To optimize the code, IDF scores for every term in the background corpus are calculated once and stored in a dictionary. Term frequency is defined as the average frequency of a term across the documents in the cluster. Rather than using a TF*IDF threshold like the MEAD system, the centroid size is selected by a system argument. A centroid size of 50, for example, saves the top 50 terms for the cluster, ranked by descending TF*IDF scores.

When building a centroid, the system preprocesses the text by tokenizing each sentence within the cluster, lowercasing every token, and removing punctuation-only tokens. A list of cleaned tokens is stored within every Sentence instance, allowing the centroid score for the aforementioned sentence to be easily calculated. For each token in the sentence, if the token appears in the centroid for this cluster, the token's TF*IDF score is added to the total centroid score for the sentence. This process ignores terms not appearing in the centroid, giving better scores to sentences with relevant terms.

Second, the system calculates the position score for every Sentence instance. The formula from Radev et. al. (2003) prefers first sentences, since the position score of the first sentence in every document equals the highest centroid score for that cluster. Every sentence after the first sentence gets penalized with the following formula, where *n* is the position of the sentence within the document and $c_(max)$ is the value of the highest centroid score for the cluster:

$$P_i = ((n - i + 1)/n) * C_(max)$$

Third, the system initializes the first sentence overlap score for each sentence, defined as the dot product between vectorized sentences. As a form of optimization, rather than vectorizing the text, every Sentence instance contains a dictionary of (word, count) pairs; the overlap between sentences can be quickly calculated by finding the intersection of keys between dictionaries. For every word shared by both sentences, the counts are multiplied and added to the first sentence overlap score.

Our improved system incorporates a fourth score for topic focus. This score is calculated by creating a continuous BOW model based on the background corpus (i.e., Reuters or Brown). Then, for every token in the document, that token is compared to each word in the topic phrase (e.g., "Giant Panda"). This comparison produces a similarity score by looking at the surrounding context of both words to determine if the words occur in in similar contexts. By hand-tuning the threshold, we concluded it should be set to 0.75, meaning *sim(token, topic_word)* $>= 0.75$ produced good results. If two words produce such a similarity score, an extra point is added to the sentence's topic relatedness score. After processing, this summed score is multiplied by the topic weight parameter to produce a weighted topic relatedness score.

Like the topic score, all the scores in our system have weights. Radev et al. discuss weights for MEAD, though they set the weights equal to 1 and leave tuning for future work. We performed parameter optimization for the weights in our system, using a grid search of ROUGE recall scores,

to determine the best values. In addition, we added the choice of background corpora (for calculating TF*IDF) as a parameter, and found that Reuters produced the best results. This makes sense, since Reuters belongs to the news domain, and is therefore relevant to the input corpora.

Currently, the following parameters produce the highest average ROUGE recall scores:

Size	TopN	Corpus	CentroidWeight
100	10	reuters	5
PosWeight	FirstWeight	Redundancy	TopicWeight
1	1	100	1

Finally, a redundancy penalty is subtracted from the sum of the previous three scores. The original MEAD algorithm calculates redundancy pairwise, comparing the similarity between each sentence and the immediately preceding sentence, ranked by total score in descending order. We found this approach still had too much redundancy within the summaries, and opted for a revised approach. The redundancy penalty for our system is cumulative; that is, the overlap for each sentence is calculated using the intersection of the word types of the current sentence to the word types of all sentences with a higher total score.

$$R_s = 2 \times \frac{\#overlappingwords}{\#wordsinpreviussentence + \#wordsincurrentsentence}$$

After the total score of each sentence is calculated, the sentences are ordered by their total score, in descending order, and only the top N are considered to create the summary. In our experiment, we used N = 20.

In order to create the summary, we use the knapsack algorithm, which allows us to select a subset of the top N sentences such that the word count of the summary, equal to the sum of the word counts of the sentences in the subset, is less than or equal to a given threshold. The knapsack algorithm also selects a subset of the top N sentences that has the highest total score, calculated as the summation of the total scores of each sentence in the subset. Per the assignment, the threshold for the summary word count limit is set as 100.

3.2 Information Ordering

Our information ordering algorithm is an adaptation of Bollelaga, et al.'s chronological ordering algorithm. The original algorithm sorted sentences first on publication date of the sentence, then on sentence position. Ordering based on sentence position helps to break ties in the case where two sentences have been extracted from the same document and thus have the same date-time associated with it, or two sentences come from separate documents yet still have the same date-time.

The modified version of the algorithm takes the chronological ordering but first tries to order based on whether the sentence is a first sentence or not. If the sentences being compared are not first sentences, it then will compare based on date-time and sentence position. This was found to have improved readability among all members of our group compared to the original algorithm. This may be due to the fact that newspapers often place the most pertinent and most general information towards the top of the article, with detailed information following after the lede. This also prevents summaries where one sentence with an earlier date-time that contains more detailed information is placed before an older but more general lede sentence.

3.3 Content Realization

We have not implemented content realization. For the final deliverable, we would like to try sentence compression using the algorithm proposed by Zajic, Dorr, Lin, and Schultz (2007) and entity linking to improve coherence and cohesion.

4 Results

Reviewing the ROUGE scores, the ROUGE-1 scores are consistently the best for summaries produced by our system. This makes sense, since the TF*IDF metric favors relevant unigrams.

4.1 Improved results

The following table displays the ROUGE-1, ROUGE-2, ROUGE-3, and ROUGE-4 recall values, in terms of percentage, compared to our original ROUGE scores. The scores increased across the board, but less so for ROUGE-3 and ROUGE-4. The ROUGE-1 and ROUGE-2 scores increased 1-2%.

Scores	ROUGE-1	ROUGE-2	ROUGE-3	ROUGE-4
Old	23.654	6.117	1.829	0.618
Improved	25.363	7.330	2.577	1.001
Difference	1.709	1.213	0.748	0.383

5 Discussion

Overall, we found the our parameter optimization and topic focus did improve our ROUGE scores, though perhaps not as much as we had expected.

In the last assignment, D2, our system contained a bug that caused the scores from the ROUGE evaluation toolkit to be partly unreliable. We erroneously generated unique 10-character alphanumeric codes for each summary when we should have generated one alphanumeric code per test run. Our error was causing the system to recognize each summary as a test set, leading us to have 46 test runs in the resulting scoring file produced by ROUGE. Upon fixing the code, ROUGE works as it should. We also created several scripts to automate the process of running ROUGE and retrieving the scores.

As a footnote, we attempted to incorporate lemmatization into our system, replacing the terms in each sentence with their lemmas in order to collapse lexically-similar terms, but we found the WordNetLemmatizer from NLTK introduced more mistakes than improvements. For example, the token "was" became "wa", and "species" became "specie". We tried POS tagging the tokens and passing either "N" or "V" (to disambiguate between nouns and verbs; e.g., "dog") as an optional parameter to the lemmatizer, but it did not improve our system overall. At this time, we have backed off to terms instead of lemmas, though we may explore a more sophisticated approach for the next deliverable.

6 Conclusion

In conclusion, our multi-document summarization system pre-processes documents, scores the sentences within each topic cluster, and uses the knapsack algorithm to create a summary, keeping in mind the highest scoring subset of sentences as well as the summary word count threshold. These sentences are then ordered by using a modified version of Bollelaga, et al.'s algorithm, where first sentences are placed at the top of the summary and the other sentences are ordered chronologically and then by sentence position. The final resultant summaries are then evaluated using the ROUGE evaluation toolkit. The results show that ROUGE-1 and ROUGE-2 consistently perform the best, although there is much improvement to be made, while ROUGE-3 and ROUGE-4 often produce scores of zero even though the bug in our ROUGE score was fixed. As stated in our last report, this is explained by the fact that types of n-grams become less frequent as the n-gram size increases. This is still an area that requires improvement in our next deliverable.

In the future, we would like to add a machine learning module, and perhaps more semantic similarity measures to capture finer-grained relationships between sentences, to futher improve content selection. The interplay between redundancy and relevancy remains a balancing act, and we hope to make greater improvements.

References

Dragomir R. Radev, Hongyan Jing, & Malgorzata Budzikowska. 2000. *Centroid-based Summarization of Multiple Documents: Sentence Extraction, Utility-based Evaluation, and User Studies*. NAACL-ANLP-AutoSum '00 Proceedings of the 2000 NAACL-ANLP Workshop on Automatic Summarization. Pages 21-30.

Dragomir R. Radev, Hongyan Jing, Malgorzata Stys, & Daniel Tam. 2004. *Centroid-based Summarization of Multiple Documents*. Information Processing and Management 40. Pages 919-938.

Kai Hong, Mitchell Marcus, & Ani Nenkova. 2015. *System Combination for Multi-document Summarization*. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal. Pages 107-117.

Natalia Vanetik, & Marina Litvak. 2015. *Multilingual Summarization with Polytope Model*. Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue. Pages 227-231.

David Zajic, Bonnie Dorr, Jimmy Lin, & Richard Schwartz. 2007. *Multi-candidate Reduction: Sentence Compression as a Tool for Document Summarization Tasks*. Information Processing & Management. Pages 1549-1570.

Danushka Bollegala, Naoaki Okazaki, & Mitsuru Ishizuka. 2012. *A Preference Learning Approach to Sentence Ordering for Multi-Document Summarization*. Journal of Information Sciences. Pages 78-95.