J2MSummarizer

Mami Hackl **John Keesling** **Cuijun Wu**

University of Washington
P.O. Box 354340
Seattle, WA 98195
{mami1203, keesling, cuijunwu}@uw.edu

Abstract

In this paper, we describe the J2MSummarizer, a query-biased, multi-document, extractive summarization system built for Linguistics 573. We show that various shallow processing techniques are a simple yet effective way to process queries. We also integrate Lucene, an information retrieval system, into our summarizer. Finally, we evaluate our results using the ROUGE metric and compare our system with the top-scoring systems from several past years of the Document Understanding Conference (DUC).

1 Introduction

The highest-performing systems from DUC are diverse in methodology. Their approaches span the full spectrum of NLP. Some solely make use of shallow processing techniques while others almost exclusively use deep processing techniques. However, there are quite a few things that a majority (at least of the ones we read) have in common. Most systems use some kind of shallow processing to refine and/or expand upon the query in some way and most are extractive and query-biased. This observation led us to select several of these points for our system, in hopes of replicating some of the reported successes.

Following Conroy et al. (2005, 2007), we developed our query-processing module around the notion that generating part of speech tags and retaining only the most meaningful words would yield similar results to more complex methods.

We chose to use Lucene for several reasons. The most significant reason is merely that Lucene is an effective tool that can be easily obtained and implemented off-the-shelf. That

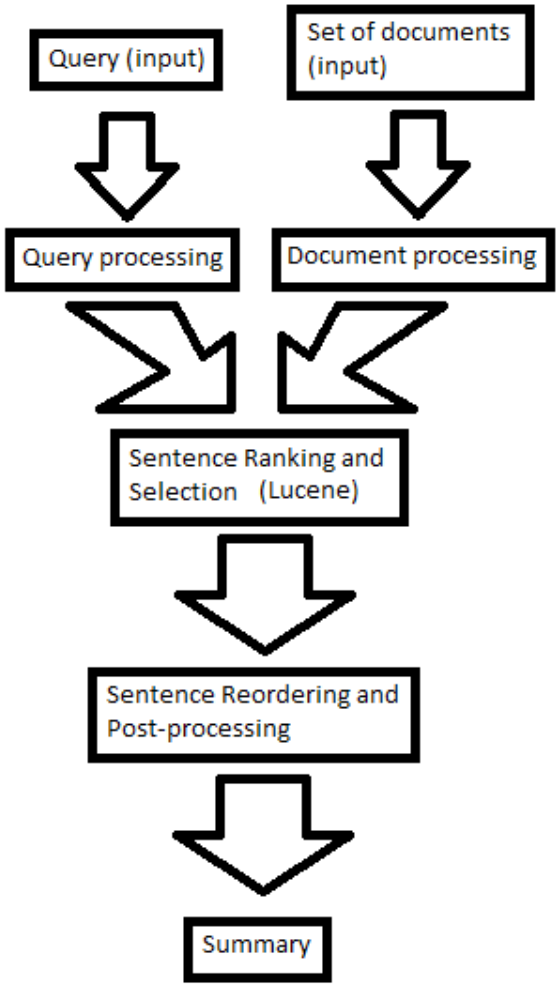

Figure 1. System Overview

was especially desirable for this type of task as we hoped to make this system easily and quickly replicable. However, there are additional benefits to using Lucene. It is based around an implementation of the *Tf-idf* function that is particularly fast. This is beneficial as the *Tf-idf* is well-established as a common aspect of NLP.

After a set of ranked sentences is returned, our system calls LingPipe (Alias-i, 2008) to reorder the sentences into a more coherent summary. LingPipe uses disjoint clustering to accomplish this task.

In addition to using LingPipe for reordering our summary, we also use LingPipe to resolve some anaphora ambiguity. Although this aspect of the system suffers from the flaws that afflict all anaphora resolvers, it does seem to improve readability quite successfully.

Finally, the J2MSummarizer performs a great deal of additional processing in order to improve readability. As the development process occurred, we found that focusing on human readability instead of ROUGE performance was of greater importance to us. So, these processing steps are designed to mend as many readability issues as possible, sometimes even at the expense of a few fractions of a ROUGE point.

The J2MSummarizer performed well in terms of the ROUGE metric. Although development was centered around improving human readability of the summaries, our final qualitative evaluation scores were poor compared to the rest of Linguistics 573. This suggests that perhaps more time could have been devoted to grammaticality, referential clarity, and structural coherence.

2 System Overview

Our system, shown in Figure 1, draws inspiration from numerous sources. The system begins with query and document processing. For query processing, we begin by generating a POS tagset and keeping all nouns from the result.

After this initial processing step, each sentence from each document is ranked using the Lucene information retrieval system. This ranking is then used as part of the formula for ranking and selecting sentences for extraction. After selecting the highest ranking sentences, the system reorders and organizes the extracted sentences as much as possible in order to create the final output. Additional details appear in the subsequent sections: In Section 2.1, there is some discussion of the resources used; in Section 3, we elaborate on our methods for query processing; in Section 4, we present an overview of how our system selects content at this time; in Section 5, we discuss ways in which we order the selected content; in Section 6, we discuss various post-processing steps, in Section 7, we evaluate our results; finally, in Section 8, we discuss potential future directions for our system.

2.1 Resources Used

This system makes use of multiple off-the-shelf resources. We select sentences from these documents by ranking them using the information retrieval system Lucene.[1] We sort these extracted sentences into a more coherent summary and resolve pronoun reference ambiguity using the suite of Java libraries LingPipe.[2] Finally, we remove overly-similar sentences and post-process. In this final stage, readability and brevity become a focus, as the system relies on regular expressions and chunking to strip irrelevant, redundant elements, and stray punctuation and phrases from a sentence's previous formatting.

We experimented with two other resources that did not prove beneficial in our system. We tried processing all input using an implementation of the Porter stemmer that was developed by its author (Porter, 1980).[3] We also tried expanding the set of query terms to include the top synonyms for each term using WordNet (Fellbaum, 1998).[4] These two additions did not improve our ROUGE score, so we chose to omit them from the system entirely. It should be mentioned that the inclusion of LingPipe for coreference resolution remains to prove its value to the overall system. Although it appears to improve human readability, our claim remains naïve without some kind of empirical evaluation.

3 Query Processing

We have integrated a fairly simple method of query processing into our system. First, queries are stripped of .xml tags using our own script. This results in a list of words that is then processed further.

For this new list, POS tags are then generated for each word in the query using the NLTK POS tagger (Loper and Bird, 2005). From this set of tagged words, we eliminate all words that are not 'NN,' 'NNS,' or 'NNP.' This eliminates a great deal of the unnecessary words, including stop words and words that are less important to the semantics of the query.

As mentioned above, we first intended to apply the Porter stemmer to the list at this step to remove as much morphology as possible (Porter, 1980). This final step was originally deemed useful following numerous others that found significant improvements when stemming is used (Conroy et al, 2007; Hennig, 2009), but we found no improvement for our system.

[1] http://lucene.apache.org/

[2] http://alias-i.com/lingpipe/

[3] http://tartarus.org/~martin/PorterStemmer/

[4] http://wordnet.princeton.edu/

$$score(q,d) = coord.factor(q,d) \cdot query.boost(q) \cdot \frac{V(q) \cdot V(d)}{|V(q)|} \cdot doc.len.norm(d) \cdot doc.boost(d)$$

Figure 2. Lucene's Scoring function

4 Content Selection

The process of selecting sentences for extraction is divided into several steps. Our strategy for this task has evolved since the beginning stages of development. At first, the J2MSummarizer ranked each document by relevance to a query using Lucene. However, we ultimately found that ranking each sentence directly was a better approach. Thus, we select content using a ranking of each individual sentence. This process is described in more detail below.

Lucene is an information retrieval system that searches for various key terms in a set of documents or sentences and returns those which score the highest. The score of a given document or sentence is calculated using several methods including term frequency in a document (*Tf*) and inverse document frequency (*idf*), among other things. Thus, Lucene plots *Tf-idf* weighted vectors in a multi-dimensional space and calculates a refined Cosine Similarity for those vectors (see Figure 2).

This Cosine Similarity function is the product of the following: The coordination factor is a score based on how many of the query terms are found in the text. The query boost is set at 1.0 by default, but allows the user to place greater importance on any given query term relative to the others. The third term is the dot product of the weighted vectors *V(q)* and *V(d)* divided by the Euclidean norm of the weighted vector *V(q)*. The *doc-len-norm(d)* allows Lucene to avoid the problems created by normalizing *V(d)*. This normalization removes all document length information, which might be problematic for texts that are not composed of many duplicate paragraphs. Finally, the document boost allows the user to increase a document's relevance to the search much in the same way that query boost does. It is also set to 1.0 by default.

As mentioned above, this particular step has evolved over the course of development. In our first approach, which ranked all documents, Lucene returned the vast majority of the available documents. Although they were ranked, we decided that it makes more sense for a variety of reasons to simply rank all sentences and return the top 20 of those. As a final step, we manually sort out any duplicate content that is returned at this stage.

Our reasons for eliminating the document ranking are roughly two-fold. First, and foremost, some problems could potentially arise when dealing with ranked documents. For example, a lower-ranking document's sentences may never be processed if the word count is already reached, even though that document might contain the single most informative sentence in the entire set. It is unclear how this would affect our ROUGE evaluation results; however, it naïvely does not seem to do any harm. The second problem with ranking the documents is one of performance. On Patas, the document-ranking system took upwards of 20 minutes, while the sentence-ranking system currently only takes around three minutes. This particular point would not be relevant for a competition such as DUC, but for the purposes of developing a commercially-viable system, this kind of speed increase is potentially important.

5 Content Ordering

Our system relies on Lucene to rank the sentences. They are ordered by relevance to the query, as is expected of the typical output of Lucene.

After Lucene generates this ranked output, the J2MSummarizer then calls LingPipe to help sort the extracted sentences into a more coherent order. LingPipe uses disjoint clustering to group similar items together. In this case, LingPipe groups the most similar sentences together, which in turn leads to a more coherent summary for J2MSummarizer.

6 Post-processing

The J2MSummarizer system improves readability and ROUGE scores using numerous post-processing methods. It should be noted that some of the methods described here do not necessarily happen after content ordering. However, for reasons of presentational simplicity, they are included in this section.

6.1 Redundancy removal

In addition to re-ordering a summary based on sentence similarity, the J2MSummarizer uses sentence similarity to remove overly similar sentences. This is performed by an in-house system that, in short, uses simple string comparison to assign a score between each pair of sentences. We determined the maximum allowable score between two sentences manually, by looking at the summaries returned by various thresholds.

6.2 Coreference resolution

The J2MSummarizer makes use of the LingPipe Java library to resolve ambiguous pronouns when applicable. For any appropriate sentence, the resolver analyses that sentence's source document, creating a whole-document reference chain. Then, the relevant pronoun is replaced by the first element in the reference chain.

At the time of final submission, this aspect of our system resolves pronouns in the nominative, accusative, and genitive case. This has two primary effects. First, it improves readability due to the reduced ambiguity. Second, it sometimes hinders readability due to the still somewhat flawed implementation. In rare cases, it returns an incorrect form of a genitive pronoun – specifically, it will attach the possessive *'s* morpheme to the possessive pronoun instead of the original referent noun phrase and return something like *her's*. Due to time constraints, this particular error was left unfixed, but the net benefit of including the coreference resolution system still outweigh this drawback.

6.3 Other improvements to readability

The J2MSummarizer also attempts to improve readability and ROUGE scores by eliminating many things from the final summary. This includes both eliminating entire sentences and removing portions of sentences. To do so, we developed a list of criteria from which we implemented a filter into the system. These criteria are discussed below.

Sentences that are extracted from a corpus tend to be written such that they fit into a given discourse structure. When used in an extractive summary, however, the elements related to that discourse have a strong tendency to reduce readability. For example, taken out of context, the sentence *'Instead, Butler, 82, has applied for a permit to hold another parade down the streets of nearby Coeur d'Alene, Idaho, in October.'* makes little sense. However, the incoherence is largely provided by the first word in the sentence, '*instead.'* So, the J2MSummarizer removes words and phrases like *instead, nonetheless, for example,* and *however* when they appear at the beginning of a sentence.

For fairly similar reasons, the J2MSummarizer also removes the headers from newspaper articles. The first sentence in a newspaper article is typically prefaced by a brief phrase that states the location, date, and publisher of the article's contents. For example, one such sentence begins, '*HAYDEN LAKE, Idaho (AP) –* .' This format has no place in an automatically generated summary, so we opt to remove it.

In order to maximize the content's brevity and relevance to the query, the J2MSummarizer also removes appositives from sentences. The information contained in appositives tends to be inappropriate for a summary, as it is often commonly-known information that does not contribute to the actual events. Therefore, the J2MSummarizer excludes it.

The J2MSummarizer also skips some sentences entirely. We deem a sentence inappropriate for the summary if it contains a phone number or quotation or if it begins with a wh- word or phrase.

To implement this filtration into the J2MSummarizer, we make use of two pieces of machinery. The majority of the tasks are accomplished using regular expressions. A smaller portion of the tasks require the use of NLTK's chunking and chinking capabilities.

7 Component Evaluation

Our system performs well in terms of the ROUGE metric on both DUC 2006 and DUC 2007 data. Our system performs less well in terms of qualitative metrics. The exact numbers are shown in Tables 1 and 2 below.

These numbers are derived from scoring our automatically-generated summaries against the gold standards provided by the DUC competition. Had we competed in the 2007 contest, we would have placed 17th overall in ROUGE performance. Compared to others in Linguistics 573, we placed 5th overall in this metric, with two groups still left to report.

Over the course of development, we made use of two primary means of evaluation. First, we compared very rough, oversized summaries against the gold standard when evaluating our

content selection methods. This was done merely to insure that the J2MSummarizer was returning the appropriate kind of content, even if it performed poorly in precision. Second, instead of concerning ourselves much with ROUGE scores in the later stages of development, we focused on improving readability. It remains to be seen if this method paid off. However, it is clear that this method did improve our precision and f-measure successfully.

Metric	Mean Score	Median Score	Standard Deviation
Grammati-cality	2.2500	2.0000	1.1990
Referential Clarity	2.5833	3.0000	0.9965
Structure-coherence	2.0833	2.0000	0.7022

Table 1. Qualitative evaluation scores

ROUGE-*	06 Score	07 Score
ROUGE-1 P(recision)	0.2884628	0.41205
ROUGE-1 R(ecall)	0.4767956	0.41158
ROUGE-1 F(-score)	0.3568712	0.41169
ROUGE-2 P	0.0616860	0.09783
ROUGE-2 R	0.1038114	0.09774
ROUGE-2 F	0.0768482	0.09776
ROUGE-SU4 P	0.1041430	0.15328
ROUGE-SU4 R	0.1744728	0.15319
ROUGE-SU4 F	0.1295142	0.15319

Table 2. ROUGE performance for 06 and 07

Unfortunately, our focus on improving readability does not seem to have been sufficient to be competitive with the others in Linguistics 573. The J2MSummarizer ranks lowest in grammaticality and structure-coherence and second-lowest in referential clarity. These results likely reflect the few remaining bugs in our regular expression systems (for grammaticality) and the coreference resolver (for referential clarity). It is currently unclear why our system performs so poorly in terms of structure-coherence. Perhaps there is an unknown bug affecting portions of our LingPipe code.

8 Future Directions

Due to time constraints, we were unfortunately unable to explore many of the directions we had initially hoped to. In this section, we detail these directions and discuss their potential implications for the overall system.

Document level boosting in Lucene allows the user to increase the weight given to specific documents. This weight is set to 1.0 by default and occurs at the time of indexing.

Document field level boosting in Lucene allows the user to increase the weight given to just some specific field of a given document or set of documents. This weight is also set to 1.0 by default and also occurs at the time of indexing.

Query level boosting in Lucene allows the user to increase the weight of some specific query terms. It is set to 1.0 by default, but occurs at search time.

All of these boosts will affect the types of sentences returned by the J2MSummarizer. Although the gains made by such adjustments might be insignificant, it would be worth exploring in future incarnations of the system.

In addition to experimenting with different Lucene parameters, our system might see a substantial gain in readability if it integrated more facts from discourse structure. For example, at this time, a coreference chain in one of our automatically-generated summaries might not be at all similar to one from a manually-generated summary. Specifically, the ways in which named entities are referred to, whether by the entire noun phrase or some anaphoric pronoun, vary a great deal between a human's and the J2MSummarizer's summaries.

In order to remedy this observation, we would first need to describe the most natural way in which a human creates a coreference chain. This task alone might be more nuanced than one might expect, or it might be as simple as using the complete noun phrase for the reference only when it is first mentioned and having subsequent mentions make use of the appropriate pronoun. Other complications that might arise include things like grammatical gender, grammatical number, and various interactions between multiple coreference chains in a single summary. Indeed, it would be difficult to implement a perfect solution to this problem, but significant improvements might be made with some partial solution.

References

Alias-i. 2008. LingPipe 3.9.2 http://alias-i.com/lingpipe (accessed October 1, 2008)

H.T. Dang. 2006. Overview of DUC 2006. In *Proceedings of the Document Understanding Conference 2006*, NIST, 2006.

John M. Conroy, Judith P. Schlesinger, Dianne P. O'Leary. 2007. Classy 2007 at DUC 2007. In *Proceedings of the Document Understanding Conference 2007*, NIST, 2007.

Christiane Fellbaum. 1998. WordNet: An Electronic Lexical Database. Cambridge, MA: MIT Press.

Sanda Harabagiu, Finley Lacatusu, and Andrew Hickl. 2006. Answering complex questions with random walk models. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Seattle, WA.

Leonhard Hennig. 2009. Topic-based Multi-document Summarization with Probabilistic Latent Semantic Analysis. In *RANLP '09*, 2009.

R. Katragadda and V. Varma. 2009. Query-focused summaries or query-biased summaries? In *Proc. ACL-IJCNLP 2009 Conference Short Papers*, Suntec, Singapore, Association for Computational Linguistics pages 105-108.

E. Loper and S. Bird. 2005. NLTK: The natural language toolkit. In *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, pages 62-69, Somerset, NJ. Association for Computational Linguistics.

D. Molla and S. Wan. 2006. Macquarie University at DUC 2006: Question answering for summarisation. In *Proceedings of DUC, 2006.*

Martin Porter. 1980. An algorithm for suffix stripping. *Program*, 14(3):130-137.