LING 573: Initial Project Report

Clara Gordon
University of Washington
Seattle, WA
cgordon1@uw.edu

Claire Jaja
University of Washington
Seattle, WA
cjaja@uw.edu

Andrea Kahn
University of Washington
Seattle, WA
andrea.m.kahn@gmail.com

Abstract

Later, this section will have the abstract - a short high-level overview of the paper, usually 150 words or so.

1 Introduction

2 System Overview

A description of the major design, methodological, and algorithmic decisions in your project. It often includes a schematic of the system architecture.

3 Approach

3.1 System architecture

3.1.1 Query processing

3.1.2 Information Retrieval

The infomation retrieval module uses the Indri/Lemur IR system to retrieve a series of twenty snippets and associated documents for each set of query terms passed to it by the query processing module. Although the Python wrapper pymur is used to index the document collection, this portion of the system uses the subprocess module to directly query Indri using command line arguments. This is necessary because certain Indri functions, such as snippet retrieval, are not available in the wrapper.

Once the snippets and associated document metadata are retrieved from standard out, a series of character-based split commands is used to segment the string into the relevant fields: document ID number, document weight, and snippet text. These fields are used to construct a Passage object. A list of these objects is then passed to the answer processing module, which uses the snippets and associated metadata to derive answer candidates.

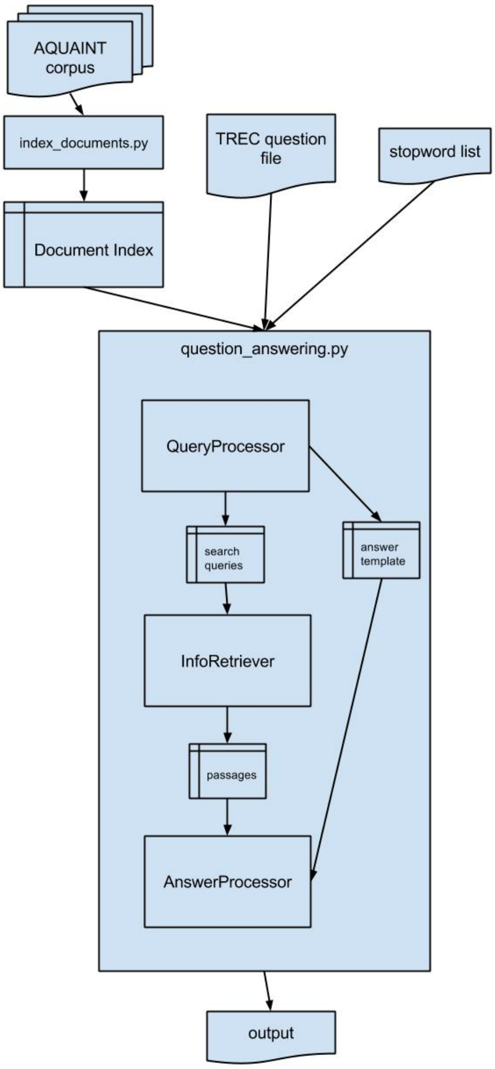

Figure 1: System architecture.

3.1.3 Answer candidate extraction and ranking

In order to extract and rank answers, the AnswerProcessor class is used. An object of this class is initialized with a list of Passage objects, an AnswerTemplate object, and an optional stopword list. This object can then generate and rank answers. This is done in a series of steps.

First, possible answers are extracted from the Passages by generating all unigrams, bigrams, trigrams, and 4-grams from the text of each passage; the score of each of these possible answers is the sum of the negated inverse of the retrieval score of the passages it is found in. If an n-gram appears multiple times in a passage, the n-gram's score is updated each time the n-gram appears, so a possible answer that appears frequently in a passage is scored higher than one that appears just once in the passage. The negated inverse of the retrieval score is used because Indri/Lemur returns a negative score where higher (i.e. closer to zero) is better; we wanted a positive score where higher (i.e. further from zero) is better for our own readability for troubleshooting. While these scores are being calculated, the document IDs where the n-grams occur are also being tracked for later use. At the end, a list of AnswerCandidate objects is generated which contains a possible answer, its score, and the documents it is found in.

After this, the AnswerCandidates go through a filtering step. At this step, any answers that start or end with a stopword or contain any words from the original query (retrieved from the AnswerTemplate) or any standalone punctuation tokens are discarded. Then, a combining step updates the score of each answer to be the current score plus the sum of the scores of the unigram answers contained within it; this prevents unigrams from being the highest ranked answers and instead favors longer answers.

Next, the answers are reweighted. At this point, any answers that did not appear in more than one passage are discarded. Additionally, this is the place where constraints on named entity type from the AnswerTemplate can be applied to increase or decrease AnswerCandidate scores; currently, since no constraints are being generated at the query processing stage, there is nothing to apply.

Lastly, the answers are ranked by score, and the top 20 are returned. When the output is generated, for each answer, one of the document IDs where it occurred is randomly selected as the source of the answer. In future development, a more clever method for selecting which document ID to use as the answer source will be employed.

4 Results

At this point, we are reporting only baseline results, shown in Table 1 below.

System	Strict	Lenient
Baseline	0.00511	0.02894

Table 1: Baseline results based on automatic pattern scoring. All scores rounded to five significant digits.

5 Discussion

Our baseline results are very low. Error analysis indicates that this is most likely the result of the low quality of the snippets returned at retrieval time. Additionally, there are some duplicate documents, so an answer may be retained as being from more than one document when in reality, it is from two identical documents and should be treated as being from one document and thus discarded.

6 Conclusion

References

Sanda M Harabagiu, Dan I Moldovan, Christine Clark, Mitchell Bowden, Andrew Hickl, and Patrick Wang. 2005. Employing two question answering systems in trec-2005. In *TREC*.

Lynette Hirschman and Robert Gaizauskas. 2001. Natural language question answering: The view from here. *Natural Language Engineering*, 7(4):275–300.

Jimmy Lin. 2007. An exploration of the principles underlying redundancy-based factoid question answering. *ACM Transactions on Information Systems (TOIS)*, 25(2):6.