The University of Washington's $UW_{CLMA}QA$ System

Dan Jinguji[†], William Lewis[†], Efthimis N. Efthimiadis[*], Joshua Minor[‡], Albert Bertram[‡], Shauna Eggers[‡], Joshua Johanson[‡], Brian Nisonger[‡], Ping Yu[‡], and Zhengbo Zhou[‡]

[‡] Computational Linguistics Master's Program
[†] Department of Linguistics
[*] Information School
University of Washington
{danjj,wlewis2,efthimis}@u.washington.edu

October 31, 2006

Abstract

The University of Washington's $UW_{CLMA}QA$ is an open-architecture Question Answering system, built around open source tools unified into one system design using customized enhancements. The goal was to develop an end-to-end QA system that could be easily modified by switching out tools as needed. Central to the system is Lucene, which we use for document retrieval. Various other tools are used, such the GoogleAPI for web boosting, fnTBL Chunker for text chunking, Lingua::Stem for stemming, Lingpipe for Named Entity Recognition, etc. We also developed several in-house evaluation tools for gauging our progress at each major milestone (e.g., document classification, document retrieval, passage retrieval, etc.) and statistical classifiers were developed that we use for various classification tasks.

1 Introduction

The University of Washington's Computational Linguistics Master's Program participated in the TREC 2006 main QA track, developing a system we call $UW_{CLMA}QA$. The system architecture is shown in Figure 1. The system was built using primarily open source tools, such as Lucene and LingPipe, which were unified into one system design using customized enhancements. We chose an open architecture because of the advantages offered by open source software, that is, no cost, and easy access to source code for customization.

The goal was to create a basic end-to-end system, with each of the subprocesses of the system designed as separable components. This would enable us to enhance individual parts of the process and check the goodness of each enhancement. The system architecture was topologically very simple, linear, as seen in Figure 1. At each intermediate point, simple flat files were used to transfer information from one subprocess to the next. This meant that as additional information was needed by some subprocess, it simply involved adding the appropriate file(s) to the input list for that subprocess. Most of the subprocesses required only a single input file, the output of the previous subprocess.

2 System Architecture and Data Flow

The XML question input file was parsed into individual topic files, separated and tagged by type, namely "factoid" and "list". Instances of the "other" question type were not included in these topic files since the

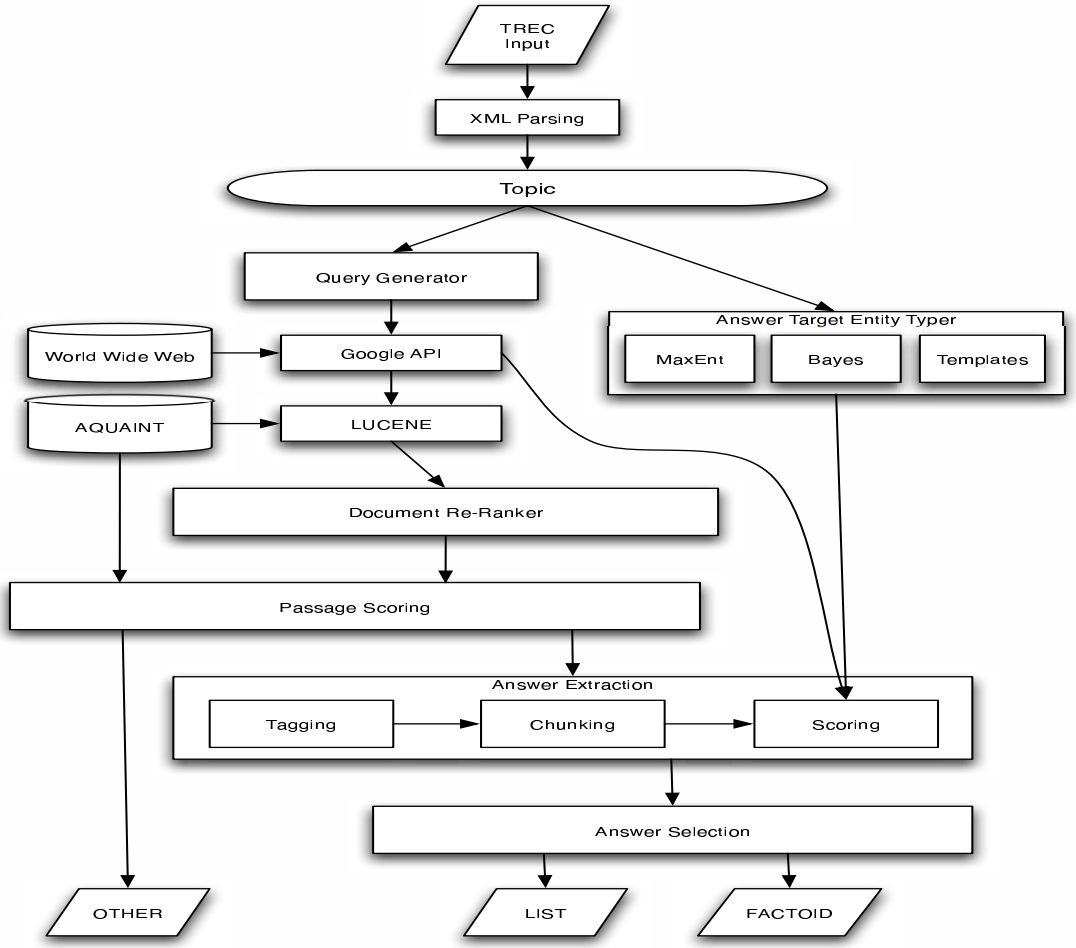

Figure 1: $UW_{CLMA}QA$ Architecture

"other" category was uniformly present and the "query text" did not provide any information germane in document retrieval, passage selection, or answer extraction.

As shown in Figure 1, the process bifurcates here. The topic files were processed to identify target types for each question (section 3). These target types were used in ranking the individual answer candidates. The topic files also fed into query processing (section 4), which involved using the GoogleAPI as a form of web-boosting (Harabagiu *et al.*, 2005). The five additional terms from the Google response were identified as a form of query expansion. The individual questions, augmented by the topic text and web-based query-expansion terms, were submitted as the Lucene query against the Aquaint database. Lucene returned a list of 1000 documents (articles) for each question. Our original plan was that this list of documents would undergo a reranking process. In the implemented system, however, this document ranking algorithm simply used the top 3 documents for "factoid" questions and the top 25 documents for "list" questions. The ranking was based on the Lucene retrieval score. The paragraphs from these documents were used as the candidates for passages, and in turn, the passages were ranked using a simple tf/idf algorithm, as discussed in section 6. There were 10 passages for each "factoid" question and 45 passages for each "list" question. This portion of the process was run in parallel for the individual topics.

After the passages were categorized by topic, each passage was then divided into noun phrase (NP) chunks. Answer candidates were drawn from the individual NP chunks (section 7). These candidate answers were ranked based on the conformance to the target types identified in initial query processing, on the similarities

to the web-boosting text, on the location of the NP within the passage, with the presence of query words counting against the score. From these ranked NP chunks, the top 1 was taken as the answer to "factoid" questions; the top group of answers were taken as the "list" answer (section 8) Our "other" answer was taken from the list of passages, sorted by tf/idf (section 6) and filtered by length. This portion of the process was also run in parallel. The NP chunking process was very resource-intensive so it did not allow the level of parallelism we were able to use in the initial processing.

In the following sections of this document, we outline each of the subprocesses in the $UW_{CMLA}QA$ system. Each section discussed what we did and also includes some background on types of enhancements we planned for the respective subprocesses.

3 Preprocessing

The Aquaint corpus is the collection of source documents from which answers are extracted, and are organized into a set of compressed files organized by the publisher (New York Times, Associated Press and Xin Hua) and by date of publication. After decompressing these files, the contents were split into separate files, one for each article, using OpenSP. The individual article files were cleaned removing all SGML markup. In the final files each paragraph occupied a single line.

A Lucene index was generated for the corpus using the default analyzer. The index was created at the article level, as well.

3.1 Potential Enhancements

There are two enhancements we considered at this stage: tagging named entities and anaphora resolution. Preliminary testing showed a marked improvement in system performance using named entity (NE) recognition to tag specific entities in the original corpus. The NEs could be tagged by type, and the resulting corpus could then be indexed. In Question Analysis (see section 4) we could then use the tagged NEs as potential targets (we do named entity analysis in Question Analysis) and the target type for each question could be determined.

Even though we were unable to run our NE system as part of our preprocess, we did build a comprehensive database of named entities for Question Analysis. The full list consists of the following:[1]

1. Biography.com - Famous people names
2. US Census Bureau - lists and frequencies of male, female and last names of people in the US in 1990
3. U.S. Securities and Exchange Commission - fairly comprehensive list of company names
4. European Commission Translation Service - list of the full and common names of countries, and appropriate adjectives for countries.
5. City Names - World Time Server
6. Misc - Manually compiled list of US states and their abbreviations, Canadian provinces and their abbreviations, and months and their abbreviations

Another promising prospect was anaphora resolution within the corpus. The antecedents could either replace the anaphora or could be placed in apposition with the anaphora.

[1] A full list of the resources we consulted for building our Named Entity database can be found at our group's wiki: http://depts.washington.edu/uwcl/twiki/bin/view.cgi/Main/TrecGroup.

4 Query Analysis

Most of the QA systems we examined from previous TREC submissions used some form of query analysis, typically identifying the type of answer being sought. Within the context of the $UW_{CLMA}QA$ project we analyzed each question and categorized them first using the top-level categories in the UIUC question classification, which are *Abbreviation*, *Description*, *Entity*, *Human*, *Location*, and *Numeric*. The UIUC categories were used since there was a body of training and evaluation data available, some 5,000+ tagged queries.

For the implemented system, we enhanced the UIUC classification scheme to include *Date* and *Measure*, which had been second-tier classification within *Numeric*, and *Country*, *State*, and *City*, which had been second-tier within *Location* in the UIUC scheme.

The implemented system used three different techniques to determine the target type for each question: one Bayesian, one maximum entropy, and one based on template matching.

4.1 Potential Enhancements

Each of these three systems used the simplest of techniques to determine question type. More robust mechanisms to determine target type could be employed. Similarly, these values are used to rank the answer candidates. This aspect will be discussed in section 8.1, where we discuss potential enhancements to answer extraction.

5 Query Processing

Starting in 2004, the questions for the TREC QA track have been arranged into topics. The questions within the topics are to be processed like a dialog, with each question having as context the topic, previous questions, and their answers.

We tried a number of experiments in query expansion, using alternate wordings. In general, we found that simply appending the topic to the end of the query gave the best results. We had evaluation metrics that checked the goodness of our document selection. The sole enhancement that seemed not to cause a degradation in the goodness of document selection was Boolean alternations of verb forms using WordNet. However, the enhancement here was negligible so was omitted in the final system.

We also enhanced our queries for document selection using web boosting. Specifically, we supplied the question and topic to Google using the GoogleAPI. The resulting passages were cleaned up: HTML mark-up was removed, query terms discarded, and stop-words were removed. The resulting output text was ranked by frequency. The document selection query was enhanced by adding the 5 most frequent terms. These web-boosted terms, which we termed "webgrams", were given a lower significance than the actual query terms and topic.

In examining the "list" questions, we found that our best results came from a heavy reliance on the topic phrase and target type information. This was accomplished by heavily weighting the words in the topic for the document selection query.

As you have noticed, all of this document selection processing occurred at the level of single words.

5.1 Potential Enhancements

One of the enhancements we could make would be to see how we could "tile" the web boosting results, generating phrases for the aggregation of counts. For example, we could individuate the counts for "Elvis" and "Presley" which then would be accumulated together based on the occurrence of the larger phrase "Elvis Presley." One danger comes in the misidentification of coreferential words and phrases, for example including "Elvis impersonator" in the counts for "Elvis Presley". The best application of this technique would probably use NP-chunking to identify both the larger phrases and the smaller ones. In this case, "Elvis impersonator" would be the chunk, not simply "Elvis".

Alternate search engines are available for web boosting work: MSN Search and Yahoo!, for example, also have programmatic search capabilities.

Question processing could also be enhanced by resolving external references, for anaphora, other pronouns, and ellipsis. This can occur in the "raw" questions, or using the answer to a previous question as the antecedent to a pronoun in the following question. This type of "dialog-system" would be supported by our architecture. In this case, rather than individual topics being submitted to the process, individual questions would be. Then the question/answer pairings from the previous questions would serve as a context for the subsequent questions. This would also alleviate the need to simply append the topic phrase to the query.

6 Document Selection

There were two steps in our document selection process. First, Lucene was used to select documents from the 1,000,000+ articles within the Aquaint corpus. Based on the Lucene rankings of the documents retrieved, we selected a small number (3) for each "factoid" question and a larger number (25) for each "list" question.

6.1 Potential Enhancements

The actual process of document retrieval using Lucene seems to work well enough. For each question, we returned a ranked list of 1,000 documents. As noted above, we did provide a hook for document reranking, after the initial document retrieval process supported by Lucene. This reranking could take into account the target type information for the question, giving preference to those documents that contain values of the target type. This would necessitate some extensive preprocessing of the corpus, tagging the entities within the corpus based on target types (see sections 3.1 and 4).

In fact, the number of documents selected for each "factoid" and "list" question was based in some quick empirical data. This threshold could be determined dynamically, based on the goodness of the retrieved documents. Alternately, it could be "learned" using some machine learning algorithm.

7 Passage Ranking

Having a relatively small number of candidate documents, we examined the individual passages, where each paragraph in the articles was considered a passage. Given the typical newspaper writing style, the passages tended to be relatively short, topically cohesive, and generally did not include pronouns without antecedents. (However, ellipsis of names is a common feature in this writing style.) Additionally, any duplicate passages were removed, since paragraphs may be repeated on subsequent days to provide background information for a breaking story.

The paragraphs were scored using tf/idf, where the complete set of articles from that news source for that day was taken as the document collection. This kept the process of tf/idf calculation relatively contained, while ensuring a wide enough distribution of topical areas to make the tf/idf score meaningful. The tf/idf score was scaled by the count of the query terms that appear within the passage.

The tf/idf implementation was novel code. Each passage (paragraph of the candidate articles) was scored. The tf term was calculated as 1 + log(word frequency) for each unique word within the passage. The idf term was calculated as log(total document count [number of articles for that day from the given news source: NYT, APW, XIE] / the number of documents that contain the given word). These tf/idf scores were summed over the words in each passage. This score was normalized by dividing by the length of the passage.

For "factoid" questions, the top 10 passages were returned; for "list" questions, the top 45.

7.1 Evaluation of Document and Passage Retrieval

During the development of our document selection routines, we trained and tested against the questions supplied for TREC 2003-2005, and we used the perl patterns and file pointers supplied by Ken Litowski (2004). The latter helped us identify relevant documents and passages in the Aquaint documents. The Litowski files contain two pieces of information useful to evaluation: the documents from which answers are derived, and an answer "pattern", expressed as a regular expression, that maps to a specific answer or set of answers that can be found in the relevant documents. Although we could not assume that Litowski's patterns were without error, we did assume that the noise introduced by error was minimal.

Our evaluation methodology worked as follows: For document ranking, we used the Litowski files to help determine how well our document ranking and reranking methods were working. For any given query, Lucene returned a set of a thousand documents. From this thousand documents, we reranked the output to the top 3 documents for factoid questions and the top 25 documents for the List questions (as mentioned in Section 6), a process that was guided by the Litowski patterns. For evaluation purposes, we held out the 2005 questions as our test set, and trained against the 2003 and 2004 questions. We could then evaluate our performance on the 2005 questions by comparing them to the results from other participants, where accuracy was measured by the precision and recall of correct documents returned in the set of the top n reranked documents (where we favored precision for "factoid" questions, and precision and recall equally (i.e., a balanced F-measure) for "list" questions). We reduced n over time as system performance improved, finally settling on the 3 and 25 for the actual implemented system. Overall, our system's precision for document selection for "factoid" questions using this evaluation metric as applied against the 2005 data was .3517 for n=3 and .3620 for n=1.[2] The overall results for factoid responses for the set of 2005 participants overall using this evaluation metric were: $\bar{x}$ = .2958 (where the the outliers below precision .05 were dropped), and the max was .7920 (which was the result for Language Computer Corporation (Harabagiu *et al.*, 2005)).

For passage retrieval, we adapted the Litowski patterns to locate the specific passages in each of the documents, and then employed a similar training, testing and evaluation methodology.

7.2 Potential Enhancements

The web boosting material ("webgrams", see section 5) could be used to further enhance the selection of passages. Our anecdotal evidence suggests that often the web boosting results contain or are close to the answers being sought. Some small "bonus" score could be given to passages which contain webgrams.

Also, the cut-off for the number of passages returned is based on empirical evidence. These values can be obtained using machine learning algorithms.

[2]It should be noted that this evaluation is only for the answers specifically supplied by NIST.

8 Answer Extraction

Examination of questions (see section 4) indicates that "factoid" answers are typically noun phrases (NPs). This is bourn out by the fact that *WH*-question words are pronouns; they replace noun phrases. So, to extract the answers, we applied NP-chunking to the paragraphs, considering each NP as a potential candidate answer.

To accomplish the NP-chunking, the passages were divided into individual sentences using Lingua::Sentence. Then, each sentence was tagged for part of speech (POS) using the Stanford POS Tagger. Having labeled the individual words in the sentence for POS, we then used the fnTBL Chunker. This chunking of the sentences, to isolate the noun phrases, was the single most computationally expensive part of the process, being both memory-intensive and time-consuming.

Having extracted noun phrases from the passages, these were assigned types, matching the question target types identified during question analysis, see section 4. This typing task made significant use of almanacs for direct loop-up, with some heuristics to identify patterned types, such as dates (see section 3.1 for a list of our sources for NEs).

We used the TREC 2005 questions as our development test set. At this point, we noticed several patterns that we used to enhance our ranking of the answer candidate NPs. For instance, given the newspaper writing style, the answer NP was twice as likely to appear in the first 10th of the sentence than in any other tenth of the sentence. Moreover, we noticed that our classification of the candidate NPs into our answer categories had some systematic errors. Our ranking system included these observations, favoring NPs from the beginning of the sentence, favoring chunks identified as the appropriate answer category, with lesser weight given to categories where we anticipated misidentification, as well as the tf/idf ranking of the original passage, and the similarity of the passage to one or more of the web-boosting terms. Terms from the query itself were heavily penalized in this ranking.

8.1 Potential Enhancements

There are several key areas where future enhancements can be made in this part of the process. We found the computational overhead of NP-chunking to be the limiting factor in our through-put; in fact, it limited the amount of testing we were able to accomplish during development, as well as limited our flexibility in scheduling the runs of the system once the 2006 question source file was downloaded. A less resource-intensive NP-chunking mechanism would greater enhance the development (and run-time) timelines.

The almanacs used to type the resulting NP chunks were adequate. One of the simplifying assumptions made at the initial stages was the very limited set of choices of target types, see section 4. A richer selection of target types, with appropriate means to recognize these types in the pool of candidate answers, would benefit this system.

Also, the processing of the answers did not include "stop words". That is, the potential answers were not filtered for obvious "non-answers", as in "Where is the Louvre?" being answered by "there".

And, as with most of the other sections, the selection of parameter values for ranking the answer candidates could be automated using one of several machine learning techniques.

9 "List" and "Other" Questions

Careful processing to target the "list" questions would require quite sophisticated semantic processing of the questions and the retrieved passages. Our approach to the "list" questions parallels the statistically based approach we used for the "factoid" questions.

In examining the "list" questions, we noted that they text of the question itself typically did little to enhance our passage retrieval or extraction of "webgrams". In fact, our identification of candidate documents worked best when we heavily weighted the topic terms within our Lucene query. This, coupled with question target-type identification, gave us the best results. For example, question 71.6 from 2005, "What countries besides U.S. fly F16s?" Topic 71 is "F16". Our best results came from searching for documents that contained "F16", more or less ignoring any other text within the query itself. Then the individual paragraphs from these documents were scored based on tf/idf, retaining the top 45 (see section 7). These passages were tagged and NP-chunked as for the "factoid" answers, with the resulting NP-chunks scored for target-type conformance. For our two runs, UWCLMA and UW574, different cut-off points were used to select the final "list" answers. These cut-off points were created empirically, based on performance over the TREC 2005 questions.

For "other" questions, we simply took the list of candidate passages for the entire topic, that is the passages selected as potentially answer-bearing for both the "factoid" and "list" questions. This was the candidate answer for "other". The duplicate passages were conflated, and their scores combined. The resulting list of passages was filtered for "overly long" passages, to keep us within the character limit for the question. The top 15 of the remaining passages was submitted as our "other" answer.

9.1 Potential Enhancements

Some initial work was done examining the list answers for internal cohesiveness and relevance. These showed some initial promise but were unable to be completed for the submission.

Given that we had relatively mediocre confidence in our answers to the "factoid" and "list" questions, we did not filter out these answers from the "other" answers. As the general goodness of the "factoid" and "list" answers improves, this would be an obvious concurrent enhancement to make.

10 Results and Conclusion

How did we do? We can provide answer based on two criteria: infrastructure, and our ratings in the competition.

On the infrastructure front, we were able to create a small modular system that should prove to be a reasonable starting point for future exploration in information retrieval, question answering, and future research and thesis work, as well as subsequent TREC submissions. Most importantly, this system provides a baseline against which the later work may be measured.

Our performance for "factoid" and "list" questions was a bit below this year's median scores. For "factoid" questions, our accuracy scores for the runs UW574 and UWCLMA were 0.112 and 0.109 respectively (best 0.578, median 0.186, worst 0.040). For "list" questions, our F-scores were 0.051 and 0.046 respectively (best 0.433, median 0.087, worst 0.000).

Our performance for "other" questions was better than the median, 0.164 and 0.153 respectively (best 0.250, median 0.125, worst 0.000). Per-question F scores for "other" questions (pyramid evaluation) was better than the median at 0.160 and 0.165 respectively (best 0.251, median 0.139, worst 0.000). While, per-series scores were just below median at 0.108 and 0.104 respectively (best 0.394, median 0.134, and worst 0.013).

Considering it was a first attempt, we are pleased with these results.

11 References

Harabagiu, Sanda, Dan Moldovan, Christine Clark, Mitchell Bowden, Andrew Hickl, and Patrick Wang. (2005). Employing Two Question Answering Systems in TREC-2005. In: The Fourteenth Text REtrieval Conference (TREC 2005) Proceedings. NIST Special Publication: SP 500-266. http://trec.nist.gov/pubs/trec14/papers/lcc-sanda.qa.pdf

Litkowksi, K.C. (2004). Use of metadata for question answering and novelty tasks. In: The Twelfth Text Retrieval Conference (TREC 2003) Proceedings. NIST Special Publication: SP 500-255. http://trec.nist.gov/pubs/trec12/papers/clresearch.qa.novelty.pdf

12 Appendix: Software Used

The software used in this project is listed below. Some were used in conjunction with others, for example: PyGoogle and SOAPy are used to access the Google API from Python.

```
SGML::Parser::OpenSP
   http://search.cpan.org/~bjoern/SGML-Parser-OpenSP-0.98/
OpenSP
   http://openjade.sourceforge.net/
TREC
   http://trec.nist.gov
Aquaint
   http://www.ai.sri.com/aquaint/
UIUC Question Classification
   http://l2r.cs.uiuc.edu/~cogcomp/Data/QA/QC/
Lucene
   http://lucene.apache.org/java/docs/index.html
SAX (Simple API for XML)
   http://www.saxproject.org/
Maxent Toolkit
   http://homepages.inf.ed.ac.uk/s0450736/maxent_toolkit.html
PyGoogle
   http://pygoogle.sourceforge.net/
SOAPy
   http://soapy.sourceforge.net/
Google API
   http://www.google.com/apis/
Lingua::Stem
   http://search.cpan.org/~snowhare/Lingua-Stem-0.82/lib/Lingua/Stem/En.pm
Lingua::Sentence
   http://search.cpan.org/~shlomoy/Lingua-EN-Sentence-0.25/lib/Lingua/EN/Sentence.pm
Stanford POS Tagger
   http://nlp.stanford.edu/software/tagger.shtml
fnTBL Chunker
   http://nlp.cs.jhu.edu/~rflorian/fntbl/
Lingpipe
   http://www.alias-i.com/lingpipe/
LevenshteinXS.pm
   http://search.cpan.org/~jgoldberg/Text-LevenshteinXS-0.03/
```