UW Question Answering 2013

Marie-Renée Arend
University of Washington
rcarend@uw.edu

Josh Cason
University of Washington
casonj@uw.edu

Anthony Gentile
University of Washington
agentile@uw.edu

Abstract

stub

1 Introduction

As a baseline, we developed a redundancy based QA system which draws on web results from the Bing search engine. We then built upon that baseline to prepare to treat the question-answering process as a classification task using the maximum entropy algorithm to learn features and weights, though we have not yet incorporated the classifier. In addition, we examined the effect of changes such as query word sense disambiguation, and added some preliminary answer pattern detecting.

2 Approach

The four main components of our redundancy baseline are query processing, web result filtering and reranking, ngram generation and filtering, and answer projection.

Tokenizing, stemming, POS tagging, a WordNet (Miller et al., 1990) interface, and a stopwords list in these modules are supported by the open-source Natural Language Toolkit (Bird, et al., 2009).

2.1 Query Modification

Often when searching the web or a document collection, it is difficult to get the correct results because of mismatch between the wording of a user's question and the wording of potential answers. To alleviate this to some degree, we incorporate a query processing unit. In our baseline iteration of our system, this module only performed the most basic processing tasks, such as stemming, removing stopwords and adding the target if not already present in the question.

In our current iteration, we attempted to use our own version of the modified Lesk algorithm as described by Huang et al. (2008). Some researchers found that WordNet senses hurt their classifier accuracy in question classification (Li and Roth, 2005). Others, however, found that when question headwords were extracted and a disambiguation algorithm was applied, question classification accuracies were high (Huang et al., 2008). Instead of just disambiguating the head word in the query, we chose to disambiguate every word in the query that can be found in WordNet. Our intention was to get a word sense disambiguation solution in place to make way for the use of synonyms and hypernyms in query processing. On one hand, it might be useful to add synonyms to the query. On the other, hypernyms could be used as features once we convert our approach to a machine learning task. For this iteration we tried the former.

The word sense disambiguation and synonym-generating code did not work as intended for this version of our QA system. Perhaps the most troublesome problem with the code's results was that the words were not properly disambiguated, leading to a large number of both relevant and irrelevant synonyms being added to the reformed query. This created too much noise in the query and, as a result, many questions that were answered in our baseline system were not answered this time around. A second major problem is that the disambiguation algorithm takes too much time to process each query. Because of these problems, we rolled back these changes for this deliverable.

To address the issues that arose with this latest version of our query expansion module, we are working on introducing some major improvements, such as increasing the efficiency and accuracy of our version of the modified Lesk algorithm and reducing the number of synonyms added to the final reformulated query to cut down on the amount of noise generated. If we are unable to improve the

Lesk algorithm to a satisfactory extent, we plan to try using a version of Resnik's word sense disambiguation algorithm (Resnik, 1995).

2.2 Web Result Filtering and Ranking

When retrieving the results of our query from the web, we want to give priority to the results that most likely have the answer we are looking for. To do this, we initially apply a weighting to the results based on the ranking of the initial web search of our query. This is done as it is likely that the answer to our question will be found in the first handful of results. Next, we determine specific answer patterns based on our query, look for those answer patterns, and for the results they appear in, give a weighted boost that can be used later in our pipeline.

Currently, we look for three different answer pattern categories: temporal, location and numerical. When a query begins with 'In what year' we can assume that our answer will match a particular temporal pattern such as four digits. Likewise, a query beginning with 'What country' can assume a location in the answer pattern. When these query clues are encountered, we check our result set for results that have answer patterns for the applicable category. These results are then given a weight boost.

We plan to convert this information into features as we incorporate the classifier. Currently, however, weight information is used in our ngram generation step (described below) to better select possible answer candidates from our web results. In particular, the score of each ngram, *m*, is the sum of the count of *m* in each web result, *r*, times the weight of *r*.

$$\sum_{r \in R} c(m, r) w(r)$$

The ranking of ngrams is a reversed sort on their scores.

2.3 N-gram Generation and Filtering

Adapting some of the techniques mentioned in (Lin, 2007), we generate ngrams from the web results and filter them. One hundred results are gleaned from the Bing API search, tokenized, and, from the title and snippet, unigrams, bigrams, trigrams, and quadrigrams are counted. Multi-token ngrams are then filtered according to the following criteria. An ngram is removed from the ranking if the ngram begins or ends with a stopword, contains words from the question, or has other problems like if there is punctuation leftover from tokenizing.

Ngrams are then reranked with the following formula.

$$S_c = S_c + \sum_{t \in c} S_t$$

The score, S_c, of a candidate answer in increased by the sum of the scores of its unigram tokens. This combats the tendency to reward shorter answers due to their natural higher frequency.

2.4 Answer Projection

The top twenty answer candidates are chosen and are used to project the answer onto the AQUAINT corpus (Graff, 2002). AQUAINT is used for evaluating QA systems, and it is necessary to find a document in it to support our web retrieved answer. We use a standard IR system Lucene (Hatcher et al., 2004) in order to retrieve the most relevant documents.

3 Results

	Del. 2	Del. 3	Del. 4
Lenient MRR	0.0455	0.0475	
Strict MRR	0.0039	0.007	

Table 1: MRR scores

4 Conclusion

stub

References

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python*. O'Reilly Media.

Graff, D. (Ed.). (2002). *The AQUAINT corpus of English news text*. Linguistic Data Consortium.

Li, X., & Roth, D. (2006). Learning question classifiers: the role of semantic information. *Natural Language Engineering*, *12*(3), 229-250.

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

Hatcher, E., Gospodnetic, O., & McCandless, M. (2004). Lucene in action.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., & Miller, K. J. (1990). Introduction to wordnet: An on-line lexical database*. *International journal of lexicography*, *3*(4), 235-244.

Resnik, Philip. (1995). Disambiguating Noun Groupings with Respect to WordNet Senses. *Third Workshop on Very Large Corpora*. Retrieved from http://acl.ldc.upenn.edu/W/W95/W95-0105.pdf