MIM QA System: Deliverable 3

Antariksh Bothale
abothale@uw.edu

Julian Chan
jchan3@uw.edu

Yi-shu Wei
yishuwei@uw.edu

Abstract

We detail all the work done on MIM, our Question Answering system. We describe the architecture and design choices of our system, along with a detailed description of the various NLP and information retrieval techniques used to improve MRR scores, such as query reformulation, question classification, web-boosting, answer re-ranking, and so on. We report Lenient/Strict MRR scores of 0.390/0.269 and 0.416/0.267 on the development and evaluation data sets, respectively.

1 Introduction

This document reports the creation and performance of an end-to-end Question Answering system called MIM that answers questions based on documents from the AQUAINT (1 or 2) corpus. We have the complete system in place for the overall task. The techniques that our system utilizes include question classification, web-boosting, answer re-ranking, and so on.

In Section 2, we present the overall architecture of MIM. In Section 3, we document the approaches and design choices in building the system. Section 4 reports final results (MRR scores) on the development and evaluation (TREC 2006 and 2007) data, together with error analysis.

2 System Overview

Our system architecture can be broken down into three major classes—MainFacilitator, TaskExecutor, and Session. Each question and answering session is represented by a Session object. For each session, the MainFacilitator runs the TaskExecutor(s) in sequence to produce the answers.

The three key tasks in the pipeline, viz. Document Retrieval, Passage Retrieval and Answer Processing, are each handled by a TaskExecutor. MainFacilitator creates a new Session object runs each of the TaskExecutor(s) sequentially. Barring any execution error, the answer will be written to the Session object.

The SystemEvaluator reads in all the question from the data set and invokes MainFacilitator. It then reads the answer from the returned Session object and writes to the output file.

The schematic diagram of our system is shown in Figure 1.

We were also able to cut down our evaluation time dramatically by running the sessions in parallel on the Condor computing cluster.

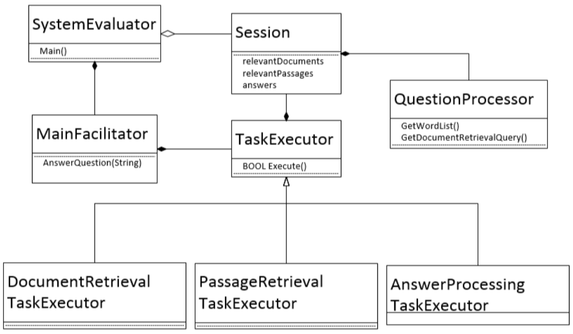

Figure 1: Class diagram of the MIM QA system

3 Approach

3.1 Question Classification

We build an SVM classifier that classifies a given question into one of the question types defined in the UIUC question taxonomy [5]. The classifier uses unigrams as well as the *Wh-* words and the headwords as features. (Headwords are extracted from parsing the questions using the Berkeley parser [2]).

We train the classifier on the dataset from [4] with 5500 labeled questions, and test the classifier on TREC 2004 and 2005 question-answering data. Since the test data are only labeled for coarse-grained question types (6 classes), we can only evaluate the classifier with coarse-grained classification, for which our SVM classifier achieves 87% accuracy. However, for our end-to-end system we retrained the classifier for fine-grained question classification (50 classes) because coarse-grained class label does not provide sufficient information for selecting correct answers.

3.2 Query Formation

In this step we convert the question into a query that can be used for document retrieval. The query indicates that the subsequent module should search for the question terms in the body text, and for the target words either in the title or the body text. We also expand the query in two ways as described below in order to increase the recall of document retrieval.

3.2.1 Morphology-Based Query Expansion

We first expand our query with verb conjugation. To achieve this, we use a dictionary of conjugations of over 8,000 verbs from NodeBox Linguistics [6]. (According to the documentation, the dictionary is converted from the tree adjoint grammar developed by UPenn.) For each verb in the question in the infinitive form, we add its past tense or 3rd person singular present tense to the query depending on the auxiliaries in the question. If there is no auxiliary in the question, we do not add any verb conjugations.

3.2.2 Redundancy-Based Web-Boosting

We implemented another query expansion strategy using redundancy-based web-boosting. The original question (together with the target words) is passed to the Ask.com search engine and the snippets of the first 40 results are extracted. From these snippets we obtain bigram counts and keep only the top 10 most frequent ones that do not contain any words from the original question. Often times, we found these top bigrams to contain the correct answer to the questions.

We then harvest unique terms from these top bigrams and add them to the query. We were able to increase our accuracy significantly (at least 10 points).

3.3 Document Retrieval

The Document Retrieval system takes a document retrieval query from the Query Formation step and returns a list of documents from the indexing and retrieval engine, as described below. This list of documents is then used for Passage Retrieval and Answer Processing.

3.3.1 Indexing and Retrieval Engine

For our indexing and retrieval engine, we use Whoosh [3], which is pure-Python based text indexing, search, and spell checking library. The choice of a Python based library above alternatives such as Lucene and Indri/Lemur was motivated by its ease of integration and use with our Python code-base, and its functionality being comparable to that of its Java counterparts.

3.3.2 Indexing Strategy

The Beautiful Soup XML parser [1] was used to parse all the documents and extract the Document ID, Headline and Body Text. Whoosh's in-built indexing engine was used to index all the documents present in the corpus. The current system uses Whoosh's default indexing mechanism, which does not process the text. The indexing schema was designed to store the Document ID and Headline with the Index, while the Body Text was indexed but not stored as it would have caused unnecessary duplication of data. Instead, the text is retrieved from the document using the Document ID whenever needed.

3.3.3 Retrieval Strategy

For each question, the documents are ranked by the Okapi BM25F scores against the query formulated in the Query Formation step above. The 100 documents with the highest BM25F scores are returned and passed down to the subsequent step (Passage Retrieval).

3.4 Passage Retrieval

This step reads the documents retrieved in the last step, extracting all passages in the documents that contain the keywords (the target words, the words in the question, and the words added from web-boosting). A "passage" in our current system is simply defined as a sentence, where sentence segmentation is handled by Whoosh. These passages are then scored based on the density of keywords in the passage. The passage extraction and scoring function are taken from Whoosh, which are originally used for returning snippets for search results.

Since we noticed that some documents are repeated in the corpus, we filter out duplicate passages which helps to increase our recall.

Additionally, we re-ranked the retrieved passages based on a bigram-match score, which is obtained by counting the number of bigram matches the target passage has with the original question, target words and top-10 web bigrams. Passages with higher match number are ranked higher.

3.5 Answer Processing

We do re-ranking based on pattern matching and the question classification result from the previous subsystem, and then chop off anything after 250 characters. If the question is classified to be of type *HUM*, any answer with named entities of type *PERSON* (as classified by NLTK) in the first 250 characters are bumped above answers without such entities. Likewise, if the question is classified to be of type *LOC*, any answer with named entities of type *GPE* or *LOCATION* in the first 250 characters are bumped above answers without such entities.

For questions of type *NUM:date*, answers that have a pattern match with a date/year pattern are bumped up.

Finally, for questions of type *NUM*, we prioritize answers with numerical patterns in them.

If there are multiple answers that satisfy any of the above criteria, we do not break ties between them and return them in the relative order in which they were returned by the passage retrieval system.

4 Results and Discussion

With top 20 answers for each question, the above-described system attains an MRR score of 0.269 under the strict evaluation scheme (0.390 under the lenient evaluation scheme) on the development (TREC 2006) data. On the final evaluation (TREC 2007) data, the system attains a strict score of 0.267 and a lenient score of 0.416.

4.1 Strategies that Are Most Helpful

We noticed three strategies that yield the most substantial performance gain of our system: First is to include the target words in our query; second is to use the whole sentence rather than automatically generated snippets as our answers; and third is to exploit search results from the World Wide Web.

4.2 Error Analysis

The following statistics (Table 1) were calculated to get a better understanding of system performance and of the sources of error.

	Dev	Eval
Questions	403	307
Questions with Patterns	386	290
Answers—Non-Zero Score	263	197
Answers—Top-10 Position	224	182
Answers—1st Position	117	91

Table 1: System Statistics.

It can thus be seen that roughly 30% of answerable questions (defined as those the pattern is present and hence the answer for which can be verified) in both DevTest and EvalTest sets are answered with a score of 1 (top answer correct).

The small difference between answers that have non-zero score and answers that are in the Top 10 indicates that the answer re-ranking works fairly well if the answer is present in the Top 20 answer candidates.

However, since a sizable number of questions remain unanswered (no viable candidate in the Top 20), it is also necessary to analyze the cause behind this. Here, we present three questions for which our system scores 0, and our analysis as to why they fell through. Note that they are all from the EvalTest set, but the analysis is general and applies equally well for the DevTest set.

Question 1: What is Krugman's Academic Specialty?
Answer: Economics
Analysis: A look at the candidate answers returned by the system reveals that while the term 'Economist' was present in almost all of them, 'Economics' wasn't. Even among answers obtained using Web Search, while over-specific answer terms such as 'Keynesian' were present, 'Economics' was very rare. This indicates the need for morphological processing of some sort, although that comes with its own set of issues with over-generalization.

Question 2: When was CAFTA signed?
Answer: May 2004
Analysis: The top answer returned by the system was relevant but wasn't complete—*"Bush signed the CAFTA... in May"*. On the other hand, web snippets contained too specific an answer—"*... the signing took place on May 28.*" For date-related answers, it might be useful to check if they contain the year too, and if not, to predict it using the document date.

Question 2: When was Rush Limbaugh born?

Answer: 1951
Analysis: This is a surprisingly egregious failure of a system that otherwise performs reasonably well, particularly since almost all the cached web snippets contain the exact answer string. The source of error, in this case, turns out to be the fact that the system uses top bigrams from the web cache for query expansion. Since the year of birth occurs with a wide range of neighboring words, it does not rise to the top as part of any bigram, and is therefore missing from the reformulated query. An obvious solution is to score unigrams too, but that tends to promote a lot of spurious terms as well.

5 Conclusion

We have achieved fairly good results by improving key processes in the QA pipeline. We also realized (from our experience and also from that of other teams') that techniques like n-gram scoring only gave marginal improvement (if any) and we dropped them in favor of our current system.

In the future, there are two components that we would like to improve on. The first is the document indexing and retrieval strategies, where we expect the recall of document retrieval will be further improved with more fuzzy matching techniques like stemming/lemmatization. The second is answer scoring and re-ranking, where we would like to incorporate more deep, linguistically motivated techniques such as Minimal Recursive Semantics (MRS) or other semantic representations.

Acknowledgments

We thank C.-J. Hsu for the code snippet for interfacing with Ask.com for the Web Boosting mode that he shared with all students on GoPost.

References

[1] *Beautiful Soup*. URL: http://www.crummy.com/software/BeautifulSoup/.

[2] *Berkeley parser*. URL: http://nlp.cs.berkeley.edu/Software.shtml.

[3] Matt Chaput. *Whoosh*. URL: https://pypi.python.org/pypi/Whoosh/.

[4] Xin Li and Dan Roth. *Learning Question Classifiers*. URL: http://cogcomp.cs.illinois.edu/Data/QA/QC/.

[5] Xin Li and Dan Roth. "Learning question classifiers". In: *Proceedings of the 19th International Conference on Computational Linguistics, Vol. 1*. Association for Computational Linguistics. 2002, pp. 1–7.

[6] *NodeBox Linguistics*. URL: http://nodebox.net/code/index.php/Linguistics.