Data-Intensive Question Answering

Eric Brill, Jimmy Lin, Michele Banko, Susan Dumais and Andrew Ng
Microsoft Research
One Microsoft Way
Redmond, WA 98052
{brill, mbanko, sdumais}@microsoft.com
jlin@ai.mit.edu; ang@cs.berkeley.edu

1 Introduction

Microsoft Research Redmond participated for the first time in TREC this year, focusing on the question answering track. There is a separate report in this volume on the Microsoft Research Cambridge submissions for the filtering and Web tracks (Robertson et al., 2002). We have been exploring data-driven techniques for Web question answering, and modified our system somewhat for participation in TREC QA. We submitted two runs for the main QA track (AskMSR and AskMSR2).

Data-driven methods have proven to be powerful techniques for natural language processing. It is still unclear to what extent this success can be attributed to specific techniques, versus simply the data itself. For example, Banko and Brill (2001) demonstrated that for confusion set disambiguation, a prototypical disambiguation-in-string-context problem, the amount of data used far dominates the learning method employed in improving labeling accuracy. The more training data that is used, the greater the chance that a new sample being processed can be trivially related to samples appearing in the training data, thereby lessening the need for any complex reasoning that may be beneficial in cases of sparse training data.

The idea of allowing the data, instead of the methods, do most of the work is what motivated our particular approach to the TREC Question Answering task. One of the biggest challenges in TREC-style QA is overcoming the surface string mismatch between the question formulation and the string containing its answer. For some Question/Answer pairs, deep reasoning is needed to relate the two. The larger the data set from which we can draw answers, the greater the chance we can find an answer that holds a simple, easily discovered relationship to the query string.

Our approach to question answering is to take advantage of the vast amount of text data that is now available online. In contrast to many question answering systems that begin with rich linguistic resources (e.g., parsers, dictionaries, WordNet), we begin with data and use that to drive the design of our system. To do this, we first use simple techniques to look for answers to questions on the Web. Since the Web has orders of magnitude more data than the TREC QA document collection, simple techniques are likely to work here. After we have found suitable answer strings from online text, we project them onto the TREC corpus in search of supporting documents.

2 Answer Redundancy and Question Answering

Answer redundancy (multiple, differently phrased, answer occurrences) serves two purposes for our task of question answering. First, the occurrence of multiple linguistic formulations of the same answer increases the chances of being able to find an answer that occurs within the context of a simple pattern matching the query. For instance, it is not difficult to match the question "Who killed Abraham Lincoln" with the text "John Wilkes Booth killed Abraham Lincoln," but it is more challenging to find the answer to this question in the text "John Wilkes Booth is perhaps America's most infamous assassin. He is best known for having fired the bullet that ended Abraham Lincoln's life."

The TREC corpus has considerably less answer redundancy than the Web – the TREC QA database consists of fewer than 1 million documents, whereas Web search engines are now indexing more than 2 billion pages. By analyzing the set of documents returned by the union of all groups, we see that only 37 of the TREC 2001 queries have 25 or more documents with a correct answer, and only 138 have 10 or more documents. Given a source, such as the TREC corpus, that contains only a relatively small number of formulations of answers to a query, we may be faced with the difficult task of mapping questions to answers by way of uncovering complex lexical, syntactic, or semantic relationships between question string and answer string. The need for anaphor resolution and synonymy, the presence of alternate syntactic formulations and indirect answers all make answer finding a potentially challenging task. However, the greater the answer redundancy in the source, the more likely it is that we can find an answer that occurs in a simple relation to the question, and therefore, the less likely it is that we will need to resort to solving the aforementioned difficulties facing natural language processing systems.

The second use of answer redundancy is to facilitate answer extraction. Even if we find a simple relationship between the question and the proposed answer, the answer might be incorrect. It is possible that the source made a mistake, or that the seemingly correct answer string appears in a context that identifies it as possibly incorrect (e.g. "John thinks that Andrew Jackson killed Abraham Lincoln"). Additionally, even with a highly redundant information source, there will be questions for which no simple-relationship answer can be found. To lessen these challenges, we can use answer redundancy to combine a number of uncertain guesses into a single, much more reliable guess. This is the kind of redundancy explored in Abney et al. (2000), Clarke et al. (2001) and Kwok et al. (2001).

3 System Overview

Our system utilizes a search engine[1] to find answers on the Web, an approach similar to that described in Kwok et al. (2001). Given a question, we formulate multiple queries to send to the search engine, we ask for the 100 best matching pages for each, and then harvest the returned summaries for further processing. A set of potential answers is extracted from the summary text, with each potential answer string weighted by a number of factors, including how well it matches the expected answer type and how often it occurred in the retrieved page summaries.

Given a set of possible answers, we then perform *answer projection*, searching for supporting documents in the TREC QA document collection. The system returns the four best <answer, document ID> pairs. We made no attempt to determine when an answer did not exist in the TREC corpus; instead we always returned "NIL" in the fifth position. A flow diagram of our system is shown in Figure 1. Below we discuss each component in detail.

3.1 Query Reformulation

Given a query Q, we would like to search our document collection for possible answer strings S. To give a simple example, from the question "When was Abraham Lincoln born?" we know that a likely answer formulation takes the form "Abraham Lincoln was born on <DATE>". Therefore, we can look through the data, searching for such a pattern. While it may be possible to learn query-to-answer reformulations (e.g., Agichtein et al., 2001; Radev et al., 2001), we created these manually. We did not use a parser or part-of-speech tagger for query reformulation, but did use a lexicon in order to determine the possible parts-of-speech of a word as well as its morphological variants.

We first classify the question into one of seven categories, each of which is mapped to a particular set of rewrite rules. Rewrite rule sets ranged in size from one to five rewrite types. The output of the rewrite module is a set of 3-tuples of the form [string, L/R/-, weight], where "string" is the reformulated search query, "L/R/-" indicates the position in the text where we expect to find the answer with respect to the query string (to the left, right or anywhere) and

[1] For the experiments reported here, we used Google as the backend Web search engine.

"weight" reflects how much we prefer answers found with this particular query. The idea behind using a weight is that answers found using a high precision query (e.g. "Abraham Lincoln was born on") are more likely to be correct than those found using a lower precision query (e.g. "Abraham" "Lincoln" "born").

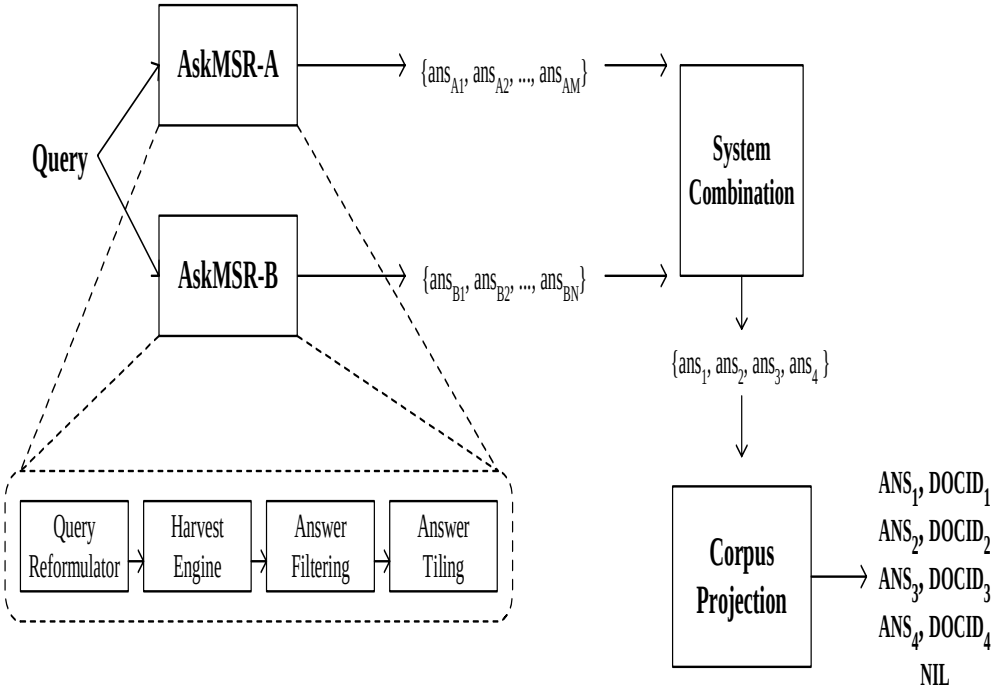

Figure 1. AskMSR System Architecture

The rewrites generated by our system were simple string-based manipulations. For instance, some question types involve query rewrites with possible verb movement; the verb "*is*" in the question "Who **is** the world's richest man married to?" should be moved in formulating the desired rewrite to "The world's richest man **is** married to". While we might be able to determine where to move a verb by analyzing the sentence syntactically, we took a much simpler approach. Given a query such as "Who is w_1 w_2 … w_n", where each of the w_i is a word, we generate a rewrite for each possible position the verb could be moved to (e.g. "w_1 is w_2 … w_n", "w_1 w_2 is … w_n", etc). While such an approach results in many nonsensical rewrites (e.g. "the world's is richest man married to"), these very rarely result in the retrieval of bad pages, and the proper movement position is guaranteed to be found via exhaustive search. If we instead relied on a parser, we would require fewer query rewrites, but a misparse would result in the proper rewrite not being found. We currently use only simple string matching, but could enhance our rewrites to include richer patterns as Soubbotin and Soubbotin (2002) have done.

The rewrites for the query "*What is relative humidity?*" are:

["+is relative humidity", LEFT, 5]
["relative +is humidity", RIGHT, 5]
["relative humidity +is", RIGHT, 5]
["relative humidity", NULL, 2]
["relative" AND "humidity", NULL, 1]

3.2 N-Gram Harvesting

Once we have obtained the set of rewrites, we submit each reformulated query to the search engine. For efficiency reasons, we worked only with the summaries returned for each hit rather than retrieving the full-text of pages (as was done by Kwok et al. (2001) and Clarke et al. (2002)).

The returned summaries contain the query terms, usually with a few words of surrounding context. The summary text is then processed to retrieve only strings to the left or right of the query string, as specified in the rewrite triple. In some cases, this surrounding context has truncated the answer string, which may negatively impact our results.

We obtain 1-gram, 2-grams and 3-grams from the short summaries. We score each n-gram according to the weight of the query that retrieved that it and sum these weights across all summaries containing the n-gram. So, the weight for each candidate n-gram is given by:

$$ngram\_weight = \sum_{ngram \in summaries} rewrite\_weight$$

There are a couple of important things to note about this weighting scheme. First, we count an n-gram only once within each summary, so there is no *tf* component. Second, the more summaries an n-gram occurs in the higher weight it gets, which is the opposite of the usual *idf* approaches to term weighting. Shorter n-grams will occur more often, but we use tiling to increase the counts for longer n-grams, as described below. Because we do not use any global term weights, we do not need to index the documents directly nor maintain a local database of term weights.

When searching for candidate answers, we enforce the constraint that stop words are not permitted to appear in any potential n-gram answers. In retrospect, this was too stringent a requirement.

3.3 Answer Typing

Next, we use type filters to increment/decrement each n-gram count based on expected type (gleaned from the question) and a guess as to the type of the n-gram. The system uses filtering in the following manner. First, the query is analyzed and assigned one of seven question types, such as *who-question*, *what-question*, or *how-many-question*. Based on the query type that has been assigned, the system determines what collection of filters to apply to the set of potential answers found during n-gram harvesting. The answers are analyzed for features relevant to the filters, and then rescored according to the presence of such information.

A collection of approximately 15 filters were developed based on human knowledge about question types and the domain from which their answers can be drawn. Most filters used surface string features, such as capitalization or the presence of digits, and consisted of hand-crafted regular expression patterns. Some filters were driven by more sophisticated properties such as semantic features or part-of-speech assignments, and used natural language analysis (Jensen et al., 1993) capable of associating such characteristics with strings. For example, these filters indicate that the strings “Pope Julius”, “Julius II”, and “David” refer to people, whereas “Vatican” refers to a location, which will be helpful for correctly answering *who-* or *where-question*s.

The selected filters are applied to each candidate string and used to adjust the initial score of the string. In most cases, filters are used to boost the score of a potential answer when it has been determined to possess the features relevant to the query type. In other cases, filters are used to remove strings from the candidate list altogether. This type of exclusion was only performed when the set of correct answers was determined to be a closed set (e.g. “Which continent….?”) or definable by a set of closed properties (e.g. “How many…?”).

The filters were determined to yield 26.4% relative improvement in MRR on a held-out subset of TREC9 queries, compared to using no type filter re-weighting.

3.4 Answer Tiling

Finally, we applied an answer tiling algorithm, which both merges similar answers and assembles longer answers out of answer fragments. Tiling constructs longer n-grams from sequences of overlapping shorter n-grams. For example, "A B C" and "B C D" is tiled into "A B C D." The weight of the new n-gram is the maximum of the constituent n-gram weights. The algorithm proceeds greedily from the top-scoring candidate - all subsequent candidates (up to a certain

cutoff) are checked to see if they can be tiled with the current candidate answer. If so, the higher scoring candidate is replaced with the longer tiled n-gram, and the lower scoring candidate is removed. The algorithm stops only when no n-grams can be further tiled.

4 System Combination

We had developed two semi-independent versions of the system, differing in the set of rewrite rules, tiling algorithm and type filters. It has been demonstrated in many settings that, given several algorithms for a prediction problem, combining their results via a voting scheme can frequently result in performance better than that of any of the individual algorithms, since different systems can reinforce each other's strengths and also help correct each other's wrong answers. Towards realizing such gains for our system, we learned an automatic method for combining the results from the two systems (AskMSR-A and AskMSR-B).

The first step in combining the answers was to run through AskMSR-A and AskMSR-B's lists of outputs to determine when two of their answers can be deemed equivalent and hence should be merged. This is a step that can frequently be omitted by most voting schemes that have to deal with only a small set of possible output values (rather than the large set of all possible strings in our setting), and we note that determining whether two answers match is a more subtle task than might appear at first blush. For instance, exact string matching will fail to recognize that ``redwood trees" and ``redwoods'' are almost certainly the same thing; simple substring matching also fails on this example.

In our system, we tested whether two answers A and B matched by checking if either every stem of every word in A matches a stem of some word in B, or vice versa. Armed with this test, we then merge AskMSR-A and AskMSR-B's lists of answers into a single, combined list, as follows: We initialize the ``combined list" to the empty list, and then repeatedly test pairs of answers (one from AskMSR-A, one from AskMSR-B) to see if they match; upon finding a match, both answers are deleted from their respective lists, and the (lexicographically) longer answer is added to the combined list. When no more pairs of matches are to be found, we then add the answers still remaining in the AskMSR-A and the AskMSR-B lists to the combined list.

Having formed a combined list of answers, we then learn a way for ranking them. For almost any learning method, the choice of features critically impacts performance. In our specific application, we desire features that can be used to characterize how ``confident" we are about each answer's correctness, so that we can rank the answers we are confident about higher. While AskMSR-A and AskMSR-B both output their own confidence scores, the absolute values of these scores were only very weakly predictive of confidence. On the other hand, the rankings of the answers output by each of the two methods were far more predictive of confidence; this can also be thought of as using the relative, rather than absolute, values of these scores.

We therefore learned a function that took as input the rankings of an answer output by either or both algorithms, and whose task it was then to output a ``score" determining how confident we are that this answer is correct. Here, these ``scores" have no intrinsic meaning (such as the probability of being correct), and our goal is only that when the results are sorted according to the scores, that the resulting expected MRR be high.

Using TREC-9 QA queries 201-400 as our training data, the parameters of our function approximator were automatically tuned to maximize the empirical MRR on the training set. On holdout test data, we estimated that this method improved our overall MRR by 11% over the better of AskMSR-A and AskMSR-B.

5 Answer Projection

At this point the system has produced a list of the n-best answers for a question. These answers were determined using web data. The next task was to find supporting documents in the TREC document collection for each answer candidate. In the projection phrase, five possible supporting documents are found for each of the five candidate answers. The Okapi IR system was used for

finding the supporting documents for each candidate answer (Robertson et al., 1995). The query submitted to Okapi was just the list of query words along with the candidate answer. Documents were ranked using the standard best match ranking function, bm25. We did not use any phrase or proximity operators to increase precision nor any pseudo relevance feedback to increase coverage. We did not use Boolean operators to ensure that the candidate answer be matched.

To generate the final answers, the first supporting document for each candidate answer was chosen, unless there existed a supporting document for the candidate answer that was also retrieved as the supporting document for another candidate answer, in which case the duplicate supporting document is returned. For example, if a candidate answer had supporting documents d1, d2, etc., d2 is returned if another candidate answer is supported by d2. The reasoning behind this strategy is that candidate answers tend to be related to the correct answer, and multiple occurrences of a document suggested that the document contain either the answer or terms related to the answer. In practice, however, this mechanism was rarely used - almost all supporting documents returned were the first one.

Although we designed the answer projection component to work for the TREC QA track, we believe it is more generally applicable. For example, if one had a small reliable source like an encyclopedia, newspaper, or help documentation, one could use the same idea – first find possible answers using our simple system in a large noisy collection like the Web and then project the answers to the reliable sources for verification.

6 Results and Analysis

We present the official TREC 2001 results for our two submitted runs, AskMSR and AskMSR2, in the table below. We used exactly the system described above for the AskMSR run. For AskMSR2, we used a somewhat different projection algorithm than described above, which improved performance on our TREC9 hold-out set but had little impact on the actual test data, as shown in the table. The average answer length was 14.6 bytes for both systems, well below the 50 byte limit. Since we had no training data to calibrate the system scores, we did nothing to handle NIL queries, and simply placed a NIL response in position 5 for every query.

System	*Strict*	*Lenient*
AskMSR		
MRR	0.347	0.434
% no answers	49.2	40.0
AskMSR2		
MRR	0.347	0.437
% no answers	49.6	39.6

Table 1. TREC 2001 results

We were quite pleased with the results of our very simple system in our first participation in the TREC QA track. Although we had been working on Web QA for a few months, our entire TREC QA endeavor was done, from scratch, in two weeks. There is still a great deal that can be done to improve the system. One of the biggest weaknesses of our system was the simple strategy we used to map an answer onto a supporting document, as seen in our .09 drop in MRR from finding an answer to finding a supporting document for that answer. Clarke et al. (2002) and Buchholz (2002) also report lower TREC performance compared to Web performance. A number of projection errors came from the temporal differences in the Web and TREC collections. E.g., for query *1202: Who is the governor of Alaska?,* we return Tony Knowles, who is the governor in 2001, but not Steve Cowper who was the governor in 1989.

There were several other bugs and sub-optimal design decisions in our initial TREC QA system. One problem was our decision not to include stop words in the n-gram strings (e.g., For query *1358: In which state would you find the Catskill Mountains?,* our top answer was ‘Regional York

State', but we omitted 'New' because it was a stop word. We have removed this constraint in our current system and it improves performance considerably. Other problems occurred in answer tiling (e.g., For query *1288: What is nepotism?*, our top answers were: 'favoritism shown'; 'relatives'; and 'employment' which we did not tile correctly because 'to' and 'in' were linking stop words that we removed. Other areas for improvement are: handling of quantities, answer typing, and question reformulation, which are useful more broadly than the TREC question-answering task.

There are several examples where our simple approach does quite well compared to other systems. Typically these are cases where simple rewrites work well with the large Web collection but much more complex processing is required to find answers within the small TREC document collection. Consider the following query-document pairs. (These are the only relevant documents for these queries within the TREC collection.)

1083: What is the birthstone for June? <answer: pearl>

<DOC> … *For anyone fascinated by pearls who wants to learn more about them, a tiny but magical London jewellery shop, Manguette, is having a festival of pearls (faux and real) for two weeks during June (the pearl is the birth-stone for those born in that month).…* Only three groups find this document. There are two difficulties in finding this document in the TREC collection -- pronominal reference must be used to know that 'that month' refers to June, and the query term birthstone needs to be rewritten as birth-stone which occurs in the document. With the wealth of data available on the Web, we can find the answer without solving either of these problems.

1340: What is the rainiest place on Earth? <answer: Mount Waialeale>

<DOC> … *In misty Seattle, Wash., last year, 32 inches of rain fell. Hong Kong gets about 80 inches a year, and even Pago Pago, noted for its prodigious showers, gets only about 196 inches annually. … (The titleholder, according to the National Geographic Society, is Mount Waialeale in Hawaii, where about 460 inches of rain falls each year.).…* Again, only three groups find this document. This is a much more interesting case. Some fairly sophisticated processing needs to be done to know that titleholder means rainiest.

After submitting our TREC run, we continued to improve the system for general web QA capabilities. After receiving the TREC relevance judgments, we tried the new system on the TREC queries, and were pleased to see some sizable improvements. We analyzed our new system on the 30 "worst" questions for our system – that is, the questions with the greatest difference between mean score across groups and our score for a question. On these 30 questions, our official submission attained an MRR of 0. The improved system attained an MRR of 0.390 on these 30 questions. There would be improvements on other queries as well although we have not scored the full set by hand.

We believe that data redundancy is a readily available and valuable resource that should be exploited for question answering in much the same way as linguistic resources often are. The performance of our system shows promise for approaches to question answering which make use of very large text databases, even with minimal natural language processing.

References

S. Abney, M. Collins and A. Singhal (2000). Answer extraction. In *Proceedings of ANLP 2000*.

E. Agichtein, S. Lawrence and L. Gravano (2001). Learning search engine specific query transformations for question answering. In *Proceedings of WWW10*.

M. Banko and E. Brill; Scaling to very very large corpora for Natural Language Disambiguation. In *Proceedings of ACL*, 2001.

S. Buchholz (2002). Using grammatical relations, answer frequencies and the World Wide Web for TREC question answering. To appear in *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*.

C. Clarke, G. Cormack and T. Lyman (2001). Exploiting redundancy in question answering. In *Proceedings of SIGIR'2001*.

C. Clarke, G. Cormack and T. Lynam (2002). Web reinforced question answering. To appear in *Proceedings of the Tenth Text REtrieval Conference (TREC 2001).*

C. Kwok, O. Etzioni and D. Weld (2001). Scaling question answering to the Web. In *Proceedings of WWW'10.*

Jensen, K., Heidorn, G.E., and Richardson, S.D. eds. 1993. *Natural Language Processing: The PLNLP Approach.* Kluwer Academic Publishers.

D. R. Radev, H. Qi, Z. Zheng, S. Blair-Goldensohn, Z. Zhang, W. Fan and J. Prager (2001). Mining the web for answers to natural language questions. In *ACM CIKM 2001: Tenth International Conference on Information and Knowledge Management*.

S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu and M. Gatford (1995). Okapi at TREC-3. In *Proceedings of the Third Text REtrieval Conference (TREC-3).*

S. E. Robertson, S. Walker and H. Zaragoza (2002). Microsoft Cambridge at TREC-10: Filtering and web tracks. To appear in *Proceedings of the Tenth Text REtrieval Conference (TREC 2001).*

M. M. Soubbotin and S. M. Soubbotin (2002). Patterns and potential answer expressions as clues to the right answers. To appear in *Proceedings of the Tenth Text REtrieval Conference (TREC 2001).*

E. Voorhees and D. Harman, Eds. (2001). Proceedings of the Ninth Text REtrieval Conference (TREC-9).

E. Voorhees and D. Harman, Eds. (2002). Proceedings of the Tenth Text REtrieval Conference (TREC 2001).