An Analysis of the AskMSR Question-Answering System

Eric Brill, Susan Dumais and Michele Banko
Microsoft Research
One Microsoft Way
Redmond, Wa. 98052
{brill,sdumais,mbanko}@microsoft.com

Abstract

We describe the architecture of the AskMSR question answering system and systematically evaluate contributions of different system components to accuracy. The system differs from most question answering systems in its dependency on data redundancy rather than sophisticated linguistic analyses of either questions or candidate answers. Because a wrong answer is often worse than no answer, we also explore strategies for predicting when the question answering system is likely to give an incorrect answer.

1 Introduction

Question answering has recently received attention from the information retrieval, information extraction, machine learning, and natural language processing communities (AAAI, 2002; ACL-ECL, 2002; Voorhees and Harman, 2000, 2001). The goal of a question answering system is to retrieve answers to questions rather than full documents or best-matching passages, as most information retrieval systems currently do. The TREC Question Answering Track, which has motivated much of the recent work in the field, focuses on fact-based, short-answer questions such as "*Who killed Abraham Lincoln?*" or "*How tall is Mount Everest?*" In this paper we describe our approach to short answer tasks like these, although the techniques we propose are more broadly applicable.

Most question answering systems use a variety of linguistic resources to help in understanding the user's query and matching sections in documents. The most common linguistic resources include: part-of-speech tagging, parsing, named entity extraction, semantic relations, dictionaries, WordNet, etc. (e.g., Abney et al., 2000; Chen et al. 2000; Harabagiu et al., 2000; Hovy et al., 2000; Pasca et al., 2001; Prager et al., 2000). We chose instead to focus on the Web as a gigantic data repository with tremendous redundancy that can be exploited for question answering. We view our approach as complimentary to more linguistic approaches, but have chosen to see how far we can get initially by focusing on data per se as a key resource available to drive our system design. Recently, other researchers have also looked to the web as a resource for question answering (Buchholtz, 2001; Clarke et al., 2001; Kwok et al., 2001). These systems typically perform complex parsing and entity extraction for both queries and best matching Web pages, and maintain local caches of pages or term weights. Our approach is distinguished from these in its simplicity and efficiency in the use of the Web as a large data resource.

Automatic QA from a single, small information source is extremely challenging, since there is likely to be only one answer in the source to any user's question. 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 data collection, the more likely it is that we can find an answer that occurs in a simple relation to the question. Therefore, the less likely it is that we will need to solve the aforementioned difficulties facing natural language processing systems.

In this paper, we describe the architecture of the AskMSR Question Answering System and evaluate contributions of different system components to accuracy. Because a wrong answer is often worse than no answer, we also explore strategies for predicting when the question answering system is likely to give an incorrect answer.

2 System Architecture

As shown in Figure 1, the architecture of our system can be described by four main steps: query-reformulation, n-gram mining, filtering, and n-gram tiling. In the remainder of this section, we will briefly describe these components. A more detailed description can be found in [Brill et al., 2001].

2.1 Query Reformulation

Given a question, the system generates a number of weighted rewrite strings which are likely substrings of declarative answers to the question. For example, "*When was the paper clip invented?*" is rewritten as "*The paper clip was invented"*. We then look through the collection of documents in search of such patterns. Since many of these string rewrites will result in no matching documents, we also produce less precise rewrites that have a much greater chance of finding matches. For each query, we generate a rewrite which is a backoff to a simple ANDing of all of the non-stop words in the query.

The rewrites generated by our system are simple string-based manipulations. We do not use a parser or part-of-speech tagger for query reformulation, but do use a lexicon for a small percentage of rewrites, in order to determine the possible parts-of-speech of a word as well as its morphological variants. Although we created the rewrite rules and associated weights manually for the current system, it may be possible to learn query-to-answer reformulations and their weights (e.g., Agichtein et al., 2001; Radev et al., 2001).

2.2 N-Gram Mining

Once the set of query reformulations has been generated, each rewrite is formulated as a search engine query and sent to a search engine from which page summaries are collected and analyzed. From the page summaries returned by the search engine, n-grams are collected as possible answers to the question. For reasons of efficiency, we use only the page summaries returned by the engine and not the full-text of the corresponding web page.

The returned summaries contain the query terms, usually with a few words of surrounding context. The summary text is processed in accordance with the patterns specified by the rewrites. Unigrams, bigrams and trigrams are extracted and subsequently scored according to the weight of the query rewrite that retrieved it. These scores are summed across all summaries containing the n-

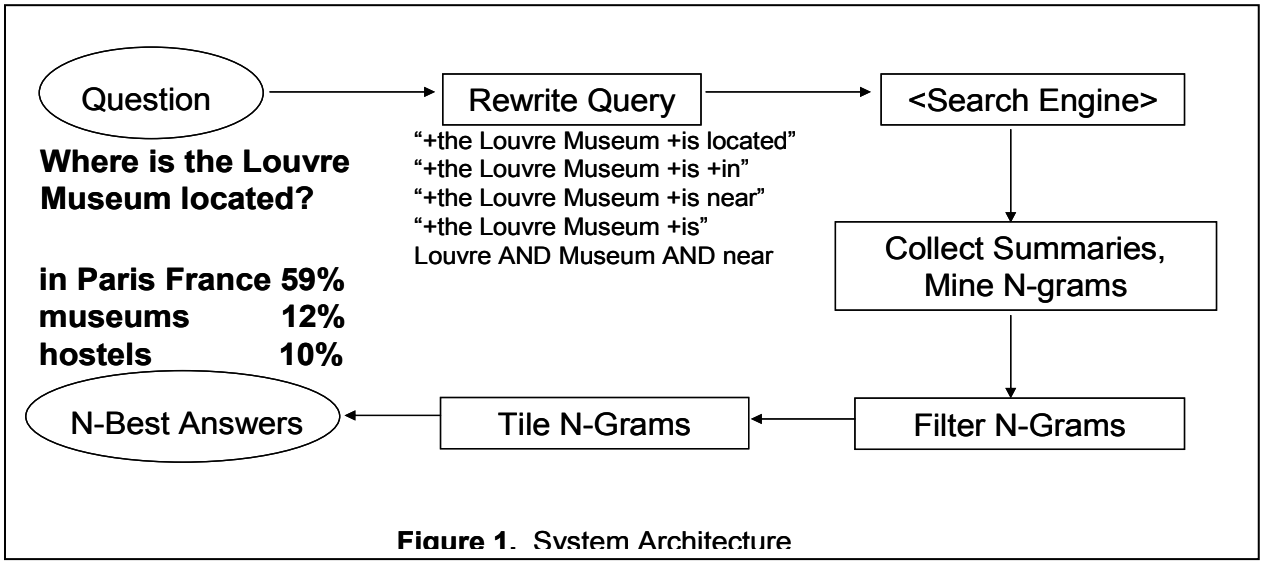

Figure 1. System Architecture

gram (which is the opposite of the usual inverse document frequency component of document/passage ranking schemes). We do not count frequency of occurrence within a summary (the usual tf component in ranking schemes). Thus, the final score for an n-gram is based on the weights associated with the rewrite rules that generated it and the number of unique summaries in which it occurred.

2.3 N-Gram Filtering

Next, the n-grams are filtered and reweighted according to how well each candidate matches the expected answer-type, as specified by a handful of handwritten filters. 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 the collection of n-grams. The candidate n-grams are analyzed for features relevant to the filters, and then rescored according to the presence of such information.

A collection of 15 simple filters were developed based on human knowledge about question types and the domain from which their answers can be drawn. These filters used surface string features, such as capitalization or the presence of digits, and consisted of handcrafted regular expression patterns.

2.4 N-Gram Tiling

Finally, we applied an answer tiling algorithm, which both merges similar answers and assembles longer answers from overlapping smaller answer fragments. For example, "A B C" and "B C D" is tiled into "A B C D." 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.

3 Experiments

For experimental evaluations we used the first 500 TREC-9 queries (201-700) (Voorhees and Harman, 2000). We used the patterns provided by NIST for automatic scoring. A few patterns were slightly modified to accommodate the fact that some of the answer strings returned using the Web were not available for judging in TREC-9. We did this in a very conservative manner allowing for more specific correct answers (e.g., Edward J. Smith vs. Edward Smith) but not more general ones (e.g., Smith vs. Edward Smith), and also allowing for simple substitutions (e.g., 9 months vs. nine months). There also are substantial time differences between the Web and TREC databases (e.g., the correct answer to *Who is the president of Bolivia?* changes over time*),* but we did **not** modify the answer key to accommodate these time differences, because it would make comparison with earlier TREC results impossible. These changes influence the absolute scores somewhat but do not change relative performance, which is our focus here.

All runs are completely automatic, starting with queries and generating a ranked list of 5 candidate answers. For the experiments reported in this paper we used Google as a backend because it provides query-relevant summaries that make our n-gram mining efficient. Candidate answers are a maximum of 50 bytes long, and typically much shorter than that. We report the Mean Reciprocal Rank (MRR) of the first correct answer, the Number of Questions Correctly Answered (NAns), and the proportion of Questions Correctly Answered (%Ans).

3.1 Basic System Performance

Using our current system with default settings we obtain a MRR of 0.507 and answers 61% of the queries correctly (Baseline, Table 1). The average answer length was 12 bytes, so the system is returning short answers, not passages. Although it is impossible to compare our results precisely with TREC-9 groups, this is very good performance and would place us near the top of 50-byte runs for TREC-9.

3.2 Contributions of Components

Table 1 summarizes the contributions of the different system components to this overall performance. We report summary statistics as well as percent change in performance when components are removed (%Drop MRR).

Query Rewrites:
As described earlier, queries are transformed to successively less precise formats, with a final backoff to simply ANDing all the non-stop query terms. More precise queries have higher weights associated with them, so n-grams found in these responses are given priority. If we set all the rewrite weights to be equal, MRR drops from 0.507 to 0.489, a drop of 3.6%. Another way of looking at the importance of the query rewrites is to examine performance where the only rewrite the system uses is the backoff AND query. Here the drop is more substantial, down to 0.450 which represents a drop of 11.2%.

Query rewrites are one way in which we capitalize on the tremendous redundancy of data on the web – that is, the occurrence of multiple linguistic formulations of the same answers increases the chances of being able to find an answer that occurs within the context of a simple pattern match with the query. Our simple rewrites help compared to doing just AND matching. Soubbotin and Soubbotin (2001) have used more specific regular expression matching to good advantage and we could certainly incorporate some of those ideas as well.

	MRR	NAns	%Ans	%Drop MRR
Baseline	0.507	307	61.4%	0.0%
Query Rewrite:				
Same Weight All Rewrites	0.489	298	59.6%	3.6%
AND-only query	0.450	281	56.2%	11.2%
Filter N-Gram:				
Base, NoFiltering	0.416	268	53.6%	17.9%
AND, NoFiltering	0.338	226	45.2%	33.3%
Tile N-Gram:				
Base, NoTiling	0.435	277	55.4%	14.2%
AND, NoTiling	0.397	251	50.2%	21.7%
Combinations:				
Base, NoTiling NoFiltering	0.319	233	46.6%	37.1%
AND, NoTiling NoFiltering	0.266	191	38.2%	47.5%

Table 1. Componential analysis of the AskMSR QA system.

N-Gram Filtering:
Unigrams, bigrams and trigrams are extracted from the (up to) 100 best-matching summaries for each rewrite, and scored according the weight of the query rewrite that retrieved them. The score assigned to an n-gram is a weighted sum across the summaries containing the n-grams, where the weights are those associated with the rewrite that retrieved a particular summary. The best-scoring n-grams are then filtered according to seven query types. For example the filter for the query *How many dogs pull a sled in the Iditarod?* prefers a number, so candidate n-grams like *dog race, run, Alaskan, dog racing, many mush* move down the list and *pool of 16 dogs* (which is a correct answer) moves up. Removing the filters decreases MRR by 17.9% relative to baseline (down to 0.416). Our simple n-gram filtering is the most important individual component of the system.

N-Gram Tiling:
Finally, n-grams are tiled to create longer answer strings. This is done in a simple greedy statistical manner from the top of the list down. Not doing this tiling decreases performance by 14.2% relative to baseline (down to 0.435). The advantages gained from tiling are two-fold. First, with tiling substrings do not take up several answer slots, so the three answer candidates: *San*, *Francisco*, and *San Francisco*, are conflated into the single answer candidate: *San Francisco*. In addition, longer answers can never be found with only trigrams, e.g., *light amplification by stimulted emission of radiation* can only be returned by tiling these shorter n-grams into a longer string.

Combinations of Components:
Not surprisingly, removing all of our major components except the n-gram accumulation (weighted sum of occurrences of unigrams, bigrams and trigrams) results in substantially worse performance than our full system, giving an MRR of 0.266, a decrease of 47.5%. The simplest entirely statistical system with no linguistic knowledge or processing employed, would use only AND queries, do no filtering, but do statistical tiling. This system uses redundancy only in summing n-gram counts across summaries. This system has MRR 0.338, which is a 33% drop from the best version of our system, with all components enabled. Note, however, that even with absolutely no linguistic proc-

essing, the performance attained is still very reasonable performance on an absolute scale, and in fact only one TREC-9 50-byte run achieved higher accuracy than this.

To summarize, we find that all of our processing components contribute to the overall accuracy of the question-answering system. The precise weights assigned to different query rewrites seems relatively unimportant, but the rewrites themselves do contribute considerably to overall accuracy. N-gram tiling turns out to be extremely effective, serving in a sense as a "poor man's named-entity recognizer". Because of the effectiveness of our tiling algorithm over large amounts of data, we do not need to use any named entity recognition components. The component that identifies what filters to apply over the harvested n-grams, along with the actual regular expression filters themselves, contributes the most to overall performance.

4 Component Problems

Above we described how components contributed to improving the performance of the system. In this section we look at what components errors are attributed to. In Table 2, we show the distribution of error causes, looking at those questions for which the system returned no correct answer in the top five hypotheses.

Problem	% of Errors
Units	23
Time	20
Assembly	16
Correct	14
Beyond Paradigm	12
Number Retrieval	5
Unknown Problem	5
Synonymy	2
Filters	2

Table 2. Error Attribution

The biggest error comes from not knowing what units are likely to be in an answer given a question (e.g. How fast can a Corvette go → xxx mph). Interestingly, 34% of our errors (Time and Correct) are not really errors, but are due to time problems or cases where the answer returned is truly correct but not present in the TREC-9 answer key. 16% of the failures come from the inability of our n-gram tiling algorithm to build up the full string necessary to provide a correct answer.

Number retrieval problems come from the fact that we cannot query the search engine for a number without specifying the number. For example, a good rewrite for the query *How many islands does Fiji have* would be « *Fiji has <NUM> islands »,* but we are unable to give this type of query to the search engine. Only 12% of the failures we classify as being truly outside of the system's current paradigm, rather than something that is either already correct or fixable with minor system enhancements.

5 Knowing When We Don't Know

Typically, when deploying a question answering system, there is some cost associated with returning incorrect answers to a user. Therefore, it is important that a QA system has some idea as to how likely an answer is to be correct, so it can choose not to answer rather than answer incorrectly. In the TREC QA track, there is no distinction made in scoring between returning a wrong answer to a question for which an answer exists and returning no answer. However, to deploy a real system, we need the capability of making a trade-off between precision and recall, allowing the system not to answer a subset of questions, in hopes of attaining high accuracy for the questions which it does answer.

Most question-answering systems use hand-tuned weights that are often combined in an ad-hoc fashion into a final score for an answer hypothesis (Harabagiu et al., 2000; Hovy et al., 2000; Prager et al., 2000; Soubbotin & Soubbotin, 2001; Brill et. al., 2001). Is it still possible to induce a useful precision-recall (ROC) curve when the system is not outputting meaningful probabilities for answers? We have explored this issue within the AskMSR question-answering system.

Ideally, we would like to be able to determine the likelihood of answering correctly solely from an analysis of the question. If we can determine we are unlikely to answer a question correctly, then we need not expend the time, cpu cycles and network traffic necessary to try to answer that question.

We built a decision tree to try to predict whether the system will answer correctly, based on a set of features extracted from the question string:

word unigrams and bigrams, sentence length (QLEN), the number of capitalized words in the sentence, the number of stop words in the sentence (NUMSTOP), the ratio of the number of nonstop words to stop words, and the length of longest word (LONGWORD). We use a decision tree because we also wanted to use this as a diagnostic tool to indicate what question types we need to put further developmental efforts into. The decision tree built from these features is shown in Figure 2. The first split of the tree asks if the word "How" appears in the question. Indeed, the system performs worst on "How" question types. We do best on short "Who" questions with a large number of stop words.

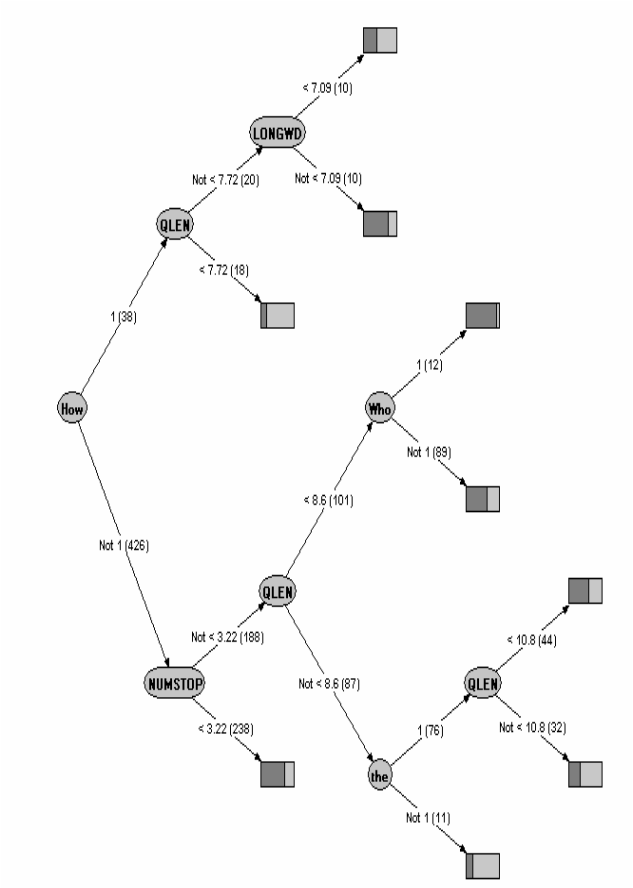

Figure 2. Learning When We Don't Know -- Using Only Features from Query

We can induce an ROC curve from this decision tree by sorting the leaf nodes from the highest probability of being correct to the lowest. Then we can gain precision at the expense of recall by not answering questions in the leaf nodes that have the highest probability of error. The result of doing this can be seen in Figures 3 and 4, the line labeled "Question Features". The decision tree was trained on Trec 9 data. Figure 3 shows the results when applied to the same training data, and Figure 4 shows the results when testing on Trec 10 data. As we can see, the decision tree overfits the training data and does not generalize sufficiently to give useful results on the Trec 10 (test) data.

Next, we explored how well answer correctness correlates with answer score in our system. As discussed above, the final score assigned to an answer candidate is a somewhat ad-hoc score based upon the number of retrieved passages the n-gram occurs in, the weight of the rewrite used to retrieve each passage, what filters apply to the n-gram, and the effects of merging n-grams in answer tiling. In Table 3, we show the correlation coefficient calculated between whether a correct answer appears in the top 5 answers output by the system and (a) the score of the system's first ranked answer and (b) the score of the first ranked answer minus the score of the second ranked answer. A correlation coefficient of 1 indicates strong positive association, whereas a correlation of –1 indicates strong negative association. We see that there is indeed a correlation between the scores output by the system and the answer accuracy, with the correlation being tighter when just considering the score of the first answer.

	Correlation Coefficient
Score #1	.363
Score #1 – Score #2	.270

Table 3 . Do answer scores correlate with correctness?

Because a number of answers returned by our system are correct but scored wrong according to the TREC answer key because of time mismatches, we also looked at the correlation, limiting ourselves to Trec 9 questions that were not time-sensitive. Using this subset of questions, the correlation coefficient between whether a correct answer appears in the system's top five answers, and the score of the #1 answer, increases from .363 to .401. In Figure 3 and 4, we show the ROC curve induced by deciding when not to answer a question based on the score of the first ranked answer (the

line labeled "score of #1 answer"). Note that the score of the top ranked answer is a significantly better predictor of accuracy than what we attain by considering features of the question string, and gives consistent results across two data sets.

Finally, we looked into whether other attributes were indicative of the likelihood of answer correctness. For every question, a set of snippets is gathered. Some of these snippets come from AND queries and others come from more refined exact string match rewrites. In Table 4, we show MRR as a function of the number of non-AND snippets retrieved. For instance, when all of the snippets come from AND queries, the resulting MRR was found to be only .238. For questions with 100 to 400 snippets retrieved from exact string match rewrites, the MRR was .628.

NumNon-AND Passages	NumQ	MRR
0	91	0.238
1 to 10	80	0.405
11 to 100	153	0.612
100 to 400	175	0.628

Table 4 . Accuracy vs. Number of Passages Retrieved From Non-AND Rewrites

We built a decision tree to predict whether a correct answer appears in the top 5 answers, based on all of the question-derived features described earlier, the score of the number one ranking answer, as well as a number of additional features describing the state of the system in processing a particular query. Some of these features include: the total number of matching passages retrieved, the number of non-AND matching passages retrieved, whether a filter applied, and the weight of the best rewrite rule for which matching passages were found. We show the resulting decision tree in Figure 5, and resulting ROC curve constructed from this decision tree, in Figure 3 and 4 (the line labeled "All Features"). In this case, the decision tree does give a useful ROC curve on the test data (Figure 4), but does not outperform the simple technique of using the ad hoc score of the best answer returned by the system. Still, the decision tree has proved to be a useful diagnostic in helping us understand the weaknesses of our system.

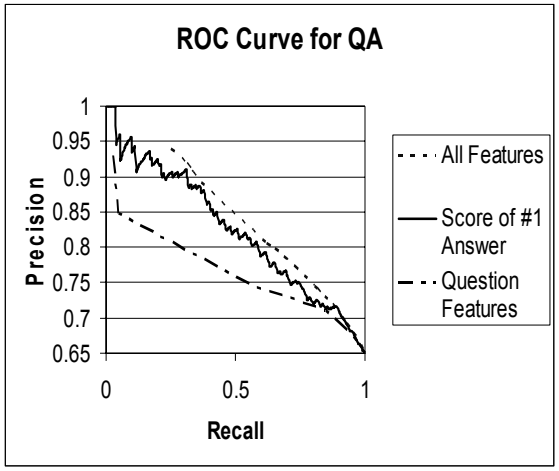

Figure 3. Three different precision/recall tradeoffs, trained on Trec 9 and tested on Trec 9.

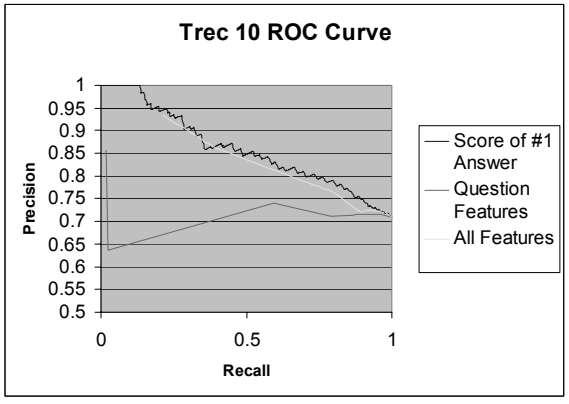

Figure 4. Three different precision/recall tradeoffs, trained on Trec 9 and tested on Trec 10.

6 Conclusions

We have presented a novel approach to question-answering and carefully analyzed the contributions of each major system component, as well as analyzing what factors account for the majority of errors made by the AskMSR question answering system. In addition, we have demonstrated an approach to learning when the system is likely to answer a question incorrectly, allowing us to reach any desired rate of accuracy by not answering some portion of questions. We are currently exploring whether these techniques can be extended beyond short answer QA to more complex cases of information access.

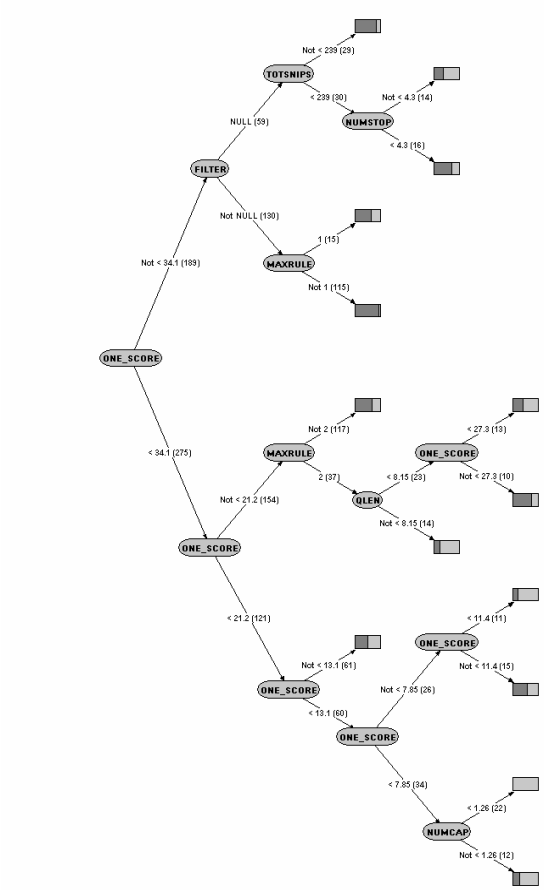

Figure 5. Learning When We Don't Know -- Using All Features

References

AAAI Spring Symposium Series Mining answers from text and knowledge bases (2002).

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

ACL-EACL Workshop on Open-domain question answering. (2002).

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

E. Brill, J. Lin, M. Banko, S. Dumais and A. Ng (2001). Data-intensive question answering. In *Proceedings of the Tenth Text Retrieval Conference (TREC 2001)*.

S. Buchholz (2001). 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)*.

J. Chen, A. R. Diekema, M. D. Taffet, N. McCracken, N. E. Ozgencil, O. Yilmazel, E. D. Liddy (2001). Question answering: CNLP at the TREC-10 question answering track. 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 (2001). Web reinforced question answering. To appear in *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*.

S. Harabagiu, D. Moldovan, M. Pasca, R. Mihalcea, M. Surdeanu, R. Bunescu, R. Girju, V. Rus and P. Morarescu (2000). FALCON: Boosting knowledge for question answering. In *Proceedings of the Ninth Text REtrieval Conference (TREC-9)*.

E. Hovy, L. Gerber, U. Hermjakob, M. Junk and C. Lin (2000). Question answering in Webclopedia. In *Proceedings of the Ninth Text REtrieval Conference (TREC-9)*.

E. Hovy, U. Hermjakob and C. Lin (2001). The use of external knowledge in factoid QA. 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*.

M. A. Pasca and S. M. Harabagiu (2001). High performance question/answering. In *Proceedings of SIGIR'2001*.

J. Prager, E. Brown, A. Coden and D. Radev (2000). Question answering by predictive annotation. In *Proceedings of SIGIR'2000*.

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*.

M. M. Soubbotin and S. M. Soubbotin (2001). 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. (2000). Proceedings of the Ninth Text REtrieval Conference (TREC-9).

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