Question-Answering by Predictive Annotation

John Prager, Eric Brown, Anni Coden
IBM T.J. Watson Research Center
Yorktown Heights, N.Y. 10598
jprager/ewb/anni@us.ibm.com

Dragomir Radev
University of Michigan
Ann Arbor, Michigan
radev@umich.edu

Abstract

We present a new technique for question answering called Predictive Annotation. Predictive Annotation identifies potential answers to questions in text, annotates them accordingly and indexes them. This technique, along with a complementary analysis of questions, passage-level ranking and answer selection, produces a system effective at answering natural-language fact-seeking questions posed against large document collections. Experimental results show the effects of different parameter settings and lead to a number of general observations about the question-answering problem.

1. Introduction

Question-answering is an area of Information Retrieval (IR) that is attracting increasingly more attention, as evidenced by new tracks in conferences such as AAAI[1] and TREC[5,14], and several Web sites. A Question-answering system searches a large text collection and finds a short phrase or sentence that precisely answers a user's question. To solve the Question-answering problem, we might first turn to traditional IR techniques, which have been applied successfully to large scale text search problems[6]. Unfortunately, traditional text search engines typically return lists of documents in response to a user's query, and therefore provide inappropriate solutions to this problem. Alternatively, the Natural Language Processing (NLP) and Information Extraction (IE) communities have developed techniques for extracting very precise answers from text. However, these communities use domain specific techniques applied to relatively small text collections.

It would appear that an approach combining the strengths of IR and NLP/IE might provide an appropriate way to answer questions using large bodies of text. The extent to which information extraction or deep parsing techniques must be applied to solve the Question-answering problem is an open question. We explore this question and present results showing that combining shallow NLP/IE techniques with a custom text search produces an effective Question-answering system. In particular, we describe a new text-processing technique called Predictive Annotation (PA) and analyze its effectiveness using the TREC8 benchmark. Our analysis reveals a number of characteristics about the Question-answering problem, leading to a variety of widely applicable system parameter settings.

This paper is in 5 sections. In Section 2 we describe the search system we implemented. In Section 3 we analyze the performance of the system. We also attempt to determine the limits of our approach. We discuss related work in Section 4 and in Section 5 we conclude and discuss future work.

2. The GuruQA System

Our approach is based on the following observations about fact-seeking questions:

- Questions can be classified by the kind of answer they are seeking.
- Answers are usually in the form of phrases.
- Answer phrases can be classified by the same scheme as the questions.
- Answers can be extracted from text using shallow parsing techniques.
- The context of the answer phrase that validates it as an answer to the question is usually a small fraction of the document it is embedded in.

We took our existing prototype search engine Guru[19] and modified it in three significant ways. We modified the query analysis to detect the question type and to modify the question accordingly. We modified the indexing process to perform shallow linguistic analysis of the text and to identify and annotate appropriate phrases with class labels. We modified the search engine to rank passages instead of documents, and to use a simple ranking formula. These steps combined to produce a system that performed well in the TREC8 QA track.

The complete GuruQA system is depicted in Figure 1. Two components, Query Analysis and Textract/Resporator, involve the generation of QA-Tokens, which we now describe.

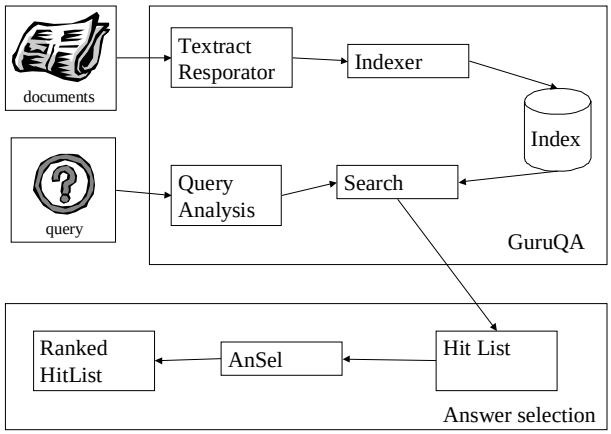

Figure 1. Block diagram of the GuruQA system.

QA-Tokens

We observed earlier that both questions and answer phrases could be classified. The question immediately arises of what is the most appropriate granularity of such a classification. The coarsest useful granularity would be according to the so-called "wh-words"; a much finer level would be to use, at least for noun phrases, the basic object categories of Rosch et al.[11]. We chose an intermediate level of about 20 categories which correspond fairly closely to the named-entity types of [13]: each such category is identified by a construct we call a QA-Token. The QA-Token serves both as a category label and a text-string used in the search process. Table 1 lists our QA-Tokens, along with associated question types and representative matching text phrases.

QA-Token	Question type	Example
PLACE$	Where	In the Rocky Mountains
COUNTRY$	Where/What country	United Kingdom
STATE$	Where/What state	Massachusetts
PERSON$	Who	Albert Einstein
ROLE$	Who	Doctor
NAMES$[1]	Who/What/Which /Where/Name the	Shakespeare Festival
ORG$	Who/What	The US Post Office
DURATION$	How long	For 5 centuries
AGE$	How old	30 years old
YEAR$	When/What year	1999
TIME$	When	In the afternoon
DATE$	When/What date	July 4th, 1776
VOLUME$	How big	3 gallons
AREA$	How big	4 square inches
LENGTH$	How big/long/high	5 miles
WEIGHT$	How big/heavy	25 tons
NUMBER$	How many	1,234.5
METHOD$	How	By rubbing
RATE$	How much	50 per cent
MONEY$	How much	$4 million

Table 1. List of QA-Tokens used in GuruQA

[1] The NAME$ token was used for proper names that Textract was unable to subclassify as Person, Place or Organization.

Textract

An important subsystem which we took great advantage of is the Textract text-processing system[2,17]. Textract provides basic document tokenization plus advanced text processing including lemmatization and annotation. The annotators include Nominator, which finds proper names, Terminator which finds technical terms (e.g. operating system), and Abbreviator which finds and resolves abbreviations. As well as generating document- and collection-level statistics, Textract also aggregates alternate forms of proper names and selects canonical forms.

The internal representation of text in Textract is the *word-list*, a linked list of tokens representing the individual words in the source, along with properties such as lemma form, potential part of speech (from a dictionary) and capitalization. When an annotator detects a sequence of tokens of interest (for example when Nominator detects that Bill Clinton is a name), it annotates the word-list with a new token representing the recognized entity. Bill Clinton will be annotated as a PERSON, with a pointer to the canonical form dictionary generated for the collection (which would likely have automatically registered Bill Clinton as a variant form of William Jefferson Clinton). Other functions of Textract include normalizing numbers, dates and monetary amounts.

Indexing

Prior to indexing, the collection is processed by Textract which is augmented by an additional annotator called Resporator. The purpose of Resporator is to identify the potential answer phrases in the text and annotate them with the corresponding QA-Token. Resporator consults a file of patterns for each QA-Token which it applies at each point in the word-list. Resporator runs after the previously described annotators, so quantities that the other annotators detect can be represented as quantities in the Resporator patterns. These patterns are written in a regular-expression-like language where tokens can be:

- Lemma forms of words (e.g. "be" which will match "am, is, are, was, were, be")
- Literal forms of words (e.g. "\am" which will match "am")
- Set-names (e.g. "_TIME" which will match "second, minute, hour, day, week …")
- Textract-names (e.g. ":CARDINAL" which will match any cardinal number recognized by Textract)

- Function names (e.g. "%ING" which calls a function to test if the word ends in 'ing')

Thus one of the patterns for the DURATION$ QA-Token might be

```
for :CARDINAL _TIME
```

The indexing process takes as input the word-list from Textract and indexes each word it encounters. When a Resporator annotation is encountered, it indexes the QA-Token as a text item, along with the underlying text as if the annotation didn't exist. Both the base form and the annotation are recorded at the same offset in the document.

Query Analysis

The query analysis stage takes the input question and converts it to a form suitable for the search engine. Here we match the question against one of about 400 templates. The purpose of the template matching is to introduce one or more QA-Tokens into the query and at the same time to remove unnecessary or unwanted words. Query words are converted to lemma form before matching, and lists of synonyms and categorical equivalents (e.g. all measures of time or adjectives of size) are consulted. So, for example, the query "How tall is the Matterhorn" gets translated into "LENGTH$ is the Matterhorn" (LENGTH$ represents a one-dimensional distance measure regardless of orientation). A conflict-resolution algorithm based on coverage is used when two or more templates match a query. In case no suitable template is found, weights are assigned to different parts of the question. In particular, the first noun phrase gets the same weight as a QA-Token. Stop words are removed subsequently.

There are two kinds of ambiguity that affect this process, semantic and granular. Semantic ambiguity occurs with questions like "How long …" which could be asking about time or distance (or conceivably, for works of literature, number of pages). Granular ambiguity occurs when there are QA-Tokens that represent nested classes (for example DATE$ and YEAR$ for "When" questions; PERSON$, ORG$, ROLE$, and NAME$ for "Who" questions). To this end we use the @SYN() operator which the search engine understands as enclosing a set of disjunctive entities (QA-Tokens or regular words) of which only one need match.

The query analysis also introduces operators to weight the query terms and to express the window size and type for matching. These are described later.

Matching and Ranking

The GuruQA search engine is a modified form of the Guru search engine, which returns ranked lists of documents in the traditional manner. GuruQA differs in that it ranks passages rather than entire documents. These passages are in the form of contiguous sequences of N sentences (possibly crossing paragraph boundaries), where N is passed to the search engine from Query Analysis as an argument in the @WIN() operator. Since the infrastructure of GuruQA is document-based, the search engine assigns each document the score of the best-matching passage of size N within it. This technique allowed us to most easily adapt an existing search engine to Question-Answering, but had the potential disadvantage of not being able to return multiple passages per document.

Ranking is a type of combination match, rather than traditional *tf*idf*[12]. We contend that the optimum matching passage only need have a single instance of each of the query terms in it, so that term frequency is irrelevant. The search engine will, for each passage, assign a score computed initially by summing the weights of each query term found in the passage. The weights for each query term are set in Query Analysis. We do not use *idf*, but rather a very coarse weighting scale: QA-Tokens have a score of 400, proper names a score of 200 and common words a score of 100. To act as a tie-breaker, the search engine computes the *density* of the match and from it assigns a delta score in the range 0-99. The density is inversely proportional to the distance (in words) between the first passage word that matched and the last; all matching words being consecutive would give a density of 99. The order of the words is irrelevant. The search engine holds off matching QA-Tokens till last, and ensures that they do not match words that had already matched. For example, the question: "What is the capital of Sri Lanka", is converted by Query Analysis to "PLACE$ capital Sri Lanka" (weight and window operators not shown); it is important for the PLACE$ QA-Token not to match "Sri Lanka" in text.

There is a serious issue concerning optimal window (passage) size. On the one hand, a small window containing a match for all of the query terms is, we felt, quite likely to express the same relation between the query terms as is intended by the question. On the other hand, a larger window size will deal with those cases where the question is answered (or more accurately, the correct answer is justified by matching qualifying phrases and clauses) over a span of several sentences. A larger window will be more likely to avoid problems with anaphora, but by the same token will be liable to noise from extraneous terms.

We assert that if a window W generates a certain score and a larger window W´ which contains W generates the same score, then the smaller passage will be preferred. This leads us to the concept of a *dynamic window* of size N. Using this, a document's score is the score of the best passage of any size from 1 to N sentences, with smaller passages beating equal-scoring containing passages.

Answer Selection

GuruQA returns top-matching passages; to complete the task we need to select the best answer phrases from these. For TREC8, we experimented with two different answer-selection algorithms, AnSel and WerLect, as reported in [20,21]. For this paper, we concentrate on the performance of the overall better-performing of these, namely AnSel. The TREC8 QA-tracks required submissions of the system's best 5 answers of length up to 50 bytes and up to 250 bytes. Using AnSel, we computed the 5 most likely named entities, and generated a 50 or 250-byte span of text surrounding the entity. For purposes of this paper, though, we judge the correctness of the named entity itself. On the rare occasions when there were no QA-

Tokens associated with the query, we discovered that WerLect performed better than the default AnSel behavior, but we don't explore this difference here.

AnSel scores all of the named entities labelled with QA-Tokens found within the top 10 passages returned by the search engine. AnSel is a linear classifier using a set of seven features with weights developed by a machine-learning algorithm employing logistic regression, with a little hand-tweaking. The features used were: sequential position of named entity amongst all of those returned, sequential position of named entity amongst all of those returned in the same passage, number of named entities in the passage, number of words in the entity not in the query, the position of the QA-Token for the entity in the @SYN() component of the query (or 1 if the QA-Token was not in a @SYN() group), the average distance in words between the beginning of the entity and the words in the query that also appear in the passage, and the passage relevance as computed by GuruQA.

3. Analysis

We used the TREC8 Question-answering testbed to evaluate the performance of our system and explore various characteristics of the Question-answering problem *per se*. Below we describe the testbed and explore the effects of window size, window type and question types on Question-anwering systems.

Testbed

The testbed for our system was the Question-Answering track introduced by NIST in 1999. Understanding that participants would be drawn from both the NLP and IR communities, subtracks for both 50-byte and 250-byte answers were established. 200 questions were sent out to the participants, and the top 5 answers in either or both categories were sent back. NIST used a team of judges to score the responses. A participant's score for a particular question was calculated as the reciprocal rank of the first correct answer noted. If all 5 answers were wrong, a score of 0 was assessed. A participant's overall score was calculated as the mean reciprocal rank (MRR) across all 200 questions (in practice 198 since 2 questions were later discarded).

Window Size

Based on limited experimentation with the relatively small amount of training data made available by NIST, we chose to operate GuruQA with a dynamic window size of 3. After the conference, a set of correct answers was made available. Using these judgments, we were able to run our system on the same questions for a variety of window sizes, fixed and dynamic. The overall MRR as a function of window size and type is shown in Table 2.

We see that for every size tested, dynamic windows perform better than fixed, as expected. Our testing with the 38 training questions led us to believe that a dynamic window size of 3 sentences was optimum, but the results with a collection over 5 times larger suggests that the optimum is 2, and in fact we see that 3 sentences is a local minimum! We believe, though, that there is no real significance in the differences in overall performance between window sizes 2-7. Figure 2 shows MRR by window size.

	1	2	3	4	5	7	10
Fixed	0.328	0.314	0.317	0.313	0.311	0.233	0.167
Dynamic		0.366	0.351	0.36	0.356	0.351	0.330

Table 2. Overall MRR as a function of window type and size

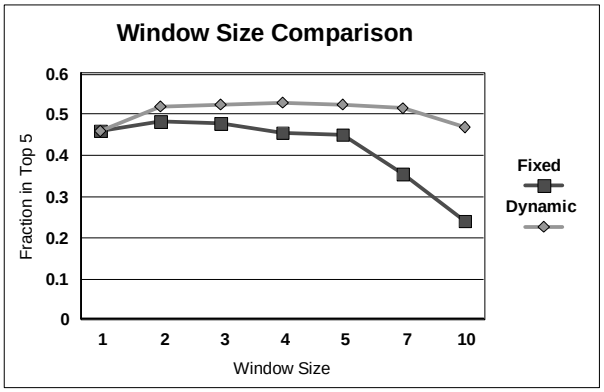

Figure 2. Performance (MRR) as a function of window size.

Examining the performance by question type, however, does reveal some differences. When the questions are grouped according to 9 broad classes, the performance of each group, along with the overall performance, is shown in Figures 3 and 4. The number of questions of each class is shown in Table 3.

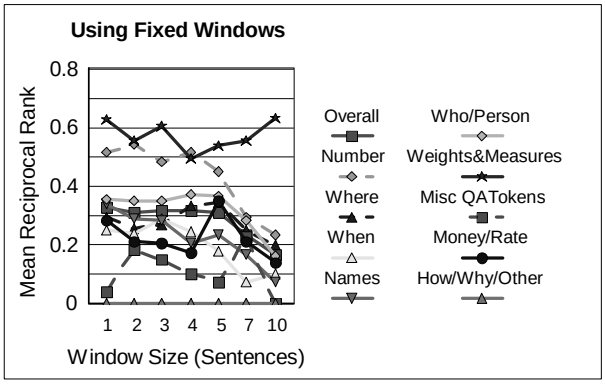

Figure 3. Performance as a function of fixed window size.

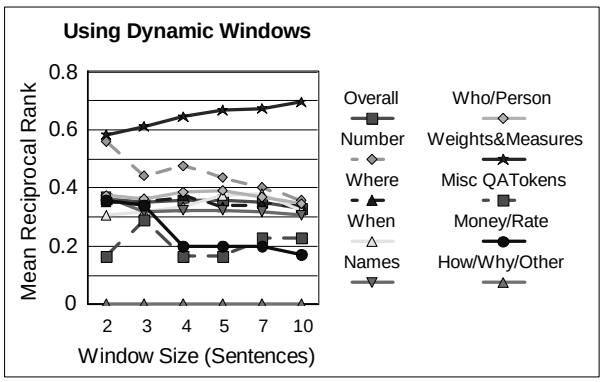

Figure 4. Performance as a function of dynamic window size.

Two general conclusions can be reached from this data, although it is conceded that some categories contain too few questions to make strong statements about. Firstly, varying the window size did not affect each question type equally. The most representative question types (Who/When/Where) performed about equally, but some of the other types showed divergent behavior, which might permit some advantage to be taken. Secondly, some kinds of questions are handled much better than others by our system. Weights and Measures (e.g. weight, length, volume, etc.) do far better than others, although Numbers do well with small windows. How and Why questions got zero scores. We will examine these issues in a little more detail in the next sections.

Class of Question	Count	Class of Question	Count
Who/Person	55	Number	21
Weights & Measures	9	Where	40
Misc. Other QA-Tokens	8	When	26
Money/Rate	7	Names	24
How/Why/Other	8	Overall	198

Table 3. Cardinality of Question Classes

Varying the Window Size

Since it appears that the performance for a given question class is a function of window size, we ask by how much the performance might improve if we vary the window size by question class. Using the results from the TREC8 questions we determined that the overall performance with variable dynamic window size increases to 0.394, compared with 0.366 for a uniform dynamic window size of 2. This improvement of 7.7% is only suggestive; we will know better when we test on a different set of questions.

Dealing with How and Why

How and Why questions are difficult for all question-answering systems; in particular, Predictive Annotation is by design primarily for fact-seeking questions rather than those seeking explanations. The difficulty is not in identifying the question type, but rather answer phrases in the text. We look briefly here at what we did for such cases, and what it might take to improve our treatment. For both of these question types, we looked for sufficient but not necessary conditions for detecting instances of such answers. Given our observations that in collections of news articles, events are often mentioned several if not many times, we considered this to be a reasonable first step.

One way in which methods and procedures are described in English is the word "by" followed by the present participle. Hence the pattern that Resporator uses, in fact the only pattern Resporator uses for How (the METHOD$ QA-Token) is

by %ING

This might have seemed to be perfectly adequate to answer the question: "How did Socrates die?". Unfortunately for us, the answers lay in sentences like:

> *"We also meet snake root, which is toxic, and poison hemlock, which for over two thousand years has been famous for curing Socrates of life."*

and

> *"His chapter on wifely nagging traces nagging back to the late Cretaceous period and notes that one of the all-time nags was Socrates' spouse, Xanthippe. Hemlock was a pleasure by comparison."*

We did not use a QA-Token for Why questions. Rather, on observing that reasons and explanations were usually introduced by words such as "because" and phrases such as "resulting in" or "as a result of" we simply replaced the word "why" in the query with:

@SYN(because, cause, result)

Now, explanations are also introduced by "in order to" or simply "to", followed by a verb; these phrases will generally either begin a sentence or will follow a main clause. To take advantage of these observations, sentence parsing or at least part-of-speech tagging is required. This will enable the answer to the question "Why did David Koresh ask the FBI for a word processor?", namely "to record his revelations" to be found.

Limitations of PA

We estimated the limitations of our approach by classifying the problems with questions whose reciprocal ranking was zero. We identify three groups:

1. Missing patterns (either in Question Analysis or Resporator text analysis)
2. Needed extensions or variations compatible with PA.
3. Extensions beyond the scope of PA.

The approach we took was to analyze the internal workings of GuruQA for each of the questions with rank 0 in order to discover the most likely reason for the failure. Where possible, the cause was verified by simulating the correct behavior of the identified faulty component and checking that the correct response was generated in the top 5 [2].

[2] The correct behavior was verified by simulation rather than fixing the component since in many cases a more general

For our purposes the *response* was the selected named entity, rather than a 50 or 250-byte passage surrounding it. This made our judgments both more severe than TREC8 but more accurate as a measure of correct system performance. This is because it often happens that incorrectly selected named entities are in the vicinity of the correct ones, which can cause the TREC8-style passage-based responses to be rated correct even though the system technically failed.

Our system with a dynamic window of size 3 and bugs fixed got 143 of 198 right (in the top 5). Each of the 55 remaining questions was put into one of the three above mentioned problem groups. Note that inappropriate window size is not listed as the cause of an error below. This is because this analysis was performed for windows of size 3, and for each question a window of size 3 was found to suffice for human answering. While we had found some few cases where the system performed better with a window different from 3, it was generally the case that some fix other than window size increase would have fixed the performance at 3 sentences. Our error types are tabulated in Table 4.

Category	Example	No.
1. Missing Patterns		
Question Analysis	Not having appropriate question template.	13
Resporator	Resporator missing a pattern to annotate text before indexing	18
2. More Complex Extensions/Fixes		
Synonyms/ Taxonomy	Question term is a synonym or hyponym of term in text	10
3. Beyond PA		
Anaphora	Proper nouns referred to by "the club", "the company", etc.	6
Text Understanding	Generally, deeper parsing & understanding required.	8

Table 4. Classification of Errors and their Counts

When patterns are detected to be missing from either the Question Analysis or Resporator processes, then these can be added without trouble. We did not consider this error condition to be a kind of bug, since these fixes will not guarantee that missing patterns won't be found when we next run our system on a collection of different questions. Since the performance of the system is a function of the completeness of the pattern set, this category must be considered a potential error source of our approach. However, it seems reasonable that over time, the size of this category will shrink considerably as full coverage develops.

Automatic synonym or hyper/hyponym expansion is not easy to get working effectively, as was shown by Voorhees[16]. However, since our very QA-Tokens are in effect hypernyms we are not convinced that some progress cannot be made without ruining precision.

The remaining sources of error – failures of anaphora resolution and text understanding - are ones that we do not think are readily removable with our Predictive Annotation technique alone. That is not to say that other technology working in conjunction with PA will not work, but simple extensions of our system in the manner of adding patterns will not address this problem. Quantitatively, 7% of the TREC8 questions cannot be handled by PA; a further 5% require an effective treatment of synonyms.

Performance by Question Type

Given the differences in performance by question types, we asked whether there was a simple explanation. We examined whether the occurrence frequencies of the QA-Tokens were correlated with performance, thinking that maybe the rarer tokens would be associated with higher precision. Such a discovery would help guide us in choosing the appropriate granularity for these types of QA-Tokens and others that we plan to add. The numbers of occurrences in our TREC8 index of the different QA-Tokens are shown in Table 5.

QA-Token	Occurrences	Documents
NAME$	7,271,982	509,899
NUMBER$	4,674,907	430,950
ORG$	3,366,153	466,423
PERSON$	3,344,594	398,304
ROLE$	2,506,693	430,938
PLACE$	2,066,163	400,266
DATE$	1,840,322	463,646
COUNTRY$	876,485	194,648
MONEY$	858,842	181,687
TIME$	659,343	272,453
RATE$	629,218	158,541
YEAR$	416,926	170,181
DURATION$	348,523	179,031
METHOD$	144,419	96,646
STATE$	133,404	47,830
LENGTH$	112,081	54,029
AGE$	62,745	35,756
WEIGHT$	32,156	11,731
AREA$	18,026	9,067
VOLUME$	3,909	2,380

Table 5. Term and Document Occurrences of QA-Tokens

solution, which would require careful planning and design, was preferred to a quick fix. Besides, a frozen version of the system was required for performing the window-size experiments described earlier.

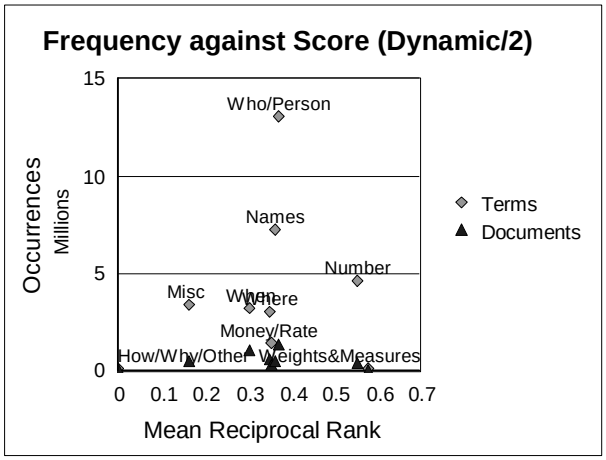

Figure 6. Scatter plot of terms (labelled) and documents against mean reciprocal rank for dynamic windows of size 2. Note that only the points for Terms are labelled.

When these were aggregated according to the question classes of Table 3 and plotted against the mean reciprocal rank with dynamic windows of size 2, the graph in Figure 6 was produced. No correlation is apparent. We think that this does not necessarily mean that no correlation exists, but rather there are too few questions examined in some of the classes to solidly identify a trend.

4.Related Work

The phrase "Question Answering" has been used in the literature to describe a number of related but distinct activities. The database community uses the phrase to describe techniques that support natural language query front ends to traditional database backends[4]. In the Artificial Intelligence community, Question Answering refers to a variety of techniques for answering questions using some sort of knowledge representation scheme, a theory for reasoning over this knowledge, and an inference engine that implements the theory[15].

Our Question Answering technique differs from both of these previous approaches in two significant ways. First, our underlying source of facts for answering questions is a document collection consisting of free text documents. Second, the free text documents are analyzed and searched in a fully automatic fashion with no human intervention required to extract, categorize, or organize the information contained within the documents.

The problem of answering questions using automatically processed text documents has been pursued by two different communities. The Message Understanding community has focussed on automatic fact extraction from free text documents[10]. The extracted facts are then represented in a structured form that supports reasoning over the facts. This approach typically requires significant human engineering to customize the fact extraction process for a particular domain, and hence does not provide the same level of domain independence as our question answering technique.

The Information Retrieval community has pursued question answering at a much more general level[12,18]. There the problem is cast more broadly as the problem of satisfying a user's information need. The typical approach is to return a set or a rank ordered list of documents that will satisfy the user's information need. This differs significantly from the goal of the Question Answering system presented here, which strives to identify the specific phrase or sentence that immediately answers the user's question.

The earliest work most similar to that presented here is the MURAX system described by Kupiec[8]. MURAX answers a question by first processing the question with a Boolean search engine, asserting possible answers based on the search results using part-of-speech tagging and shallow NLP techniques, and testing the assertions with additional queries against the search engine.

Other similar work has come from the inaugural Question Answering track at the TREC8 conference[5]. Our approach is similar to many others in the way questions are typed and mapped to named entities. The main differences lie in the degree of natural language processing and use of ontologies.

5. Conclusions

We presented in this paper Predictive Annotation and analyzed its performance in the TREC8 Question-Answering track. Predictive Annotation uses shallow pattern-matching of the textual material to augment it with a set of named entity identifiers called QA-Tokens which are indexed along with the text. Question analysis replaces question words with QA-Tokens, and bag-of-word matching of short passages takes place, using a very simple scoring mechanism. We project that our techniques can in principle handle of the order of 90% of the questions used in this track (as distinct from the 72% achieved in practice). We determined that short, dynamic windows in the range of 2-7 sentences were effective, which would lead to a recommendation to use a size of 2, due to the reduced computation entailed.

It might be instructive to list the approaches we did *not* use. We did not use deep NLP or any IE template-matching. We did not use part-of-speech tagging, nor any standard thesaurus or ontology such as WordNet[9]. We did not even use a variant of *tf*idf* scoring. We think these are all important and worthwhile techniques, and ones which we will probably use to some extent in the future, but we were primarily interested in how well PA works by itself. We conclude that PA is very effective at fact-oriented question answering, and only fails when responses require text understanding, anaphora resolution or automatic synonym or hypernym substitution, which are all hard problems for any IR system.

References.

[1] AAAI Fall Symposium on Question Answering, North Falmouth, MA, 1999.

[2] R. Byrd and Y. Ravin. "Identifying and Extracting Relations in Text", *Proceedings of NLDB 99*, Klagenfurt, Austria.

[3] V. Chaudhri and R. Fikes. Question answering systems: Papers from the *1999 AAAI fall symposium. Technical Report FS-99-02*, AAAI Press, 1999.

[4] F.J. Damerau. Problems and some solutions in customization of natural language database front ends. *ACM Trans. Inf. Syst.*, 3(2):165-184, Apr. 1985.

[5] D. Harman and E. Voorhees, editors. *The Eighth Text REtrieval Conference (TREC-8)*, Gaithersburg, MD, 2000. National Institute of Standards and Technology Special Publication.

[6] D. Hawking, N. Craswell, and P. Thistlewaite. Overview of TREC-7 very large collection track. In D. K. Harman and E. M. Voorhees, editors, *The Seventh Text REtrieval Conference (TREC-7*), pages 91-104, Gaithersburg, MD, 1999. National Institute of Standards and Technology Special Publication 500-242.

[7] V. A. Kulyukin, K.J. Hammond, and R.D. Burke. "Answering Questions for an Organization Online", *Proceedings of AAAI'98*.

[8] J. Kupiec. "Murax: A Robust Linguistic Approach for Question Answering Using an On-line Encyclopaedia", *Proceedings of SIGIR'93*.

[9] G. Miller. "WordNet: A Lexical Database for English", *Communications of the ACM* 38(11) pp 39-41, 1995

[10] Proc. of the Sixth Message Understanding Conference (MUC-6), November 1995, San Francisco: Morgan Kaufmann.

[11] E. Rosch et al. "Basic Objects in Natural Categories", *Cognitive Psychology* 8, 382-439, 1976.

[12] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[13] R. Srihari and W. Li. "Question Answering Supported by Information Extraction", *Proceedings of TREC8*, Gaithersburg, Md., 1999.

[14] TREC Q&A Evaluation official Web site: http://www.research.att.com/~singhal/qa-track.html

[15] E. Turban and J. Aronson. *Decision Support Systems and Intelligent Systems*. Prentice Hall, 1998.

[16] E. Voorhees. "Query Expansion using Lexical-Semantic Relations", *Proceedings of SIGIR'94*, 61-69, 1994.

[17] N. Wacholder, Y. Ravin and M. Choi. "Disambiguation of Proper Names in Text", *Proceedings of ANLP'97*. Washington, DC, April 1997.

[18] I.H. Witten, A. Moffat, and T.C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, New York, 1994.

[19] E.W. Brown and H.A. Chong. The Guru System in TREC-6. *Proceedings of TREC6*, Gaithersburg, MD, 1998.

[20] J.M. Prager, D. Radev, E.W. Brown and A.R. Coden. "The Use of Predictive Annotation for Question-Answering in TREC8", *Proceedings of TREC8*, Gaithersburg, MD., 2000.

[21] D. Radev, J.M.,Prager and V. Samn. "Ranking Suspected Answers to Natural Language Questions using Predictive Annotation", to be published in *Proceedings of ANLP'00*, Seattle, WA, 2000.