Chapter #

How to Select an Answer String?

Abdessamad Echihabi, Ulf Hermjakob, Eduard Hovy, Daniel Marcu,
Eric Melz, Deepak Ravichandran

Information Sciences Institute, University of Southern California, CA

Key words: question answering, answer selection

Abstract: Given a question Q and a sentence/paragraph SP that is likely to contain the answer to Q, an answer selection module is supposed to select the "exact" answer sub-string A ⊂ SP. We study three distinct approaches to solving this problem: one approach uses algorithms that rely on rich knowledge bases and sophisticated syntactic/semantic processing; one approach uses patterns that are learned in an unsupervised manner from the web, using computational biology-inspired alignment algorithms; and one approach uses statistical noisy-channel algorithms similar to those used in machine translation. We assess the strengths and weaknesses of these three approaches and show how they can be combined using a maximum entropy-based framework.

1. INTRODUCTION

The recent activity in research on automated question answering concentrating on factoids—brief answers—has highlighted the two basic stages of the process—information retrieval, to obtain a list of candidate

passages likely to contain the answer, and answer selection, to identify and pinpoint the exact answer among and within the candidates. In this paper we focus on the second stage. We situate this work in the context of the TREC question answering evaluation competitions, organized annually since 1999 by NIST (Voorhees, 1999; 2000; 2001; 2002), and use both the TREC question and answer collections, the TREC text corpus of some 1 million newspaper and similar articles, and the TREC scoring method of Mean Reciprocal Rank (MRR), in order to make our results comparable to other research.

What constitutes a correct, exact answer to a natural language question is easiest described by means of examples. The TREC guidelines (http://trec.nist.gov/pubs.html) specify, for instance, that given the question *What river in the US is known as the Big Muddy?,* strings such as "Mississippi", "the Mississippi", "the Mississippi River", and "Mississippi River" should be judged as exact answers, while strings such as "2,348 miles; Mississippi", "Mississip", and "known as the Big Muddy, the Mississippi is the longest river in the US" should be considered inexact.

Automatically finding in a document collection the correct, exact factoid answer to a natural language question is by no means a trivial problem, since it involves several processes that are each fairly sophisticated, including the ability to understand the question, derive the expected answer type, generate information retrieval queries to select documents, paragraphs, and sentences that may contain the answer, and pinpoint in these paragraphs and sentences the correct, exact, answer sub-string. The best question answering systems built to date are complex artefacts that use a large number of components such as syntactic/semantic parsers, named-entity taggers, and information retrieval engines. Unfortunately, such complexity masks the contribution of each module, making it difficult to assess why the system fails to find accurate answers. - For this reason, we are constantly designing experiments that will enable us to understand better the strengths and weaknesses of the components we are using and to prioritize our work, in order to increase the overall performance of our system. One such experiment, for example, showed clearly that answer selection is far from being a solved problem. To diagnose the impact of the answer selection component, we did the following:

- We used the 413 TREC-2002 questions that were paired by human assessors with correct, exact answers in the TREC collection.
- For each question, we selected all sentences that were judged as containing a correct, exact answer to it.

- We presented the questions and just these answer sentences to the best answer selection module we had available at that time; in other words, we created perfect experimental conditions, consistent to those that one would achieve if one had perfect document, paragraph, and sentence retrieval components.

To our surprise, we found that our answer selection module was capable of selecting the correct, exact answer in only 68.2% of the cases. That is, even when we gave our system *only* sentences that contained correct, exact answers, it failed to identify more than 30% of them! Two other answer selection modules, which we were developing at the same time, produced even worse results: 63.4% and 56.7% correct. Somewhat more encouraging, we determined that an oracle that could select the best answer produced by any of the three answer selection modules would have produced 78.9% correct, exact answers. Still, that left over 20% correct answers not being recognized.

The results of this experiment suggested two clear ways for improving the performance of our overall QA system:

- Increase the performance of any (or all) answer selection module(s).
- Develop methods for combining the strengths of the different modules.

In this chapter, we show that the Maximum Entropy (ME) framework can be used to address both of these problems. By using a relatively small corpus of question-answer pairs annotated by humans with correctness judgments as training data, and by tuning a relatively small number of log-linear features on the training corpus, we show that we can substantially increase the performance of each of our individual answer selection modules. Pooling the answers produced by all systems leads to an additional increase in performance. This suggests that our answer selection modules have complementary strengths and that the ME framework enables one to learn and exploit well the individualities of each system.

Ultimately, real users do not care about the ability of an answer selection module to find exact, correct answers in hand-picked sentences. Because of this, to substantiate the claims made in this chapter, we carried out all our evaluations in the context of an end-to-end QA system, TextMap, in which we varied only the answer selection component(s). The TextMap system implements the following pipeline (see (Hermjakob et al., 2002) for details):

- A question analyser identifies the expected answer type for the question given as input (see Section 2.1 for details).

- A query generator produces Web- and TREC-specific queries. The query generator exploits a database of paraphrases (see Section 2.3).
- Web queries are submitted to Google and TREC queries are submitted to the IR engine Inquery (Callan et al., 1995), to retrieve respectively 100 Web and 100 TREC documents.
- A sentence retrieval module selects 100 sentences each from the retrieved Web and TREC documents that are most likely to contain a correct answer.
- Each of the answer selection modules described in this paper pinpoints the correct answers and in the resulting 200 sentences and assigns them a score.
- The highest ranked answer is presented to the user.

For the contrastive analysis of the answer selection modules we present in this chapter, we chose to use in all of our experiments the 413 factoid questions made available by NIST as part of the TREC-2003 QA evaluation. In all our experiments, we run our end-to-end QA system against documents available on either the Web or the TREC collection. We pinpoint exact answers in Web- or TREC-retrieved sentences using different answer selection or combinations of answer selection modules.

To measure the performance of our answer selection modules in the context of the end-to-end QA system, we created by hand an exhaustive set of correct and exact answer patterns. If the answer returned by a system matched perfectly one of the answer patterns for the corresponding question, the answer was considered correct and exact. If the answer did not match the answer pattern, it was considered incorrect. Naturally, this evaluation is not 100% bullet proof. One can still have correct, exact answers that are not covered by the patterns we created; or one can return answers that are correct and exact but unsupported. We took, however, great care in creating the answer patterns. Qualitative evaluations of the correctness of the results reported in our experiments suggest that our methodology is highly reliable. Most importantly though, even if the evaluation results are off by 1 or 2% in absolute terms due to incomplete coverage of the patterns, the methodology is perfectly sound for measuring the relative performance of the different systems because all systems are evaluated against a common set of answer patterns.

In this chapter, we present three different approaches to answer selection. One approach uses algorithms that rely on rich knowledge bases and sophisticated syntactic/semantic processing (Section 2); one approach uses

patterns that are learned in an unsupervised manner from the web, using computational biology-inspired alignment algorithms (Section 3); and one approach uses statistical, noisy-channel algorithms similar to those used in machine translation (Section 4). We assess the performance of each individual system in terms of

- number of correct, exact answers ranked in the top position;
- number of correct, exact answers ranked in the top 5 positions;
- MRR score[1] based on the top five answers.

We show that maximum entropy working with a relative small number of features has a significant impact on the performance of each system (Section 5). We also show that the same ME-based approach can be used to combine the outputs of the three systems. When we do so, the performance of the end-to-end QA system increases further (Section 5).

2. KNOWLEDGE-BASED ANSWER SELECTION

This section describes a strongly knowledge-based approach to question answering. As described in the following subsections, this approach relies on several types of knowledge. Among them, answer typing ("Qtargets"), semantic relationship matching, paraphrasing, and several additional heuristics all heavily rely on parsing, of both the question and all answer sentence candidates.

We use the CONTEX parser (Hermjakob, 1997; 2001), a decision tree based deterministic parser, which has been enhanced for question answering by an additional treebank of 1,200 questions, named entity tagging that among other components uses BBN's IdentiFinder (Bikel et al., 1999), and a semantically motivated parse tree structure that facilitates matching for paraphrasing and of question/answer pairs.

2.1 Qtargets

After parsing a question, TextMap determines its answer type, or "Qtarget", such as *PROPER-PERSON*, *PHONE-NUMBER*, or *NP*. We have built a typology of currently 185 different types, organized into several classes (Abstract, Semantic, Relational, Syntactic, etc.). An older version of the

[1] TREC's Mean Reciprocal Rank assigns a score of 1 if the correct answer is in first place (of five guesses), 1/2 if it is in second place, 1/3 if in third place, etc.

typology can be found at http://www.isi.edu/natural-language/projects/webclopedia/Taxonomy/taxonomy_toplevel.html.

As the following example shows, Qtargets can significantly narrow down the search space (neither "exactly" nor "2.8% taller than K2" conform to a DISTANCE-QUANTITY answer type):

Question: How tall is Mt. Everest?

Qtarget: DISTANCE-QUANTITY

Answer candidates:

- Jack knows exactly how tall Mt. Everest is.
- Jack climbed the 29,028-foot Mt. Everest in 1984 and the 7,130-foot Mt. Kosciusko in Australia in 1985.
- Mt. Everest is 2.8% taller than K2.

2.2 Semantic Relations

As words and answer types are often not enough to identify the proper answer, the knowledge-base answer selection modules also boosts the scores of answer candidates whose constituents have the same semantic relationships to one another as those in the question:

Question: Who killed Lee Harvey Oswald?

Text: Jack Ruby, who killed John F. Kennedy assassin Lee Harvey Oswald

While "John F. Kennedy" is textually closer to the question terms "killed" and "Lee Harvey Oswald", our QA system will prefer "Jack Ruby", because its logical subject relation to the verb matches that of the interrogative in the question. Semantic relations hold over all roles and phrase types, and are independent of word order.

2.3 Paraphrases

Sentences with a good answer often don't match the wording of the question; sometimes simply matching surface words can result in an incorrect answer:

Question: Who is the leader of France?

Candidate answers:

- Henri Hadjenberg, who is the leader of France's Jewish community, endorsed confronting the specter of the Vichy past.

  (100% word overlap, but sentence does not contain answer.)

- Bush later met with French President Jacques Chirac.

  (0% word overlap, but sentence does contain the correct answer.)

Neither word reordering nor simple word synonym expansion will help us to identify Jacques Chirac as the correct answer.

To bridge the gap between question and answer sentence wordings, TextMap uses paraphrasing. For any given question, TextMap generates a set of high-precision meaning-preserving reformulations to increase the likelihood of finding correct answers in texts:

Question: How did Mahatma Gandhi die?

Reformulation patterns:

- Mahatma Gandhi died <how>?
- Mahatma Gandhi died of <what>?
- Mahatma Gandhi lost his life in <what>?
- Mahatma Gandhi was assassinated?
- Mahatma Gandhi committed suicide?
- … plus 40 other reformulations …

The fourth reformulation will easily match "Mahatma Gandhi was assassinated by a young Hindu extremist," preferring it over alternatives such as "Mahatma Gandhi died in 1948."

Paraphrases can span a wide range, from simple syntactic reformulations (When did the Titanic sink? => The Titanic sank when?) to rudimentary forms of inference (Where is Thimphu? => Thimphu is the capital of <which place>?); for example:

Question: How deep is Crater Lake?

Reformulation patterns:

- Crater Lake is <what distance> deep?
- depth of Crater Lake is <what distance>?
- Crater Lake has a depth of <what distance>?
- <what distance> deep Crater Lake?

- and more

Question: Who invented the cotton gin?

Reformulation patterns:

- <who> was the inventor of the cotton gin?
- <who>'s invention of the cotton gin?
- <who> was the father of the cotton gin?
- <who> received a patent for the cotton gin?

2.3.1 Paraphrase Patterns: A Resource

Rather than creating and storing thousands of paraphrases, we acquire paraphrase patterns, which are used at run-time to generate instantiated patterns against which candidate answers are matched. Paraphrase patterns are acquired either by manual entry or by automated learning (see Section 3) and subsequent manual refinement and generalization. The paraphrase collection is pre-parsed, and then, at run-time, pattern matching of questions and paraphrases is performed at the parse tree level.

TextMap paraphrase patterns are expressed in an extended natural language format for high user friendliness:

:anchor-pattern "SOMETHING_1 costs MONETARY_QUANTITY_2."

:is-equivalent-to "the price of SOMETHING_1 is MONETARY_QUANTITY_2."

:is-equivalent-to "SOMETHING_1 is on sale for MONETARY_QUANTITY_2."

:can-be-inferred-from "to buy SOMETHING_1 for MONETARY_QUANTITY_2."

:anchor-pattern "SOMEBODY_1 sells SOMETHING_3 to SOMEBODY_2."

:is-equivalent-to "SOMEBODY_2 buys SOMETHING_3 from SOMEBODY_1."

Expressing phrasal synonyms in extended natural language makes them easy to write, or, when they are automatically generated, easy to check and filter. The relatively generic declarative format also facilitates reuse in other applications and systems. The expressiveness and focus of the patterns is greatly enhanced when variables carry syntactic or semantic restrictions that can transcend parts of speech. Compared to automatically generated patterns such as (Ravichandran and Hovy, 2002) and (Lin and Pantel, 2001), there are also no limits on the number of variables per reformulation and, since all patterns are checked by hand, only very few misreformulations.

The reformulation collection currently contains 550 assertions grouped into about 105 equivalence blocks.

At run-time, the number of reformulations produced by our current system varies from one reformulation (which might just rephrase a question into a declarative form) to more than 40, with an average of currently 5.03 reformulations per question for the TREC-2003 questions.

2.3.2 Advanced Forms of Reformulation

As seen in earlier examples, the paraphrase paradigm can implement a form of inference. Other advanced forms of reformulation in our system are reformulation chains, answer generation, and cross-part-of-speech placeholders. Based on

:anchor-pattern "SOMEBODY_1 is a student at COLLEGE_2."

:answers "Where does SOMEBODY_1 go to college?" :answer COLLEGE_2

:anchor-pattern "SOMEBODY_1 was a student at COLLEGE_2."

:can-be-inferred-from "SOMEBODY_1 dropped out of COLLEGE_2."

:anchor-pattern "SOMEBODY_1 dropped out of COLLEGE_2."

:is-equivalent-to "SOMEBODY_1 is a COLLEGE_2 dropout."

TextMap can produce the following reformulation chain:

Text corpus: Bill Gates is a Harvard dropout.

Original question: Where did Bill Gates go to college?

Reformulations:

- Bill Gates was a student at <which college>
- Bill Gates dropped out of <which college>
- Bill Gates is a <which college> dropout.

Answer: Harvard

Allowing placeholders to cross syntactic categories makes reformulations even more powerful. To support this type of reformulation, we draw on a number of cross-part-of-speech lists, which include entries such as [France/French] and [invent/invention/inventor]:

:anchor-pattern "NATIONALITY_1 OCCUPATION_2 PERSON_3"

:is-equivalent-to "PERSON_3 is the OCCUPATION_2 of COUNTRY_1"

which enables:

Text corpus: ... French President Jacques Chirac ...

Question: Who is the president of France?

Reformulation: French president <who>

Answer: Jacques Chirac

Paraphrases not only improve answer pinpointing, but can also support document retrieval and passage selection by proving alternate and/or multi-word search expressions, and increase confidence in many answers.

2.4 Additional Heuristics

Answer selection is further guided by several additional heuristics that penalize answers for a variety of reasons, including:

- Qtarget match factor: Q: How long did the Manson trial last? Semantic mismatch: 20 miles
- Vagueness penalty: Q: Where is Luxor? Too vague: on the other side
- Negation penalty: Q: Who invented the electric guitar? Negation: Fender did **not** invent the electric guitar.
- Bad mod penalty: Q: What is the largest city in Iraq? Relevant modifier: Basra is the **second** largest city in Iraq.
- Novelty factor: Q: Where in Austin is Barton Springs? Nothing novel: in Austin
- Reporting penalty: Q: Who won the war? Reported: Saeed al-Sahhaf **claimed** that Iraq had crushed the invading American forces
- Surface distance factor: favors answers near question terms
- Structural distance factor: favors answers near question terms in parse tree constituent distance
- Bad gender penalty: Q: Which actress played …? Bad gender: John Wayne

2.5 External Knowledge

The knowledge-based answer selection module uses a limited amount of external knowledge to enhance performance.

- For some questions, WordNet glosses provide an answer, either directly, such as for definition questions, or indirectly; e.g., "What is the capital of Kentucky?" can be answered using the WordNet gloss for Frankfort,

"the capital of Kentucky; located in northern Kentucky", from which the relevant fact has been extracted and stored in a database.

- Internal quantity and calendar conversion routines can answer questions such as "How much is 86F in Celsius?"
- Abbreviation routines score how well an abbreviation conforms with its expansion, and for example strongly prefer NAFTA as an abbreviation for "North American Free Trade Agreement" rather than as an abbreviation for "rice market".

2.6 Evaluation

When evaluated against the answer patterns we created for the 413 factoid questions in the TREC-2003 collection, the knowledge-based answer selection module described in this section produced 35.83% correct, exact answers. There were 57.38% correct, exact answers in the top 5 candidates returned by a system, with a corresponding MRR score of 43.88%. More details are provided in Section 5.

3. PATTERN-BASED ANSWER SELECTION

At the TREC 2001 conference, several systems emphasized the value of matching surface-oriented patterns, even without any reformulation, to pinpoint answers. The top-scoring Insight system from Moscow (Soubbotin and Soubbotin, 2001) used some hundreds of surface-level patterns to identify answer strings without (apparently) applying Qtargets or similar reasoning. Several other systems also defined word-level patterns indicating specific Qtargets; e.g., (Oh et al., 2001). The Microsoft system (Brill et al., 2001) extended the idea of a pattern to its limit, by reformulating the input question as a declarative sentence and then retrieving the sentence verbatim, with its answer as a completion, from the web, using normal search engines. For example, "Who was Chester F. Carlson?" was transformed to, among others, "Chester was F. Carlson", "Chester F. was Carlson", and "Chester F. Carlson was", and submitted as web queries. Although this approach yielded many wrong answers (including "Chester F. Carlson was born February 8, 1906, in Seattle"), the sheer number of correct answers returned often won the day.

Our estimate is that a large enough collection of word-level patterns, used even without reformulation, can provide at least 25% MRR score, although some systems claimed considerably higher results; see (Soubbotin and Soubbotin, 2001).

3.1 Automated Learning of Patterns

The principal obstacle to using the surface pattern technique is acquiring patterns in large enough variety and number. For any given Qtarget, one can develop some patterns by hand, but there is no guarantee that one thinks of all of them, and one have no idea how accurate or even useful each pattern is.

We therefore developed an automated procedure to learn such patterns from the web (Ravichandran and Hovy, 2002). Using a regular search engine, we collected all the patterns associated with many frequently occurring Qtargets (some Qtargets, such as *Planets* and *Oceans*, are known closed sets that require no patterns). Some of the more precise patterns, associated with their Qtarget in the QA Typology, can be found at http://www.isi.edu/natural-language/projects/webclopedia/Taxonomy/taxonomy_toplevel.html.

In addition to using the learned patterns as starting points to define reformulation pattern sets (Section 2), we used them to construct an independent answer selection module. The purpose of this work was to empirically determine the limits of a QA system whose pinpointing knowledge is derived almost fully automatically.

The pattern learning procedure can be phrased as follows. Given a Qtarget (a relation such as YEAR-OF-BIRTH), instantiated by a specific QA pair such as (NAME_OF_PERSON, BIRTHYEAR), extract from the web all the different lexicalized patterns (TEMPLATEs) that contain this QA pair, and also determine the precision of each pattern. The procedure contains two principal steps:

1. Extracting the patterns
2. Calculating the precision of each pattern

3.1.1 Algorithm 1: Extracting patterns

We wish to learn the surface-level patterns that express a given QA relation such as YEAR-OF-BIRTH.

1. An instance of the relation for which the pattern is to be extracted is passed to a search engine as a QA pair. For example, to learn the patterns for the pair (NAME_OF_PERSON BIRTHYEAR), we submit a pair of anchor terms such as “Gandhi 1869” as a query to Altavista.
2. The top 1000 documents returned by the search engine are retrieved.

3. These documents are broken into sentences by a simple sentence breaker.
4. Only sentences that contain both the Question and the Answer terms are retained. (BBN's named entity tagger IdentiFinder (Bikel et al., 1999) was used to remove variations of names or a dates.)
5. Each of these sentences is converted into a Suffix Tree using an algorithm from Computational Biology (Gusfield, 1997), to collect counts on all phrases and sub-phrases present in the document.
6. The phrases obtained from the Suffix Tree are filtered so that only those containing both the Question and Answer terms are retained. This yields the set of patterns for the given QA pair.

3.1.2 Algorithm 2: Calculating the precision of each pattern

1. The Question term alone (without its Answer term) is given as query to Altavista.
2. As before, the top 1000 documents returned by the search engine for this query are retrieved.
3. Again, the documents are broken into sentences.
4. Only those sentences that contain the Question terms are saved. (Again, IdentiFinder is used to standardize names and dates.)
5. For each pattern obtained in step 6 of Algorithm 1, a pattern-matching check is done against each sentence obtained from step 4 here, and only the sentences containing the Answer are retained. This data is used to calculate the precision of each pattern according to the formula

$$\text{Precision} = \frac{\#\text{ patterns matching the Answer (step 5 in Alg 2)}}{\text{total \# patterns (step 4 in Alg 1)}}$$

6. Furthermore, only those patterns are retained for which sufficient examples are obtained in step 5 of this algorithm.

To increase the size of the data, we apply the algorithms with several different anchor terms of the same Qtarget. For example, in Algorithm 1 for YEAR-OF-BIRTH we used Mozart, Gauss, Gandhi, Nelson Mandela, Michelangelo, Christopher Columbus, and Sean Connery, each with birth year. We then applied Algorithm 2 with just these names, counting the yields of the patterns using only the exact birth years (no additional words or reformulations, which would increase the yield score).

The results were quite good in some cases. For the rather straightforward YEAR-OF-BIRTH, some patterns are:

Prec.	#Correct	#Found	Pattern
1.00	122	122	<NAME> (<BD>- <DD>
1.00	15	15	<NAME> (<BD> - <DD>) ,
1.00	13	13	, <NAME> (<BD> - <DD>)
0.9166	11	12	<NAME> was born on <BD> in
0.9090	10	11	<NAME> : <BD> - <TIME>
0.6944	25	36	<NAME> was born on <BD>

Note the overlaps among patterns. By not compressing them further we can record different precision levels.

Due to the many forms of expression possible, the Qtarget DEFINITION posed greater problems. For example, the anchor term *disease* paired with *jaundice, measles, cancer,* and *tuberculosis* (but not also paired with *illness, ailment* etc., which would have increased the counts), yields:

Prec.	#Correct	#Found	Pattern
1	46	46	heart <TERM>, <NAME>
1	35	35	<NAME> & tropical <TERM> weekly
1	30	30	venereal <TERM>, <NAME>
1	26	26	<NAME>, a <TERM> that
1	24	24	lyme <TERM>, <NAME>
1	22	22	, heart <TERM>, <NAME>
1	21	21	's <TERM>, <NAME>
0.9565	22	23	lyme <TERM> <NAME>
0.9	9	10	s <TERM>, <NAME> and
0.8815	67	76	<NAME> , a <TERM>
0.8666	13	15	<TERM> , especially <NAME>

The anchor term *metal*, paired with *gold*, *silver*, *platinum*, and *bronze*, yields:

1.00	13	13	of <NAME> <TERM> .
1.00	12	12	the <TERM> <NAME> .
0.90	9	10	<TERM> : <NAME>.
0.875	14	16	the <NAME> <TERM> .
0.8181	27	33	<TERM> (<NAME>)
0.80	8	10	s <TERM> (<NAME>
0.7575	25	33	<TERM> , <NAME> , a
0.75	177	236	the <NAME> <TERM>
0.75	12	16	<TERM> : <NAME> ,
0.7427	179	241	<NAME> <TERM> ,
0.7391	17	23	<TERM> (<NAME> ,

0.70 7 10 <TERM> - <NAME> ,

The similar patterns for Disease and Metal definitions indicate that one should not create specialized Qtargets *Definition-Disease* and *Definition-Metal*.

3.2 Integrating Patterns into a QA System

We have learned patterns for numerous Qtargets. However, for questions about a Qtarget without patterns, the system would simply fail. There are two possible approaches: either develop a method to learn patterns dynamically, on the fly, for immediate use, or integrate the patterns with other answer pinpointing methods. The former approach, developing automated ways to produce the seed anchor terms to learn new patterns, is under investigation. Here we discuss the second, in which we use Maximum Entropy to integrate answers produced by the patterns (if any) with answers produced by a set of other features. In so doing we create a standalone answer selection module. We follow Ittycheriah (2002), who was the first to use a completely trainable statistical (Maximum Entropy) QA system.

3.2.1 Model

Assuming we have several methods for producing answer candidates (including, sometimes, patterns), we wish to select the best from all candidates in some integrated way. We model the problem of answer selection as a re-ranking problem (Ravichandran et al., 2003):

$$P(a \mid \{a_1 a_2 ... a_A\}, q) = \frac{\exp[\sum_{m=1}^{M} \lambda_m f_m(a, \{a_1 a_2 ... a_A\}, q)]}{\sum_{a'} \exp[\sum_{m=1}^{M} \lambda_m f_m(a', \{a_1 a_2 ... a_A\}, q)]}$$

where

a is the answer under consideration, one of the candidates $a \in \{a_1 a_2 ... a_A\}$

q is the question

$f_m(a, \{a_1 a_2 ... a_A\}, q)], m = 1,...., M$ are the M feature functions providing answers

$\{a_1 a_2 ... a_A\}$ is the answer candidate set considered for the question.

and the decision rule for the re-ranker is given by:

$$\hat{a} = \arg\max_{a}[P(a \mid \{a_1\, a_2 \ldots a_A\}, q)]$$

$$= \arg\max_{a}[\sum_{m=1}^{M} \lambda_m f_m(a, \{a_1\, a_2 \ldots a_A\}, q)]$$

Maximum Entropy is used to determine optimal values for the coefficients λ that weight the various feature functions relative to each other.

3.2.2 Feature Functions

For the base pattern-based answer selection module described in this section, we use only five basic feature functions.

1. Pattern: The pattern that fired for the given answer and Qtarget. This is a binary feature function: 0 if no pattern fired, 1 if some pattern(s) did.
2. Frequency: Magnini et al. (2002) have observed that the correct answer usually has a higher frequency in the collection of answer chunks. Hence we count the number of time a potential answer occurs in the IR output and use its logarithm as a feature. This is a positive continuous valued feature.
3. Expected Answer Class (Qtarget): If the answer's class matches the Qtarget, as derived from the question by CONTEX, this feature fires (i.e., it has a value of 1). Details of this module are explained in (Hovy et al., 2002). This is a binary-valued feature.
4. Question Word Absent: Usually a correct answer sentence contains a few of the question words. This feature fires if the candidate answer does not contain any of the question words. This is also a binary valued feature.
5. Word Match: This function is the sum of ITF[2] values for the words in the question that match identically with words in the answer sentence. This is a positive continuous valued feature.

3.2.3 Training the system

We use the 1192 questions in the TREC 9 and TREC 10 data sets for training and the 500 questions in the TREC 11 data set for cross-validation. We extract 5000 candidate answers for each question using CONTEX. For each such answer we use the pattern file supplied by NIST to tag answer

[2] ITF = Inverse Term Frequency. We take a large independent corpus & estimate ITF(W) =1/(count(W)), where W = Word.

chunks as either correct (1) or incorrect (0). This is a very noisy way of tagging data: in some cases, even though the answer chunk may be tagged as correct, it may not be supported by the accompanying sentence, while in other cases, a correct chunk may be graded as incorrect, since the pattern file list did not represent a exhaustive list of answers.

3.3 Results

When evaluated against the answer patterns we created for the 413 factoid questions in the TREC-2003 collection, the pattern-based answer selection module described in this section produced 25.18% correct, exact answers. There were 35.59% correct, exact answers in the top 5 candidates returned by a system, with a corresponding MRR score of 28.57%. More details appear in Section 5.

4. STATISTICS-BASED ANSWER SELECTION

Being inspired by the success of noisy-channel-based approaches in applications as diverse as speech recognition (Jelinek, 1997), part of speech tagging (Church, 1988), machine translation (Brown et al., 1993), information retrieval (Berger and Lafferty, 1999), and text summarization (Knight and Marcu, 2002), we have developed a noisy channel model for question answering. This model explains how a given sentence S_A that contains an answer sub-string A to a question Q can be rewritten into Q through a sequence of stochastic operations. Given a corpus of question-answer pairs (Q, S_A), we can train a probabilistic model for estimating the conditional probability P(Q | S_A). Once the parameters of this model are learned, given a question Q and the set of sentences Σ returned by an IR engine, one can find the sentence $S_i \in \Sigma$ and an answer in it $A_{i,j}$ by searching for the $S_{i,A_{i,j}}$ that maximizes the conditional probability P(Q | $S_{i,A_{i,j}}$).

4.1 A Noisy-Channel Model for Question Answering

Assume that we want to explain, for instance, why "1977" in sentence S_A in Figure 1 is a good answer for the question "When did Elvis Presley die?" To do this, we build a noisy channel model that makes explicit how answer sentence parse trees are mapped into questions. Consider, for example, the automatically derived answer sentence parse tree in Figure 1, which associates to nodes both syntactic and shallow semantic, named-entity-specific tags. In order to rewrite this tree into a question, we assume the following generative story:

1. In general, answer sentences are much longer than typical factoid questions. To reduce the length gap between questions and answers and to increase the likelihood that our models can be adequately trained, we first make a "cut" in the answer parse tree and select a sequence of words, syntactic, and semantic tags. The "cut" is made so that every word in the answer sentence or one of its ancestors belongs to the "cut" and no two nodes on a path from a word to the root of the tree are in the "cut". Figure 1 depicts such a cut graphically.
2. Once the "cut" has been identified, we mark one of its elements as the answer string. In Figure 1, we decide to mark DATE as the answer string (A_DATE).
3. There is no guarantee that the number of words in the cut and the number of words in the question match. To account for this, we stochastically assign to every element s_i in a cut a fertility according to table $n(\phi \mid s_i)$. We delete elements of fertility 0 and duplicate elements of fertility 2, etc. With probability p_1 we also increment the fertility of an invisible word NULL. NULL and fertile words, i.e. words with fertility strictly greater than 1, enable us to align long questions with short answers. Zero fertility words enable us to align short questions with long answers.
4. Next, we replace answer words (including the NULL word) with question words according to the table $t(q_i \mid s_j)$.
5. In the last step, we permute the question words according to a distortion table d, in order to obtain a well-formed, grammatical question.

The probability $P(Q \mid S_A)$ is computed by multiplying the probabilities in all the steps of our generative story (Figure 1 lists some of the factors specific to this computation.) The readers familiar with the statistical machine translation (SMT) literature should recognize that steps 3 to 5 are nothing but a one-to-one reproduction of the generative story proposed in the SMT context by Brown et al. (see (Brown et al., 1993) for a detailed mathematical description of the model and the formula for computing the probability of an alignment and target string given a source string).[3]

To simplify our work and to enable us exploit existing off-the-shelf software, in the experiments we carried out, we assumed a flat distribution

[3] The distortion probabilities depicted in Figure 1 are a simplification of the distortions used in the IBM Model 4 model by Brown et al. (1993). We chose this watered down representation only for illustrative purposes. Our QA system implements the full-blown Model 4 statistical model described by Brown et al.

for the first two steps in our generative story. That is, we assumed that it is equally likely to take any cut in the tree and equally likely to choose as answer any syntactic/semantic element in an answer sentence.

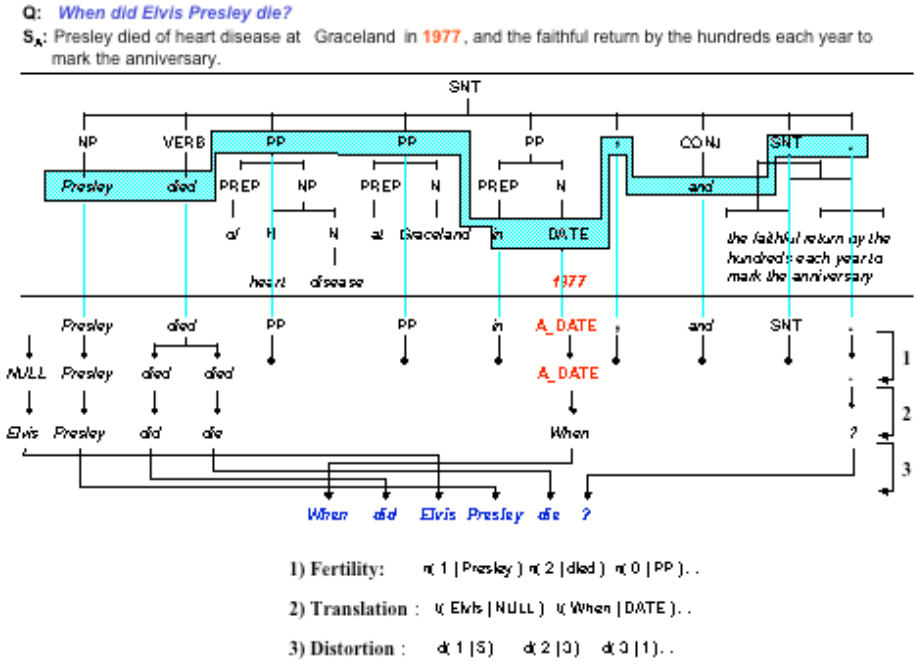

Figure 1. A generative model for question answering.

4.2 Training the Model

Assume that the question-answer pair in Figure 1 appears in our training corpus. When this happens, we know that *1977* is the correct answer. To generate a training example from this pair, we tokenize the question, we parse the answer sentence, we identify the question terms and answer in the parse tree, and then we make a "cut" in the tree that satisfies the following conditions:

a. Terms overlapping with the question are preserved as surface text

b. The answer is reduced to its semantic or syntactic class prefixed with the symbol "A_"

c. Non-leaves, which don't have any question term or answer offspring, are reduced to their semantic or syntactic class.

d. All remaining nodes (leaves) are preserved as surface text.

Condition a) ensures that the question terms will be identified in the sentence. Condition b) helps learn answer types. Condition c) brings the sentence closer to the question by compacting portions that are syntactically far from question terms and answer. And finally the importance of lexical cues around question terms and answer motivates condition d). For the question-answer pair in Figure 1, the algorithm above generates the following training example:

Q: When did Elvis Presley die ?

S_A: *Presley died* PP PP *in* **A_DATE***, and* SNT.

Figure 2 represents graphically the conditions that led to this training example being generated.

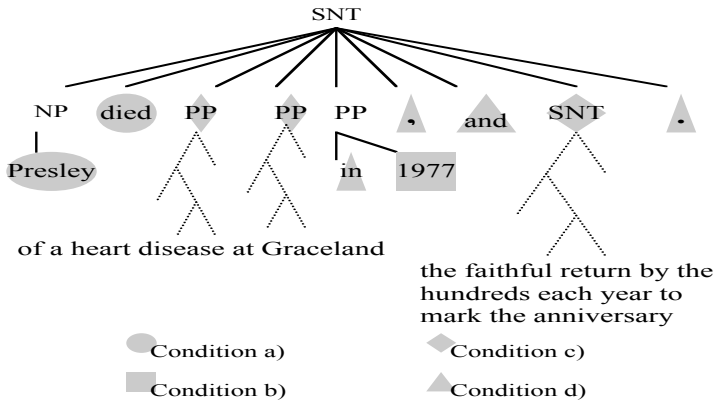

Figure 2. Generation of QA examples for training..

Our algorithm for generating training pairs implements deterministically the first two steps in our generative story. The algorithm is constructed so as to be consistent with our intuition that a generative process that makes the question and answer as similar-looking as possible is most likely to enable us learn a useful model. Each question-answer pair results in one training example. It is the examples generated through this procedure that we use to estimate the parameters of our model.

For training, we use TREC 9 and TREC 10 questions (1091) with answer sentences (18618) automatically generated from the corresponding judgment sets. We also use questions (2000) from http://www.quiz-zone.co.uk with answer sentences (6516) semi-automatically collected from the web and

annotated for correctness by a linguist[4]. To estimate the parameters of our model, we use GIZA, a publicly available statistical machine translation package (http://www.clsp.jhu.edu/ws99/projects/mt/).

4.3 Using the Model to Select an Answer String

Assume now that the sentence in Figure 1 is returned by an IR engine as a potential candidate for finding the answer to the question "When did Elvis Presley die?" In this case, we don't know what the answer is, so we assume that any semantic/syntactic node in the answer sentence can be the answer, with the exception of the nodes that subsume question terms and stop words. In this case, given a question and a potential answer sentence, we generate an exhaustive set of question-answer sentence pairs, each pair labeling as answer (A_) a different syntactic/semantic node. Here are some of the question-answer sentence pairs we consider for the example in Figure 1:

Q: When did Elvis Presley die *?*

S_{A1}: Presley died A_PP PP PP , and SNT .

Q: When did Elvis Presley die *?*

S_{Ai}: Presley died PP PP in A_DATE , and SNT .

Q: When did Elvis Presley die *?*

S_{Aj}: Presley died PP PP PP , and NP return by A_NP NP .

If we learned a good model, we would expect it to assign a higher probability to $P(Q \mid S_{Ai})$ than to $P(Q \mid S_{A1})$ and $P(Q \mid S_{Aj})$.

Hence, to select the answer string for a question Q, we exhaustively generate all $Q–S_{i,A_{i,j}}$ pairs for each answer sentence S_i, use GIZA to assess $P(Q \mid S_{i,A_{i,j}})$ and pick the answer $A_{i,j}$ that maximizes $P(Q \mid S_{i,A_{i,j}})$. Figure 3 depicts graphically the noisy-channel based answer selection.

[4] We are grateful to Miruna Ticrea for annotating the question-answer pairs.

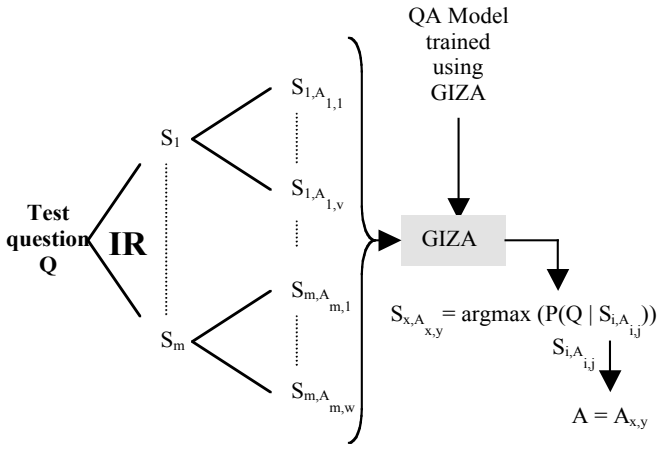

Figure 3. The noisy-channel-based answer selection.

4.4 Results

When evaluated against the answer patterns we created for the 413 factoid questions in the TREC-2003 collection, the statistical-based answer selection module described in this section produced 21.30% correct, exact answers. There were 31.23% correct, exact answers in the top 5 candidates returned by a system, with a corresponding MRR score of 24.83%. For more details, see Section 5.

5. MAXIMUM ENTROPY TO IMPROVE INDIVIDUAL ANSWER SELECTION MODULES AND COMBINE THEIR OUTPUTS

5.1 Motivation

Simple inspection of the outputs produced by our individual components on a development corpus of questions from the TREC-2002 collection revealed several consistent patterns of error:

- The pattern-based answer selection module performs well on questions whose Qtargets are recognizable named entities (NAMES, ORGANIZATIONS, LOCATIONS). However, this module performs less well on questions with more general QTargets (NPs, for example).

- The statistics-based answer selection module does not restrict the types of the answers it expects for a question Qtargets: it assumes that any semantic constituent can be an answer. As a consequence, many of the answers produced by this module may not be exact.
- Overall, all modules made some blatant mistakes. The pattern-based and statistics-based modules in particular sometimes select as top answers strings like “he”, “she”, and “it”, which are unlikely to be good answers for any factoid question one may imagine.

To address these problems, we decided to use the maximum entropy framework to re-rank the answers produced by the answer selection modules and root out the blatant errors. In addition to fixing these blatant errors, we also hoped to also create the means for capitalizing on the strengths of the individual modules. A post-analysis of the TREC 2003 questions showed that if we used an oracle to select the best answer from the pooled top 50 answers returned by each of the three modules, we could produce 77.23% correct, exact answers. This result shows that there is a big opportunity to increase the performance of our end-to-end system by combining in a suitable manner the outputs of the various answer selection modules.

5.2 Maximum Entropy Framework

There are several ways of combining the output of various systems. One simple way would be to add or multiply the answer score produced by various systems. However, this method would not exploit several useful dependencies (e.g., answer redundancy or Qtarget matching) that are not explicitly modeled by all the individual answer selection modules. A good approach to overcoming this problem is to use machine learning re-ranking wherein these dependencies and the individual answer scores are modeled as feature functions. We implemented this re-ranking approach using maximum entropy, which is a linear classifier. We chose to work with a linear classifier because we had very little data for training; in addition, any non-linear classifier is more prone to over-fitting on the training data (principle of bias-variance trade-off).

As in Section 3, we model the problem as a re-ranker as follows (Ravichandran et al., 2003):

$$P(a \mid \{a_1 a_2 ... a_A\}, q) = \frac{\exp[\sum_{m=1}^{M} \lambda_m f_m(a, \{a_1 a_2 ... a_A\}, q)]}{\sum_{a'} \exp[\sum_{m=1}^{M} \lambda_m f_m(a', \{a_1 a_2 ... a_A\}, q)]}$$

where the decision rule for the re-ranker is given by:

$$\hat{a} = \arg\max_a [P(a \mid \{a_1 a_2 ... a_A\}, q)]$$

$$= \arg\max_a [\sum_{m=1}^{M} \lambda_m f_m(a, \{a_1 a_2 ... a_A\}, q)]$$

5.3 Feature Functions

We use 48 different types of feature functions to adjust the parameters of the Maximum Entropy model. These features can be broadly classified into the following categories.

1. Component-specific: Scores from the individual answer selection module and associated features like the rank of the answer and the presence/absence of the answers produced by the individual answer selection module. We also add features based on the scores produced by the IR module and word overlap between question and answers.
2. Redundancy-specific: Count of candidate answers in the collection. We also add additional features for the logarithm and the square root of the counts.
3. Qtarget-specific: It was observed that some of the answer selection modules answer certain Qtargets better than others. We therefore model a set of features wherein we combine certain classes of Qtarget with the score of the individual answer selection module. This enables the maximum entropy model to assign different model parameters to different types of questions and answer selection module.
4. Blatant-error-specific: We model a set of features based on the development set in an iterative process. These include (negative) features such as “answers usually do not have personal pronouns” and “WHEN questions usually do not contain day of the week as answer”, among others.

5.4 Experiments

We perform the following three sets of experiments.

1. Maximum Entropy re-ranking performed on individual answer selection modules: In this experiment, we turn off all the features in any answer selection module that depends on another answer selection module and we use for re-ranking only the top 50 answers supplied by one answer selection component. Thus this experiment enables us to assess the impact separately on each individual answer selection module of redundancy-, Qtarget-, and blatant-error-specific features, as well as improved weighting of the features specific to each module.
2. Maximum Entropy re-ranking on all answer selection modules: In this experiment we add all the features described in Section 5.3 and test the performance of the combined system after re-ranking. Re-ranking is performed on the answer set that results from combining the top 50 answers returned by each individual answer selection module. This experiment enables us to assess whether the ME framework enables us to better exploit strengths specific to each answer selection module.
3. Feature Selection: Since we have only 200 training examples and almost 50 features, the training is likely to be highly prone to over-fitting. To avoid this problem we apply feature selection as described in (Della Pietra et al., 1996). This reduces the number of features from 48 to 31.

Table 1 summarizes the results: it shows the percentage of correct, exact answers returned by each answer selection module with and without ME-based re-ranking, as well as the percentage of correct, exact answers returned by an end-to-end QA system that uses all three answer selection modules together. Table 1 also shows the performance of these systems in terms of percentage of correct answers ranked in the top 5 answers and the corresponding MRR scores.

Metric	Knowledge-Based		Pattern-Based		Statistical-Based		Base from all systems followed by ME re-ranking (no feature selection)	Base from all systems followed by ME re-ranking (with feature selection)
	Base	Base followed by ME re-ranking	Base	Base followed by ME re-ranking	Base	Base followed by ME re-ranking		
Top answer	35.83%	45.03%	25.18%	30.50%	21.30%	32.20%	46.37%	47.21%
Top 5	57.38%	56.41%	35.59%	43.09%	31.23%	40.92%	57.62%	57.62%
MRR	43.88	49.36	28.57	35.37	24.83	35.51	51.07	51.27

Table 1. Performance of various answer selection modules in TextMap, an end-to-end QA system.

The results in Table 1 show that appropriate weighting of the features used by each answer selection module as well as the ability to capitalize on global features, such as the counts associated with each answer, are extremely important means for increasing the overall performance of a QA system. ME re-ranking led to significant increases in performance for each answer selection module individually. When measured with respect to the ability of our system to find correct, exact answers when returning only the top answer, it accounted for 14.33% reduction in the error rate of the knowledge-based answer selection module; 7.1% reduction in the error rate of the pattern-based answer selection module; and 13.85% reduction in the error rate of the statistical-based answer selection module. Combining the outputs of all systems yields an additional increase in performance; when over-fitting is avoided through feature selection, the overall reduction in error rate is 3.96%. This corresponds to an increase in performance from 45.03% to 47.21% on the top-answer accuracy metric.

The inspection of the weights computed by the ME-based feature selection algorithm shows that our log-linear model was able to lock and exploit strengths and weaknesses of the various modules. For example, the weight learned for a feature that assesses the likelihood of the pattern-based module to find correct answers for questions that have QUANTITY as a Qtarget is negative, which suggests that the pattern-based module is not good at answering QUANTITY-type questions. The weight learned for a feature that assesses the likelihood of the statistics-based module to find correct answers for questions that have an UNKNOWN or NP Qtarget is large, which suggests that the statistical module is better suited for answering these types of questions. The knowledge-based module is better than the others in answering QUANTITY and DEFINITION questions.

These results are quite intuitive: The pattern-based module did not have enough QUANTITY-type questions in the training corpus to learn any useful patterns. The Statistics-based module implemented here does not explicitly use the Qtarget in answer selection and does not model the source probability of a given string being a correct answer. As a consequence, it explores a much larger space of choices when determining an answer compared to the other two modules.

At the moment, the knowledge-based module yields the best individual results. This is not surprising given that it is a module that was engineered carefully, over a period of three years, to accommodate various types of Qtargets and knowledge resources. Many of the resources used by the knowledge-based module are not currently exploited by the other modules: the semantic relations identified by CONTEX; the ability to exploit the paraphrase patterns and advanced forms of reformulation for answer pinpointing; the external sources of knowledge (WordNet; abbreviation lists; etc.); and a significant set of heuristics (see Section 2.4).

Our results suggest that in order to build good answer selection modules, one needs to both exploit as many sources of knowledge as possible and have good methods for integrating them. The sources of knowledge used only by the knowledge-based answer selection module proved to have a stronger impact on the overall performance of our answer selection systems than the ability to automatically train parameters in the pattern- and statistics-based systems, which use poorer representations. Yet, the ability to properly weight the contribution of various knowledge resources was equally important. For example, Maximum Entropy naturally integrated additional features into the knowledge-based answer selection module; a significant part of the 9.2% increase in correct answers reported in Table 1 can be attributed to the addition of redundancy features, a source of knowledge that was unexploited by the base system.

6. CONCLUSIONS

Given their large conceptual similarity, factoid question answering may seem an extension of Information Retrieval, with substrings and sentences replacing documents and document collections. However, the problem has so far stubbornly resisted attempts to find a single, uniform solution that provides as good results as systems employing multiple parallel strategies to perform answer selection.

In this paper, we show how three rather different modules provide somewhat complementary results, and in the paragraphs above, we suggest

some reasons for the complementarity. It is of course possible that a uniform solution will be found one day, but whether that solution will rest upon as simple a representation as vector spaces, or be as easy to train up as surface patterns, remains to be seen. After all, as TextMap shows, factoid QA seems to benefit both from relatively deeper notions such as Qtypes and (semantic) parsing as much as from relatively shallower operations such as reformulations of surface patterns (whether created manually or by learning, and whether the starting point is anchor terms or input questions). A uniform representation/approach that encompasses all this has yet to be conceived.

Nonetheless, we remain optimistic that by investigating alternative approaches, we will be able to determine which types of modules work best for which types of problem variations (including parameters such as domain, amount of training data, ease of acquisition of knowledge, types of question asked, complexity of answers required, etc.).

7. REFERENCES

Berger, A. and J. Lafferty. 1999. Information Retrieval as Statistical Translation. *Proceedings of the 22nd Annual International ACM SIGIR Conference on Information Retrieval*. Berkeley, CA, 222-229.

Bikel, D., R. Schwartz, and R. Weischedel. 1999. An Algorithm that Learns What's in a Name. *Machine Learning—Special Issue on NL Learning*, 34, 1–3.

Brill, E., J. Lin, M. Banko, S. Dumais, and A. Ng. 2001. Data-Intensive Question Answering. *Proceedings of the TREC-10 Conference*. NIST, Gaithersburg, MD, 183–189.

Brown, P.F., J. Cocke, S.A. Della Pietra, V.J. Della Pietra, F. Jelinek, J.D. Lafferty, R.L. Mercer, and P.S. Roossin. 1990. A Statistical Approach to Machine Translation. *Computational Linguistics* 16(2):79–85.

Brown, P.F., S. Della Pietra, V. Della Pietra and R. Mercer. 1993. The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics* 19(2):263-311.

Callan, J.P., W.B. Croft, and J. Broglio. 1995. "TREC and Tipster Experiments with Inquery. *Information Processing and Management* 31(3), 327-343.

Church, K.W. 1988. A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. *Proceedings of the Second Conference in Applied Natural Language Processing*. 136-143.

Della Pietra, S., V. Della Pietra, and J. Lafferty. 1995. Inducing Features of Random Fields. *Technical Report* Department of Computer Science, Carnegie-Mellon University, CMU-CS, 95-144.

Gusfield, D. 1997. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Chapter 6: Linear Time Construction of Suffix Trees, 94–121.

Hermjakob, U. 1997. *Learning Parse and Translation Decisions from Examples with Rich Context*. Ph.D. dissertation, University of Texas Austin. file://ftp.cs.utexas.edu/pub/mooney/papers/hermjakob-dissertation-97.ps.gz.

Hermjakob, U. 2001. Parsing and Question Classification for Question Answering. *Proceedings of the Workshop on Question Answering at ACL-2001*. Toulouse, France.

Hermjakob, U., A. Echihabi, and D. Marcu. 2002. Natural Language Based Reformulation Resource and Web Exploitation for Question Answering. *Proceedings of the TREC-11*. NIST, Gaithersburg, MD.

Hovy, E.H., U. Hermjakob, C.-Y. Lin, and D. Ravichandran. 2002. Using Knowledge to Facilitate Pinpointing of Factoid Answers. *Proceedings of the COLING-2002 Conference*. Taipei, Taiwan.

Ittycheriah, A. 2001. *Trainable Question Answering System*. Ph.D. Dissertation, Rutgers, The State University of New Jersey, New Brunswick, NJ.

Jelinek, F. 1997. *Statistical Methods for Speech Recognition*. MIT Press, Cambridge, MA.

Knight, K. and D. Marcu. 2002. Summarization Beyond Sentence Extraction: A Probabilistic Approach to Sentence Compression. *Artificial Intelligence*, 139(1).

Lin. D. and P. Pantel. 2001. Discovery of Interface Rules for Question Answering. *Journal for Natural Language Engineering* 7(4), 343-360.

Magnini, B., M. Negri, R. Prevete, and H. Tanev. 2002. Is it the Right Answer? Exploiting Web Redundancy for Answer Validation. *Proceedings of the 40th Meeting of the Association of Computational Linguistics (ACL)*. Philadelphia, PA, 425–432.

Oh, JH., KS. Lee, DS. Chang, CW. Seo, and KS. Choi. 2001. TREC-10 Experiments at KAIST: Batch Filtering and Question Answering. *Proceedings of the TREC-10 Conference*. NIST, Gaithersburg, MD, 354–361.

Ravichandran, D. and E.H. Hovy. 2002. Learning Surface Text Patterns for a Question Answering System. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*. Philadelphia, PA 41–47.

Ravichandran, D., E.H. Hovy and F.J. Och. 2003. Statistical QA - Classifier vs Re-ranker: What's the difference? *Proceedings of the Association for Computational Linguistics (ACL) Workshop on Multilingual Summarization and Question Answering--Machine Learning and Beyond*. Sapporo, Japan, 69-75.

Soubbotin, M.M. and S.M. Soubbotin. 2001. Patterns of Potential Answer Expressions as Clues to the Right Answer. *Proceedings of the TREC-10 Conference*. NIST, Gaithersburg, MD, 175–182.

Voorhees, E. 1999. Overview of the Question Answering Track. *Proceedings of the TREC-8 Conference*. NIST, Gaithersburg, MD, 71–81.

Voorhees, E. 2000. Overview of the Question Answering Track. *Proceedings of the TREC-9 Conference*. NIST, Gaithersburg, MD, 71–80.

Voorhees, E. 2001. Overview of the Question Answering Track. *Proceedings of the TREC-10 Conference*. NIST, Gaithersburg, MD, 157–165.

Voorhees, E. 2002. Overview of the Question Answering Track. *Proceedings of the TREC-11 Conference*. NIST, Gaithersburg, MD, 115–123.