In Proceedings of the ACL Conference, 2002

Learning Surface Text Patterns for a Question Answering System

Deepak Ravichandran and Eduard Hovy

Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292-6695
USA
{ravichan,hovy}@isi.edu

Abstract

In this paper we explore the power of surface text patterns for open-domain question answering systems. In order to obtain an optimal set of patterns, we have developed a method for learning such patterns automatically. A tagged corpus is built from the Internet in a bootstrapping process by providing a few hand-crafted examples of each question type to Altavista. Patterns are then automatically extracted from the returned documents and standardized. We calculate the precision of each pattern, and the average precision for each question type. These patterns are then applied to find answers to new questions. Using the TREC-10 question set, we report results for two cases: answers determined from the TREC-10 corpus and from the web.

1 Introduction

Most of the recent open domain question-answering systems use external knowledge and tools for answer pinpointing. These may include named entity taggers, WordNet, parsers, hand-tagged corpora, and ontology lists (Srihari and Li, 00; Harabagiu et al., 01; Hovy et al., 01; Prager et al., 01). However, at the recent TREC-10 QA evaluation (Voorhees, 01), the winning system used just one resource: a fairly extensive list of surface patterns (Soubbotin and Soubbotin, 01). The apparent power of such patterns surprised many. We therefore decided to investigate their potential by acquiring patterns automatically and to measure their accuracy.

It has been noted in several QA systems that certain types of answer are expressed using characteristic phrases (Lee et al., 01; Wang et al., 01). For example, for BIRTHDATEs (with questions like "When was X born?"), typical answers are

"Mozart was born in 1756."

"Gandhi (1869–1948)…"

These examples suggest that phrases like

"<NAME> was born in <BIRTHDATE>"

"<NAME> (<BIRTHDATE>–"

when formulated as regular expressions, can be used to locate the correct answer.

In this paper we present an approach for automatically learning such regular expressions (along with determining their precision) from the web, for given types of questions. Our method uses the machine learning technique of bootstrapping to build a large tagged corpus starting with only a few examples of QA pairs. Similar techniques have been investigated extensively in the field of information extraction (Riloff, 96). These techniques are greatly aided by the fact that there is no need to hand-tag a corpus, while the abundance of data on the web makes it easier to determine reliable statistical estimates.

Our system assumes each sentence to be a simple sequence of words and searches for repeated word orderings as evidence for useful answer phrases. We use suffix trees for extracting substrings of optimal length. We borrow the idea of suffix trees from computational biology (Gusfield, 97) where it is primarily used for detecting DNA sequences. Suffix trees can be processed in time linear on the size of the corpus and, more importantly, they do not restrict the length of substrings. We then test the patterns learned by our system on new unseen questions from the TREC-10 set and evaluate their results to determine the precision of the patterns.

2 Learning of Patterns

We describe the pattern-learning algorithm with an example. A table of patterns is constructed for each individual question type by the following procedure (Algorithm 1).

1. Select an example for a given question type. Thus for BIRTHYEAR questions we select "Mozart 1756" (we refer to "Mozart" as the question term and "1756" as the answer term).
2. Submit the question and the answer term as queries to a search engine. Thus, we give the query +"Mozart" +"1756" to AltaVista (http://www.altavista.com).
3. Download the top 1000 web documents provided by the search engine.
4. Apply a sentence breaker to the documents.
5. Retain only those sentences that contain both the question and the answer term. Tokenize the input text, smooth variations in white space characters, and remove html and other extraneous tags, to allow simple regular expression matching tools such as egrep to be used.
6. Pass each retained sentence through a suffix tree constructor. This finds all substrings, of all lengths, along with their counts. For example consider the sentences "The great composer Mozart (1756–1791) achieved fame at a young age" "Mozart (1756–1791) was a genius", and "The whole world would always be indebted to the great music of Mozart (1756–1791)". The longest matching substring for all 3 sentences is "Mozart (1756–1791)", which the suffix tree would extract as one of the outputs along with the score of 3.
7. Pass each phrase in the suffix tree through a filter to retain only those phrases that contain both the question and the answer term. For the example, we extract only those phrases from the suffix tree that contain the words "Mozart" and "1756".
8. Replace the word for the question term by the tag "<NAME>" and the word for the answer term by the term "<ANSWER>".

This procedure is repeated for different examples of the same question type. For BIRTHDATE we also use "Gandhi 1869", "Newton 1642", etc.

For BIRTHDATE, the above steps produce the following output:

```
a. born in <ANSWER> , <NAME>
b. <NAME> was born on <ANSWER> ,
c. <NAME> ( <ANSWER> -
d. <NAME> ( <ANSWER - )
...
```

These are some of the most common substrings of the extracted sentences that contain both <NAME> and <ANSWER>. Since the suffix tree records all substrings, partly overlapping strings such as c and d are separately saved, which allows us to obtain separate counts of their occurrence frequencies. As will be seen later, this allows us to differentiate patterns such as d (which records a still living person, and is quite precise) from its more general substring c (which is less precise).

Algorithm 2: Calculating the precision of each pattern.

1. Query the search engine by using only the question term (in the example, only "Mozart").
2. Download the top 1000 web documents provided by the search engine.
3. As before, segment these documents into individual sentences.
4. Retain only those sentences that contain the question term.

5. For each pattern obtained from Algorithm 1, check the presence of each pattern in the sentence obtained from above for two instances:
   i) Presence of the pattern with <ANSWER> tag matched by any word.
   ii) Presence of the pattern in the sentence with <ANSWER> tag matched by the correct answer term.

   In our example, for the pattern "<NAME> was born in <ANSWER>" we check the presence of the following strings in the answer sentence
   i) Mozart was born in <ANY_WORD>
   ii) Mozart was born in 1756

   Calculate the precision of each pattern by the formula $P = C_a / C_o$ where

   C_a = total number of patterns with the answer term present

   C_o = total number of patterns present with answer term replaced by any word
6. Retain only the patterns matching a sufficient number of examples (we choose the number of examples > 5).

We obtain a table of regular expression patterns for a given question type, along with the precision of each pattern. This precision is the probability of each pattern containing the answer and follows directly from the principle of maximum likelihood estimation.

For BIRTHDATE the following table is obtained:

```
1.0   <NAME>( <ANSWER> - )
0.85  <NAME> was born on <ANSWER>,
0.6   <NAME> was born in <ANSWER>
0.59  <NAME> was born <ANSWER>
0.53  <ANSWER> <NAME> was born
0.50  - <NAME> ( <ANSWER>
0.36 <NAME> ( <ANSWER> -
```

For a given question type a good range of patterns was obtained by giving the system as few as 10 examples. The rather long list of patterns obtained would have been very difficult for any human to come up with manually.

The question term could appear in the documents obtained from the web in various ways. Thus "Mozart" could be written as "Wolfgang Amadeus Mozart", "Mozart, Wolfgang Amadeus", "Amadeus Mozart" or "Mozart". To learn from such variations, in step 1 of Algorithm 1 we specify the various ways in which the question term could be specified in the text. The presence of any of these names would cause it to be tagged as the original question term "Mozart".

The same arrangement is also done for the answer term so that presence of any variant of the answer term would cause it to be treated exactly like the original answer term. While easy to do for BIRTHDATE, this step can be problematic for question types such as DEFINITION, which may contain various acceptable answers. In general the input example terms have to be carefully selected so that the questions they represent do not have a long list of possible answers, as this would affect the confidence of the precision scores for each pattern. All the answers need to be enlisted to ensure a high confidence in the precision score of each pattern, in the present framework.

The precision of the patterns obtained from one QA-pair example in algorithm 1 is calculated from the documents obtained in algorithm 2 for other examples of the same question type. In other words, the precision scores are calculated by cross-checking the patterns across various examples of the same type. This step proves to be very significant as it helps to eliminate dubious patterns, which may appear because the contents of two or more websites may be the same, or the same web document reappears in the search engine output for algorithms 1 and 2.

Algorithm 1 does not explicitly specify any particular question type. Judicious choice of the QA example pair therefore allows it to be used for many question types without change.

3 Finding Answers

Using the patterns to answer a new question we employ the following algorithm:

1. Determine the question type of the new question. We use our existing QA system (Hovy et al., 2002b; 2001) to do so.

2. The question term in the question is identified, also using our existing system.
3. Create a query from the question term and perform IR (by using a given answer document corpus such as the TREC-10 collection or web search otherwise).
4. Segment the documents obtained into sentences and smooth out white space variations and html and other tags, as before.
5. Replace the question term in each sentence by the question tag ("<NAME>", in the case of BIRTHYEAR).
6. Using the pattern table developed for that particular question type, search for the presence of each pattern. Select words matching the tag "<ANSWER>" as the answer.
7. Sort these answers by their pattern's precision scores. Discard duplicates (by elementary string comparisons). Return the top 5 answers.

4 Experiments

From our Webclopedia QA Typology (Hovy et al., 2002a) we selected 6 different question types: BIRTHDATE, LOCATION, INVENTOR, DISCOVERER, DEFINITION, WHY-FAMOUS. The pattern table for each of these question types was constructed using Algorithm 1.

Some of the patterns obtained along with their precision are as follows

BIRTHYEAR
- 1.0 <NAME> (<ANSWER> -)
- 0.85 <NAME> was born on <ANSWER> ,
- 0.6 <NAME> was born in <ANSWER>
- 0.59 <NAME> was born <ANSWER>
- 0.53 <ANSWER> <NAME> was born
- 0.5 - <NAME> (<ANSWER>
- 0.36 <NAME> (<ANSWER> -
- 0.32 <NAME> (<ANSWER>) ,
- 0.28 born in <ANSWER> , <NAME>
- 0.2 of <NAME> (<ANSWER>

INVENTOR
- 1.0 <ANSWER> invents <NAME>
- 1.0 the <NAME> was invented by <ANSWER>
- 1.0 <ANSWER> invented the <NAME> in
- 1.0 <ANSWER> ' s invention of the <NAME>
- 1.0 <ANSWER> invents the <NAME> .
- 1.0 <ANSWER> ' s <NAME> was
- 1.0 <NAME> , invented by <ANSWER>
- 1.0 <ANSWER> ' s <NAME> and
- 1.0 that <ANSWER> ' s <NAME>
- 1.0 <NAME> was invented by <ANSWER> ,

DISCOVERER
- 1.0 when <ANSWER> discovered <NAME>
- 1.0 <ANSWER> ' s discovery of <NAME>
- 1.0 <ANSWER> , the discoverer of <NAME>
- 1.0 <ANSWER> discovers <NAME> .
- 1.0 <ANSWER> discover <NAME>
- 1.0 <ANSWER> discovered <NAME> , the
- 1.0 discovery of <NAME> by <ANSWER>.
- 0.95 <NAME> was discovered by <ANSWER>
- 0.91 of <ANSWER> ' s <NAME>
- 0.9 <NAME> was discovered by <ANSWER> in

DEFINITION
- 1.0 <NAME> and related <ANSWER>s
- 1.0 <ANSWER> (<NAME> ,
- 1.0 <ANSWER> , <NAME> .
- 1.0 , a <NAME> <ANSWER> ,
- 1.0 (<NAME> <ANSWER>) ,
- 1.0 form of <ANSWER> , <NAME>
- 1.0 for <NAME> , <ANSWER> and
- 1.0 cell <ANSWER> , <NAME>
- 1.0 and <ANSWER> > <ANSWER> > <NAME>
- 0.94 as <NAME> , <ANSWER> and

WHY-FAMOUS
- 1.0 <ANSWER> <NAME> called
- 1.0 laureate <ANSWER> <NAME>
- 1.0 by the <ANSWER> , <NAME> ,
- 1.0 <NAME> - the <ANSWER> of
- 1.0 <NAME> was the <ANSWER> of
- 0.84 by the <ANSWER> <NAME> ,
- 0.8 the famous <ANSWER> <NAME> ,
- 0.73 the famous <ANSWER> <NAME>
- 0.72 <ANSWER> > <NAME>
- 0.71 <NAME> is the <ANSWER> of

LOCATION

1.0 <ANSWER> ' s <NAME> .
1.0 regional : <ANSWER> : <NAME>
1.0 to <ANSWER> ' s <NAME> ,
1.0 <ANSWER> ' s <NAME> in
1.0 in <ANSWER> ' s <NAME> ,
1.0 of <ANSWER> ' s <NAME> ,
1.0 at the <NAME> in <ANSWER>
0.96 the <NAME> in <ANSWER> ,
0.92 from <ANSWER> ' s <NAME>
0.92 near <NAME> in <ANSWER>

For each question type, we extracted the corresponding questions from the TREC-10 set. These questions were run through the testing phase of the algorithm. Two sets of experiments were performed. In the first case, the TREC corpus was used as the input source and IR was performed by the IR component of our QA system (Lin, 2002). In the second case, the web was the input source and the IR was performed by the AltaVista search engine.

Results of the experiments, measured by Mean Reciprocal Rank (MRR) score (Voorhees, 01), are:

TREC Corpus

Question type	Number of questions	MRR on TREC docs
BIRTHYEAR	8	0.48
INVENTOR	6	0.17
DISCOVERER	4	0.13
DEFINITION	102	0.34
WHY-FAMOUS	3	0.33
LOCATION	16	0.75

Web

Question type	Number of questions	MRR on the Web
BIRTHYEAR	8	0.69
INVENTOR	6	0.58
DISCOVERER	4	0.88
DEFINITION	102	0.39
WHY-FAMOUS	3	0.00
LOCATION	16	0.86

The results indicate that the system performs better on the Web data than on the TREC corpus. The abundance of data on the web makes it easier for the system to locate answers with high precision scores (the system finds many examples of correct answers among the top 20 when using the Web as the input source). A similar result for QA was obtained by Brill et al. (2001). The TREC corpus does not have enough candidate answers with high precision score and has to settle for answers extracted from sentences matched by low precision patterns. The WHY-FAMOUS question type is an exception and may be due to the fact that the system was tested on a small number of questions.

5 Shortcoming and Extensions

No external knowledge has been added to these patterns. We frequently observe the need for matching part of speech and/or semantic types, however. For example, the question: "Where are the Rocky Mountains located?" is answered by "Denver's new airport, topped with white fiberglass cones in imitation of the Rocky Mountains in the background, continues to lie empty", because the system picked the answer "the background" using the pattern "the <NAME> in <ANSWER>,". Using a named entity tagger and/or an ontology would enable the system to use the knowledge that "background" is not a location.

DEFINITION questions pose a related problem. Frequently the system's patterns match a term that is too general, though correct technically. For "what is nepotism?" the pattern "<ANSWER>, <NAME>" matches "…in the form of widespread bureaucratic abuses: graft, nepotism…"; for "what is sonar?" the pattern "<NAME> and related <ANSWER>s" matches "…while its sonar and related underseas systems are built…".

The patterns cannot handle long-distance dependencies. For example, for "Where is London?" the system cannot locate the answer in "London, which has one of the most busiest airports in the world, lies on the banks of the river Thames" due to the explosive danger of unrestricted wildcard matching, as would be required in the pattern "<QUESTION>,

(<any_word>)*, lies on <ANSWER>". This is one of the reasons why the system performs very well on certain types of questions from the web but performs poorly with documents obtained from the TREC corpus. The abundance and variation of data on the Internet allows the system to find an instance of its patterns without losing answers to long-term dependencies. The TREC corpus, on the other hand, typically contains fewer candidate answers for a given question and many of the answers present may match only long-term dependency patterns.

More information needs to be added to the text patterns regarding the length of the answer phrase to be expected. The system searches in the range of 50 bytes of the answer phrase to capture the pattern. It fails to perform under certain conditions as exemplified by the question "When was Lyndon B. Johnson born?". The system selects the sentence "Tower gained national attention in 1960 when he lost to democratic Sen. Lyndon B. Johnson, who ran for both re-election and the vice presidency" using the pattern "<NAME> <ANSWER> –". The system lacks the information that the <ANSWER> tag should be replaced exactly by one word. Simple extensions could be made to the system so that instead of searching in the range of 50 bytes for the answer phrase it could search for the answer in the range of 1–2 chunks (basic phrases in English such as simple NP, VP, PP, etc.).

A more serious limitation is that the present framework can handle only one anchor point (the question term) in the candidate answer sentence. It cannot work for types of question that require multiple words from the question to be in the answer sentence, possibly apart from each other. For example, in "Which county does the city of Long Beach lie?", the answer "Long Beach is situated in Los Angeles County" requires the pattern. "<QUESTION_TERM_1> situated in <ANSWER> <QUESTION_TERM_2>", where <QUESTION_TERM_1> and <QUESTION_TERM_2> represent the terms "Long Beach" and "county" respectively. The performance of the system depends significantly on there being only one anchor word, which allows a single word match between the question and the candidate answer sentence. The presence of multiple anchor words would help to eliminate many of the candidate answers by simply using the condition that all the anchor words from the question must be present in the candidate answer sentence.

The system does not classify or make any distinction between upper and lower case letters. For example, "What is micron?" is answered by "In Boise, Idaho, a spokesman for Micron, a maker of semiconductors, said Simms are ' a very high volume product for us …' ". The answer returned by the system would have been perfect if the word "micron" had been capitalized in the question.

Canonicalization of words is also an issue. While giving examples in the bootstrapping procedure, say, for BIRTHDATE questions, the answer term could be written in many ways (for example, Gandhi's birth date can be written as "1869", "Oct. 2, 1869", "2nd October 1869", "October 2 1869", and so on). Instead of enlisting all the possibilities a date tagger could be used to cluster all the variations and tag them with the same term. The same idea could also be extended for smoothing out the variations in the question term for names of persons (Gandhi could be written as "Mahatma Gandhi", "Mohandas Karamchand Gandhi", etc.).

6 Conclusion

The web results easily outperform the TREC results. This suggests that there is a need to integrate the outputs of the Web and the TREC corpus. Since the output from the Web contains many correct answers among the top ones, a simple word count could help in eliminating many unlikely answers. This would work well for question types like BIRTHDATE or LOCATION but is not clear for question types like DEFINITION.

The simplicity of this method makes it perfect for multilingual QA. Many tools required by sophisticated QA systems (named entity taggers, parsers, ontologies, etc.) are language specific and require significant

effort to adapt to a new language. Since the answer patterns used in this method are learned using only a small number of manual training terms, one can rapidly learn patterns for new languages, assuming the web search engine is appropriately switched.

References

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.

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

Harabagiu, S., D. Moldovan, M. Pasca, R. Mihalcea, M. Surdeanu, R. Buneascu, R. Gîrju, V. Rus and P. Morarescu. 2001. FALCON: Boosting Knowledge for Answer Engines. *Proceedings of the 9th Text Retrieval Conference* (TREC-9), NIST, 479–488.

Hovy, E.H., U. Hermjakob, and C.-Y. Lin. 2001. The Use of External Knowledge in Factoid QA. *Proceedings of the TREC-10 Conference*. NIST, Gaithersburg, MD, 166–174.

Hovy, E.H., U. Hermjakob, and D. Ravichandran. 2002a. A Question/Answer Typology with Surface Text Patterns. *Proceedings of the Human Language Technology (HLT) conference*. San Diego, CA.

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

Lee, G.G., J. Seo, S. Lee, H. Jung, B-H. Cho, C. Lee, B-K. Kwak, J, Cha, D. Kim, J-H. An, H. Kim, and K. Kim. 2001. SiteQ: Engineering High Performance QA System Using Lexico-Semantic Pattern Matching and Shallow NLP. *Proceedings of the TREC-10 Conference*. NIST, Gaithersburg, MD, 437–446.

Lin, C-Y. 2002. The Effectiveness of Dictionary and Web-Based Answer Reranking. *Proceedings of the COLING-2002 conference*. Taipei, Taiwan.

Prager, J. and J. Chu-Carroll. 2001. Use of WordNet Hypernyms for Answering What-Is Questions. *Proceedings of the TREC-10 Conference*. NIST, Gaithersburg, MD, 309–316.

Riloff, E. 1996. Automatically Generating Extraction Patterns from Untagged Text. *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (AAAI-96), 1044–1049.

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.

Srihari, R. and W. Li. 2000. A Question Answering System Supported by Information Extraction. *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics* (ANLP-NAACL-00), Seattle, WA, 166–172.

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

Wang, B., H. Xu, Z. Yang, Y. Liu, X. Cheng, D. Bu, and S. Bai. 2001. TREC-10 Experiments at CAS-ICT: Filtering, Web, and QA. *Proceedings of the TREC-10 Conference*. NIST, Gaithersburg, MD, 229–241.