Using a Named Entity Tagger to Generalise Surface Matching Text Patterns for Question Answering

Mark A. Greenwood and Robert Gaizauskas
Department of Computer Science
University of Sheffield
Regent Court, Portobello Road
Sheffield S1 4DP UK
{m.greenwood,r.gaizauskas}@dcs.shef.ac.uk

Abstract

This paper explores one particular limitation common to question answering systems which operate by using induced surface matching text patterns – namely the problems concerned with question specific words appearing within the induced answer extraction pattern. We suggest a solution to this problem by generalising the learned answer extraction patterns to include named entity classes.

1 Introduction

Many of the open-domain question answering systems developed prior to 2001 were heavily reliant on external sources of knowledge and tools for pinpointing exact answers within text. These included resources such as WordNet (Miller, 1995), named entity taggers, parsers and gazetteers. The performance of these systems varied widely with one of the best performing systems, FALCON (Harabagiu et al., 2000), being able to answer approximately 65% of the questions (Voorhees, 2000).

However, at the 10th Text REtrieval Conference, TREC 2001 (Voorhees, 2001), a system was entered that used only a single resource consisting of a large collection of surface matching text patterns which were derived using simple machine learning techniques from a set of question-answer pairs and a corpus of documents in which the answer was known to occur (Soubbotin and Soubbotin, 2001). The surprising fact, was that this system did exceptionally well, performing better than any of the other systems entered into the evaluation. The performance of what appeared to be a relatively simple system, provoked so much interest that at least one group, Ravichandran and Hovy (2002), choose to implement their own version.

In this paper we observe that overly specific patterns learned by this approach can cause problems such that it becomes extremely difficult to answer certain types of question. These problems are highlighted using our own implementation which has been developed as the basis for a more advanced question answering system.

2 The Basic Pattern Induction Approach

The following sections detail the algorithms involved in the basic approach to finding and using surface matching text patterns to answer questions.

2.1 Learning Text Patterns

Our approach to learning patterns is similar to that outlined in the paper by Ravichandran and Hovy (2002) as it also relies on suffix trees (Ukkonen, 1995) to extract patterns of an optimal length from unstructured text. The best way of describing the algorithm is through an example. The input to the algorithm is a set of questions of a specific type and their associated exact answer phrases. For comparison with the Ravichandran paper we will use questions of the form *"When was X born?"*. Given this the algorithm is as follows:

1. For each example question of the specific question type, produce a pair consisting of the question term and the answer term. For example:
   - "Abraham Lincoln" "1809"
   - "Adolf Hitler" "1889"
   - "Louisa May Alcott" "1832"
   - "Isaac Asimov" "1920"
2. For each example the question and answer terms are submitted to Google, as a single query, and the top 10 documents are downloaded[1].
3. Each document then has the question term replaced by the single token `AnCHoR` and the answer term by `AnSWeR`.
4. A tokeniser and sentence splitter are then applied to the documents.
5. Those sentences which contain both `AnCHoR` and `AnSWeR` are retained and joined together to create one single document, in which each sentence is separated by a # and the end of the created document is marked by $[2].
6. The single generated document is then used to produce a token-level suffix tree, from which the repeated substrings are then extracted.
7. Finally the list of repeated substrings is filtered to retain only those which contain both `AnCHoR` and `AnSWeR` and do not span a sentence boundary (i.e. do not contain # or $).

This produces a set of patterns for the specific question type. The following are a few of the patterns generated using this approach, for questions of the form *"When was X born?"*:

```
from AnCHoR ( AnSWeR - 1969 )
AnCHoR , AnSWeR -
- AnCHoR ( AnSWeR
from AnCHoR ( AnSWeR -
:  AnCHoR , AnSWeR -
```

Unfortunately some of these patterns are specific to one or more of the questions used to generate them (e.g. the first pattern includes a date of death, which is question specific). A further stage is therefore needed to analyse the patterns to decide which are generic enough to be used to answer unseen questions.

The algorithm used to analyse the patterns and discard those which are not generic, also allows us to associate a numerical precision with each pattern which can later be used as a measure of how confident the system is in any answers it proposes. Continuing with the same example as above, the steps in this algorithm are as follows:

1. Using a different set of question-answer pairs, only the question term is submitted to Google and the top ten documents are downloaded.
2. Each document then has the question term replaced by `AnCHoR` and the answer term (if it appears within the document) is replaced by `AnSWeR`.
3. Those sentences which contain `AnCHoR` are retained and joined together to create one single document.
4. Each of the previously generated patterns is converted to a standard regular expression designed to capture the answer text, giving expressions such as[3]:

   ```
   from AnCHoR \( ([^ ]+) - 1969 \)
   AnCHoR , ([^ ]+) -
   - AnCHoR \( ([^ ]+)
   from AnCHoR \( ([^ ]+) -
   :  AnCHoR , ([^ ]+) -
   ```

   These regular expressions allow us to easily retrieve the single token which `AnSWeR` in the original pattern would have matched against.
5. Each regular expression is then matched against each sentence in the generated document. Along with each pattern, P, two counts are maintained: C_a^P, which counts the total number of times this pattern has matched

1. The documents are actually downloaded from Googles cache to guarantee that we use the version of the page indexed by Google.

2. These separators are a necessary part of the suffix tree construction and processing but they do not appear in the resulting patterns.

3. For those not familiar with standard regular expressions, `([^ ]+)` matches any sequence of one or more non-space characters and captures that part of the text in a variable for latter use.

Regular Expression	*Precision*
`AnCHoR \( ([^ ]+) -`	0.967
`AnCHoR \( ([^ ]+)`	0.566
`AnCHoR ([^ ]+) -`	0.263

Table 1: Regular expressions and their associated precision for questions of the form *"When was X born?"*.

against the text and C_c^P, which counts the number of matches which had `AnSWeR` as the extracted answer.

6. After a pattern, P, has been matched against every sentence in the generated document if C_c^P is less than five then it is discarded otherwise the precision of the pattern is calculated as C_c^P/C_a^P and the pattern is retained if its precision is greater than 0.1[4].

Using this method to produce a list of analysed patterns for the question type *"When was X born?"* gives regular expressions such as those in Table 1, which are now generic and could be applied to any other question of the same type.

2.2 Using Text Patterns to Find Answers

Using these regular expressions to find answers to questions is extremely simple. Firstly the question term is extracted from the question and submitted as a query to Google. Each document returned by Google then has the question term replaced by `AnCHoR` and those sentences containg `AnCHoR` are retained to create a single document. Each regular expression is then matched against the sentences and for each successful match the token captured by the expression is stored along with the precision of the pattern. When all the regular expressions have been applied to all the sentences, any answers found are sorted based firstly on the precision of the pattern which located them and secondly on the number of times the same answer was found.

3 The Limitations Imposed by Overly Specific Patterns

Most papers which describe systems using surface matching text patterns (Soubbotin and Soubbotin, 2001; Ravichandran and Hovy, 2002), including the current paper, explain the workings of the system through questions of the form *"When was X born?"* often using *"When was Mozart born?"* as a specific example. One or more of the analysed patterns are usually capable of extracting the answer from text such as: *"Mozart (1756-1791) was a musical genius"*. Indeed extracting the answer from sentences of this form is a sensible thing to do, due to the fact that this formulation of the answer is both precise and commonly occurring in unstructured text. This example, however, exposes a serious problem with the approach.

A similar question, which could also be answered from the example sentence, is *"When did Mozart die?"*. The problem is that generating patterns for this type of question using multiple examples will not lead to a simple generic pattern that can be applied to other questions. The extracted patterns, for a single example, will include the year of birth, such as:

```
AnCHoR (1756 - AnSWeR
AnCHoR (1756 - AnSWeR )
```

When these patterns are analysed against a second set of question-answer pairs, and their precision is calculated, they will most likely be discarded[5], due to the presence of a specific year of birth. This problem does not occur when generating patterns for questions of the form *"When was X born?"* as multiple patterns will be produced some of which contain the date of death and some of which do not, simply because the date of death usually appears after the year of birth in the answer phrases.

More generally any acquired pattern must consist of three components 1) the `AnCHoR` tag (which gets initialised as the question-specific anchor, e.g. Mozart), 2) the `AnSWeR` regular expression, and 3) literal text occurring between 1) and 2). In the basic text pattern learning approach of Section 2, component 3) cannot be generalised, i.e. cannot be a regular expression containing meta-characters, and hence can only match itself.

There are other patterns, which could be extracted, for questions of the form *"When did X*

4. These cut-off values were adopted based on empirical observations made during development.

5. An exception, specific to this example, would be if the same year of birth appeared in the question sets used for both inducing the answer patterns and assigning precisions to the answer patterns.

die?". For example:

```
AnCHoR died in AnSWeR
AnCHoR was killed in AnSWeR
```

These patterns, however, are not as precise as those possible for *"When was X born?"* (for example `AnSWeR` could easily be a location instead of a date). In an experiment (documented in Section 5) our system failed to generate any patterns for the question type *"When did X die?"*.

4 Generalising the Answer Patterns

It is clear that for this style of question answering to be as accurate as possible a way needs to be found to generate as precise a set of patterns as possible for each question type. As we have already noted one problem is that in certain formulations of the answer phrase words specific to the question appear between `AnCHoR` and `AnSWeR`. Many of these words are dates, names and locations in fact exactly the words that can be recognised using the well understood natural language techniques of gazetteers and named entity taggers. The solution employed by our system is therefore a combination of the question answering system described in Section 2 and a gazetteer and a named entity tagger [6].

The approach taken to incorporate these NLP techniques is to substitute the text marked as a named entity by a tag representing its type, hence dates become `DatE`, locations become `LocatioN`, etc. This replacement is carried out after the question and answer text have been replaced with `AnCHoR` and `AnSWeR` respectively but before any other processing is carried out. This is the only change to the algorithms for inducing and assigning precisions to the answer patterns.

When these new patterns are used to answer questions extra work is, however, required as it is possible that an answer found by a pattern may in fact be, or may include, a named entity tag. When using the patterns not only do we replace named entities with a tag but also store the original text so that if an answer contains a tag it can be expanded back to the original text.

As was previously mentioned, the standard implementation failed to create any patterns for the questions *"When did X die?"* this extended implementation, however, produces regular expressions such as those in Table 2.

Regular Expression	***Precision***
`AnCHoR \( DatE - ([^ ]+) \) .`	1.000
`AnCHoR \( DatE - ([^ ]+) \)`	1.000
`AnCHoR DatE - ([^ ]+)`	0.889

Table 2: Regular expressions, augmented with named entity tags, and the associated precisions for questions of the form *"When did X die?"*.

Regular Expression	***Precision***
`AnCHoR \( ([^ ]+) -`	0.941
`AnCHoR \( ([^ ]+) - DatE \) .`	0.941
`AnCHoR \( ([^ ]+) - DatE \)`	0.941
`AnCHoR \( ([^ ]+)`	0.600
`AnCHoR ([^ ]+) - DatE`	0.556
`AnCHoR ([^ ]+) -`	0.263

Table 3: A selection of regular expressions, augmented with named entity tags, and the associated precisions for questions of the form *"When was X born?"*.

It is clear, from these patterns, that incorporating the NLP techniques allowed us to extract exactly the type of patterns we extended the system to handle.

This extended system can also be used to generate a new set of patterns for questions of the form *"When was X born?"*, a selection of these can be seen in Table 3.

5 Results

A set of experiments was carried out to see the effect of extending the patterns in the way suggested in the previous section. The question sets used for the experiments consisted of one hundred and forty examples, divided into three groups: twenty examples for inducing the patterns, twenty for assigning precisions to the patterns and one hundred over which to test the patterns. A selection of the analysed patterns have already been presented in Tables 1, 2 and 3.

Results are given for four experiments the combination of the original and extended systems over

6. The gazetteer and named entity tagger used in these experiments are slightly modified versions of those which are included as part of the GATE 2 framework (Cunningham et al., 2002), available from http://gate.ac.uk.

System Number	*% Correctly Answered*	*MRR Score*	*Confidence Weighted*
1	52%	0.52	0.837
2	53%	0.52	0.843
3	0%	0.00	0.000
4	53%	0.53	0.852

Table 4: Results of using both the original and extended systems.

the two different question types. The experiments are as follows:

1. The original system answering questions of the form *"When was X born?"*.

2. The extended system answering questions of the form *"When was X born?"*.

3. The original system answering questions of the form *"When did X die?"*.

4. The extended system answering questions of the form *"When did X die?"*.

The results of these experiments can be seen in Table 4. The mean reciprocal rank (MRR) score (Voorhees, 2001) of 0.52 for system 1 is comparable to the results of similar experiments (Ravichandran and Hovy, 2002) over the same question type.

The results show that not only does the extended system allow us to achieve similar results for the questions *"When did X die?"* but also that extending the system in this way had no significant detrimental effects on the performance over question types answerable by the original system and actually produced a higher confidence weighted score (Voorhees, 2002) for all question types. The slight increase in the confidence weighted score is probably due to the greater number of overlapping high precision patterns induced for a specific question type. This leads to the same answer being extracted more often and with a higher precision than in the original system, leading to these answers being ranked higher when the answers for multiple questions are sorted.

6 Discussion and Conclusions

Although the results given in this paper cover only a small number of experiments, they show that the use of a gazetteer and named entity tagger allow the simple pattern matching question answering system to be extended to answer some questions which the original approach could not answer. Furthermore, the performance of the extended system on questions which could already be answered is improved.

Clearly more experimentation is needed before we can claim that this technique solves all the problems associated with overly specific answer patterns. This paper has shown that it successfully handles one specific question type. Experiments were also carried out for the question type *"What is the capital of X?"* in which although the extend system produced better results, than the original system, the improvement was not significant, because in most cases no question-specific text fell between the `AnCHoR` and the `AnSWeR`.

It should be clear, however, that this appraoch can be applied to any question type where question-specific text is likely to occur between the `AnCHoR` and the `AnSWeR`, such as *"When was America discovered?"* which can easily be answered by the text *"In 1492 Columbus discovered America"*, where Columbus needs to be generalised before a sensible pattern could be induced from this answer phrase.

There are other ways in which the text within an answer pattern can be generalised, and we do not claim that our solution is the only way forward, rather that it has been shown to work well over a small set of question types. More work is needed to expand not only the types of question the system can answer but also to test other methods of generalising the surface matching text patterns induced from free text.

References

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 2002. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*.

Sanda Harabagiu, Dan Moldovan, Marius. Paşca, Rada Mihalcea, Mihai Surdeanu, Răzvan Bunescu, Roxana Gîrju, Vasile Rus, and Paul Morărescu. 2000. FALCON: Boosting Knowledge for Answer Engines. In *Proceedings of the 9th Text REtrieval Conference*.

George A. Miller. 1995. WordNet: A Lexical Database. *Communications of the ACM*, 38(11):39–41, November.

Deepak Ravichandran and Eduard Hovy. 2002. Learning Surface Text Patterns for a Question Answering System. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 41–47, Pennsylvania.

M. M. Soubbotin and S. M. Soubbotin. 2001. Patterns of Potential Answer Expressions as Clues to the Right Answers. In *Proceedings of the 10th Text REtrieval Conference*.

E. Ukkonen. 1995. On-line Construction of Suffix Trees. *Algorithmica*, 14(3):249–260.

Ellen M. Voorhees. 2000. Overview of the TREC-9 Question Answering Track. In *Proceedings of the 9th Text REtrieval Conference*.

Ellen M. Voorhees. 2001. Overview of the TREC 2001 Question Answering Track. In *Proceedings of the 10th Text REtrieval Conference*.

Ellen M. Voorhees. 2002. Overview of the TREC 2002 Question Answering Track. In *Proceedings of the 11th Text REtrieval Conference*. Draft version which appeared in the conference notebook.