Event Matching Using the Transitive Closure of Dependency Relations

Daniel M. Bikel and Vittorio Castelli
IBM T. J. Watson Research Center
1101 Kitchawan Road
Yorktown Heights, NY 10598
{dbikel,vittorio}@us.ibm.com

Abstract

This paper describes a novel event-matching strategy using features obtained from the transitive closure of dependency relations. The method yields a model capable of matching events with an F-measure of 66.5%.

1 Introduction

Question answering systems are evolving from their roots as factoid or definitional answering systems to systems capable of answering much more open-ended questions. For example, it is one thing to ask for the birthplace of a person, but it is quite another to ask for all locations visited by a person over a specific period of time.

Queries may contain several types of arguments: person, organization, country, location, etc. By far, however, the most challenging of the argument types are the event or topic arguments, where the argument text can be a noun phrase, a participial verb phrase or an entire indicative clause. For example, the following are all possible event arguments:

- the U.S. invasion of Iraq
- Red Cross admitting Israeli and Palestinian groups
- GM offers buyouts to union employees

In this paper, we describe a method to match an event query argument to the sentences that mention that event. That is, we seek to model $p(s \text{ contains } e \mid s, e)$, where e is a textual description of an event (such as an event argument for a GALE distillation query) and where s is an arbitrary sentence. In the first example above, "the U.S. invasion of Iraq", such a model should produce a very high score for that event description and the sentence "The U.S. invaded Iraq in 2003."

2 Low-level features

As the foregoing implies, we are interested in training a binary classifier, and so we represent each training and test instance in a feature space. Conceptually, our features are of three different varieties. This section describes the first two kinds, which we call "low-level" features, in that they attempt to capture how much of the basic information of an event e is present in a sentence s.

2.1 Lexical features

We employ several types of simple lexical-matching features. These are similar to the "bag-of-words" features common to many IR and question-answering systems. Specifically, we compute the value $\text{overlap}(s, e) = \frac{\mathbf{w}_s \cdot \mathbf{w}_e}{|\mathbf{w}_e|_1}$, where $\mathbf{w}_e$ (resp: $\mathbf{w}_s$) is the $\{0,1\}$-valued word-feature vector for the event (resp: sentence). This value is simply the fraction of distinct words in e that are present in s. We then quantize this fraction into the bins $[0, 0]$, $(0, 0.33]$, $(0.33, 0.66]$, $(0.66, 0.99]$, $(0.99, 1]$, to produce one of five, binary-valued features to indicate whether none, few, some, many or all of the words match.[1]

2.2 Argument analysis and submodels

Since an event or topic most often involves entities of various kinds, we need a method to recognize those entity mentions. For example, in the event "Abdul Halim Khaddam resigns as Vice President of Syria", we have a PERSON mention, an OCCUPATION mention and a GPE (geopolitical entity) mention. We use an information extraction toolkit (Florian et al., 2004) to analyze each event argument. The toolkit performs the following steps: tokenization, part-of-speech tagging, parsing, mention detection, within-document coreference resolution and cross-document coreference resolution. We also apply the toolkit to our entire search corpus.

After determining the entities in an event description, we rely on lower-level binary classifiers, each of which has been trained to match a specific type

[1] Other binnings did not significantly alter the performance of the models we trained, and so we used the above binning strategy for all experiments reported in this paper.

Proceedings of ACL-08: HLT, Short Papers (Companion Volume), pages 145–148,
Columbus, Ohio, USA, June 2008.

of entity. For example, we use a PERSON-matching model to determine if, say, "Abdul Halim Khaddam" from an event description is mentioned in a sentence.[2] We build binary-valued feature functions from the output of our four lower-level classifiers.

3 Dependency relation features

Employing syntactic or dependency relations to aid question answering systems is by no means new (Attardi et al., 2001; Cui et al., 2005; Shen and Klakow, 2006). These approaches all involved various degrees of loose matching of the relations in a query relative to sentences. More recently, Wang et al. (2007) explored the use a formalism called *quasi-synchronous grammar* (Smith and Eisner, 2006) in order to find a more explicit model for matching the set of dependencies, and yet still allow for looseness in the matching.

3.1 The dependency relation

In contrast to previous work using relations, we do not seek to model explicitly a process that transforms one dependency tree to another, nor do we seek to come up with *ad hoc* correlation measures or path similarity measures. Rather, we propose to use features based on the transitive closure of the dependency relation of the event and that of the dependency relation of the sentence. Our aim was to achieve a balance between the specificity of dependency paths and the generality of dependency pairs.

In its most basic form, a *dependency tree* for a sentence $\mathbf{w} = \langle \omega_1, \omega_w, \ldots, \omega_k \rangle$ is a rooted tree $\tau = \langle V, E, r \rangle$, where $V = \{1, \ldots, k\}$, $E = \left\{(i, j) : \omega_i \text{ is the child of } \omega_j\right\}$ and $r \in \{1, \ldots, k\}$: ω_r is the root word. Each element ω_i of our word sequence, rather than being a simple lexical item drawn from a finite vocabulary, will be a complex structure. With each word w_i we associate a part-of-speech tag t_i, a morph (or stem) m_i (which is w_i itself if w_i has no variant), a set of nonterminal labels N_i, a set of synonyms S_i for that word and a canonical mention $cm(i)$. Formally, we let each sequence element be a sextuple $\omega_i = \langle w_i, t_i, m_i, N_i, S_i, cm(i) \rangle$.

[2]This is not as trivial as it might sound: the model must deal with name variants (parts of names, alternate spellings, nicknames) and with metonymic uses of titles ("Mr. President" referring to Bill Clinton or George W. Bush).

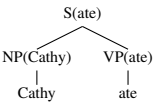

Figure 1: Simple lexicalized tree.

We derive dependency trees from head-lexicalized syntactic parse trees. The set of nonterminal labels associated with each word is the set of labels of the nodes for which that word was the head. For example, in the lexicalized tree in Figure 1, the head word "ate" would be associated with both the nonterminals `S` and `VP`. Also, if a head word is part of an entity mention, then the "canonical" version of that mention is associated with the word, where canonical essentially means the best version of that mention in its coreference chain (produced by our information extraction toolkit), denoted $cm(i)$. In Figure 1, the first word w_1 = `Cathy` would probably be recognized as a PERSON mention, and if the coreference resolver found it to be coreferent with a mention earlier in the same document, say, `Cathy Smith`, then $cm(1)$ = `Cathy Smith`.

3.2 Matching on the transitive closure

Since E represents the child-of dependency relation, let us now consider the transitive closure, E', which is then the descendant-of relation.[3] Our features are computed by examining the overlap between E'_e and E'_s, the descendant-of relation of the event description e and the sentence s, respectively. We use the following, two-tiered strategy.

Let d_e, d_s be elements of E'_e and E'_s, with $d_x.d$ denoting the index of the word that is the descendant in d_x and $d_x.a$ denoting the ancestor. We define the following matching function to match the pair of descendants (or ancestors):

$$\text{match}_d(d_e, d_s) = \\ (m_{d_e.d} = m_{d_s.d}) \vee (cm(d_e.d) = cm(d_s.d)) \quad (1)$$

where match_a is defined analogously for ancestors. That is, $\text{match}_d(d_e, d_s)$ returns `true` if the morph of the descendant of d_e is the same as the morph of the descendant of d_s, *or* if both descendants have canonical mentions with an exact string match; the

[3]We remove all edges (i, j) from E' where either w_i or w_j is a stop word.

function returns `false` otherwise, and match_a is defined analogously for the pair of ancestors. Thus, the pair of functions $\text{match}_d, \text{match}_a$ are "morph or mention" matchers. We can now define our main matching function in terms of match_d and match_a:

$$\text{match}(d_e, d_s) = \text{match}_d(d_e, d_s) \wedge \text{match}_a(d_e, d_s). \quad (2)$$

Informally, $\text{match}(d_e, d_s)$ returns `true` if the pair of descendants have a "morph-or-mention" match *and* if the pair of ancestors have a "morph-or-mention" match. When $\text{match}(d_e, d_s) =$ `true`, we use "morph-or-mention" matching features.

If $\text{match}(d_e, d_s) =$ `false` we then attempt to perform matching based on synonyms of the words involved in the two dependencies (the "second tier" of our two-tiered strategy). Recall that $S_{d_e.d}$ is the set of synonyms for the word at index $d_e.d$. Since we do not perform word sense disambiguation, $S_{d_e.d}$ is the union of all possible synsets for $w_{d_e.d}$. We then define the following function for determining if two dependency pairs match at the synonym level:

$$\text{synmatch}(d_e, d_s) = \quad (3)$$
$$(S_{d_e.d} \cap S_{d_s.d} \neq \emptyset) \wedge (S_{d_e.a} \cap S_{d_s.a} \neq \emptyset).$$

This function returns `true` iff the pair of descendants share at least one synonym and the pair of ancestors share at least one synonym. If there is a synonym match, we use synonym-matching features.

3.3 Dependency matching features

The same sorts of features are produced whether there is a "morph-or-mention" match or a synonym match; however, we still distinguish the two types of features, so that the model may learn different weights according to what type of matching happened. The two matching situations each produce four types of features. Figure 2 shows these four types of features using the event of "Abdul Halim Khaddam resigns as Vice President of Syria" and the sentence "The resignation of Khaddam was abrupt" as an example. In particular, the "depth" features attempt to capture the "importance" the dependency match, as measured by the depth of the ancestor in the event dependency tree.

We have one additional type of feature: we compute the following kernel function on the two sets of dependencies E'_e and E'_s and create features based on quantizing the value:

$$K(E'_e, E'_s) = \quad (4)$$
$$\sum_{(d_e,d_s)\in E'_e \times E'_s \,:\, \text{match}(d_e,d_s)} (\Delta(d_e) \cdot \Delta(d_s))^{-1},$$

$\Delta((i, j))$ being the path distance in τ from node i to j.

4 Data and experiments

We created 159 queries to test this model framework. We adapted a publicly-available search engine (citation omitted) to retrieve documents automatically from the GALE corpus likely to be relevant to the event queries, and then used a set of simple heuristics—a subset of the low-level features described in §2—to retrieve sentences that were more likely than not to be relevant. We then had our most experienced annotator annotate sentences with five possible tags: `relevant`, `irrelevant`, `relevant-in-context`, `irrelevant-in-context` and `garbage` (to deal with sentences that were unintelligible "word salad").[4] Crucially, the annotation guidelines for this task were that an event had to be explicitly mentioned in a sentence in order for that sentence to be tagged `relevant`.

We separated the data roughly into an 80/10/10 split for training, devtest and test. We then trained our event-matching model solely on the examples marked `relevant` or `irrelevant`, of which there were 3546 instances. For all the experiments reported, we tested on our development test set, which comprised 465 instances that had been marked `relevant` or `irrelevant`.

We trained the kernel version of an averaged perceptron model (Freund and Schapire, 1999), using a polynomial kernel with degree 4 and additive term 1. As a baseline, we trained and tested a model using only the lexical-matching features. We then trained and tested models using only the low-level features and all features. Figure 3 shows the performance statistics of all three models, and Figure 4 shows the ROC curves of these models. Clearly, the dependency features help; at our normal operating point of 0, F-measure rises from 62.2 to 66.5. Looking solely

[4] The `*-in-context` tags were to be able to re-use the data for an upstream system capable of handling the GALE distillation query type "list facts about [event]".

Feature type	Example	Comment
Morph bigram	`x-resign-Khaddam`	Sparse, but helpful.
Tag bigram	`x-VBZ-NNP`	
Nonterminal	`x-VP-NP`	All pairs from $N_i \times N_j$ for $(i, j) \in E'_e$.
Depth	`x-eventArgHeadDepth=0`	Depth is 0 because "resigns" is root of event.

Figure 2: Types of dependency features. Example features are for e = "Abdul Halim Khaddam resigns as Vice President of Syria" and s = "The resignation of Khaddam was abrupt." In example features, `x` $\in$ {`m`, `s`}, depending on whether the dependency match was due to "morph-or-mention" matching or synonym matching.

Model	R	P	F
lex	36.6	76.3	49.5
low-level	63.9	60.5	62.2
all	**69.1**	**64.1**	**66.5**

Figure 3: Performance of models.

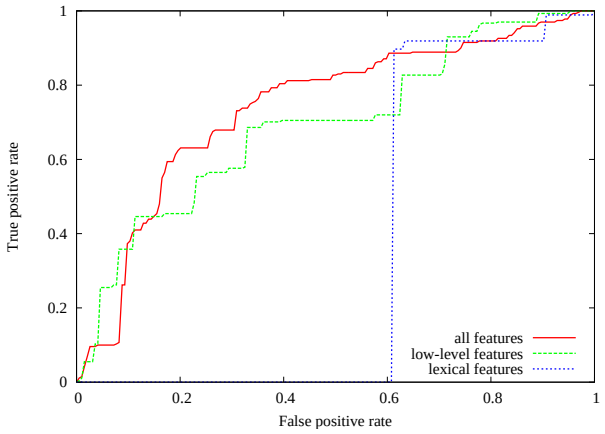

Figure 4: ROC curves of model with only low-level features vs. model with all features.

at pairs of predictions, McNemar's test reveals differences ($p \ll 0.05$) between the predictions of the baseline model and the other two models, but not between those of the low-level model and the model trained with all features.

5 Discussion

There have been several efforts to incorporate dependency information into a question-answering system. These have attempted to define either *ad hoc* similarity measures or a tree transformation process, whose parameters must be learned. By using the transitive closure of the dependency relation, we believe that—especially in the face of a small data set—we have struck a balance between the representative power of dependencies and the need to remain agnostic with respect to similarity measures or formalisms; we merely let the features speak for themselves and have the training procedure of a robust classifier learn the appropriate weights.

Acknowledgements

This work supported by DARPA grant HR0011-06-02-0001. Special thanks to Radu Florian and Jeffrey Sorensen for their helpful comments.

References

Giuseppe Attardi, Antonio Cisternino, Francesco Formica, Maria Simi, Alessandro Tommasi, Ellen M. Voorhees, and D. K. Harman. 2001. Selectively using relations to improve precision in question answering. In *TREC-10*, Gaithersburg, Maryland.

Hang Cui, Renxu Sun, Keya Li, Min-Yen Kan, and Tat-Seng Chua. 2005. Question answering passage retrieval using dependency relations. In *SIGIR 2005*, Salvador, Brazil, August.

Radu Florian, Hani Hassan, Abraham Ittycheriah, Hongyan Jing, Nanda Kambhatla, Xiaoqiang Luo, Nicholas Nicolov, and Salim Roukos. 2004. A statistical model for multilingual entity detection and tracking. In *HLT-NAACL 2004*, pages 1–8.

Yoav Freund and Robert E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

Dan Shen and Dietrich Klakow. 2006. Exploring correlation of dependency relation paths for answer extraction. In *COLING-ACL 2006*, Sydney, Australia.

David A. Smith and Jason Eisner. 2006. Quasi-synchronous grammars: Alignment by soft projection of syntactic dependencies. In *HLT-NAACL Workshop on Statistical Machine Translation*, pages 23–30.

Mengqiu Wang, Noah A. Smith, and Teruko Mitamura. 2007. What is the Jeopardy model? a quasi-synchronous grammar for QA. In *EMNLP-CoNLL 2007*, pages 22–32.