BUDAPESTACAD at TAC 2010

Dávid Nemeskey, Gábor Recski, Attila Zséder, and András Kornai
Computer and Automation Research Institute, Hungarian Academy of Sciences
{ndavid,recski,zseder,kornai}@sztaki.hu

1 Introduction

The following paper will describe the systems we have created for the Knowledge Base Population (KBP) and Recognizing Textual Entailment (RTE) tracks of TAC. Sections 2 and 3 will describe a set of tools used to perform the entity-linking and slot-filling tasks of the KBP track respectively. Section 4 will give an overview of a complex tool for recognizing textual entailment. Each section describes the various modules created, provides an overview of the process of performing the task at hand, and discusses the results obtained.

2 Entity Linking

The Entity Linking task requires that the system, given a name-string, determines which entity, if any, is being referred to in a pre-defined knowledge base. Each query also contains reference to a document from a collection of newswire articles and blog entries associated with it, which can be used to disambiguate the name-string. The knowledge base was a subset of a Wikipedia snapshot taken in 2008.

Our system uses a setup not unlike a typical question answering system. The knowledge base was loaded into an Information Retrieval engine, after which we retrieved the top 100 entities for each query. Next, a validation step selected the entity to return, or NIL, if no suitable candidates were found.

2.1 Information Retrieval

We used SZTAKI's information retrieval engine (Daróczy et al., 2009). In three sets of experiments, we loaded into the engine first only the titles, next the titles and infoboxes, and finally the entire documents. Our goal was to maximize recall and let the validation step select the entity to return. The results of our experiments are listed in Table 1. Perhaps not surprisingly, the best recall was achieved by the last setup.

We assembled the queries for our IR engine in the following way: we included the name-strings, as well as the words surrounding them in the supporting documents in a 5-term radius, albeit with a lower weight. Furthermore, we examined the queries from 2009 and identified two query types that pose difficulties for IR engines: spelling variations and abbreviations.

Sections indexed	**r@1**	**r@5**	**r@10**	**r@50**	**r@100**
Title only	N/A	N/A	N/A	N/A	0.70
Title and infoboxes	0.29	0.50	0.60	0.79	0.84
Entire document	0.57	0.79	0.84	0.92	0.94

Table 1: Recall values at various cut-off ranks

Luckily, the first problem was tackled by the built-in spell corrector of the IR engine. We handled abbreviations by building a dictionary of them. We parsed all documents for named entities (NEs) and created abbreviation candidates from them. If an abbreviation candidate actually occured in the same document as the NE, it was saved to the dictionary along with the entity as its expansion. Finally, we extended the affected queries with all possible expansions of the abbreviations included.

2.2 Entity Validation

The validation phase was rather simple. We took the top n candidates from the IR list and compared their NE tag (PER, ORG or LOC) to that of the name-string in the query. For some entities, tags were readily available in the knowledge base — for the others, as well as the name-strings, we used our in-house NER tagger (Varga and Simon, 2006). The first entity where the tags matched was returned as the answer. In case no match was found, we returned NIL.

Table 2 summarizes our results. As can be seen, our precision is below the median, which we contribute to two reasons: On the one hand, the accuracy of NER tagging was low. More importantly, our validation method was simplistic, and by examining only the top-ranked documents, it could not take advantage of the high recall achieved by the IR step.

Run type	n	Our run	Median	Highest
Full run	1	0.5018	0.6836	0.8680
	3	0.3378		
No-text run	1	0.4396	0.6347	0.7791
	3	0.2760		

Table 2: Micro-averaged precision achieved by our official runs

3 Slot-filling

The slot-filling task involves the automatic discovery of specific types of data regarding some entity based on a collection of text which may or may not contain the information neccessary to fill the database slots.

3.1 System architecture

As a first approach to the task at hand we proceeded to reimplement the basic ideas of the system described in (Agirre et al., 2009).

First, we obtained entity-slot-value triplets from the knowledge base (infoboxes). To do this, we needed a mapping from the noisy infobox field names to the proper slot names. We did this manually and non-exhaustively from the most frequent field names, only in order to obtain sufficient training data.

Next, we searched for occurences of the previously obtained data in the tagged documents near entities that they belonged to. To be able to train our classifiers, we needed negative examples as well, so for every positive answer we collected 3 other ones with the same slot type.

Next, we trained a maximal entropy classifier for each slot. Features were extracted from the context of the entities and the positive and negative examples of their facts. We annotated each word with a tag which describes whether the words referring to the entity are situated to the left or right of the given word. Features for each word consisted of n-gram sequences of such tags in their context.

When evaluating the questions, we first filtered the slots we needed using given ignore lists and data already present in the KB. We also searched for occurences of the given entities, but only those which matched the entity types required by the slots. Finally, we classified all data with the models trained before.

3.2 Results and discussion

From the total of 1034 slots to fill we submitted some response to 901, of which 43 was correct. The scores are shown in the Table 3.

As can be seen, we were not able to replicate the performance of the Stanford system. One of the reasons is that our manual work of mapping noisy infobox names to slot names was very poor. Moreover, one serious shortcoming of our system is that our NER tagger is unable to find dates and we created a small software for this purpose. In addition, we did not use any geographical database at this time, therefore we did not distinguish between e.g. cities and countries.

Precision	0.041
Recall	0.047
F1	0.044

Table 3: Slot filling results

4 RTE

The textual entailment task requires a system to make binary decisions about whether the propositional content of a given sentence (the hypothesis) is supported by another (the support sentence). The system has access to the context in which the support sentence appears, the hypothesis sentence is arbitrary.

4.1 System architecture

Our solution to the RTE task is based on the extraction of primitive semantic relations from both the hypothesis and support sentence and using them to compare the propositional content of each. In order to do this we first parsed the data using the Charniak parser (Charniak, 2000). Then we implemented a tool which performs a traversal over each parse tree and creates triplets of the form *(predicate, argument1, argument2)* based on subtrees with specific configurations. We will now describe the basic types of triplets extracted.

The first type of triplets consist of a verb and its arguments. We created one such triplet for each VP within a sentence providing we could find at least one argument. The second slot of the triplet is occupied by the external argument (the subject of the verb in active constructions), which we obtained by simply taking the head of the last noun phrase preceding the VP in question. The third slot of the triplet contains the internal argument, the head of the NP within the given phrase which follows the verb. Thus the subtree in Figure 1 will yield the triplet `(loves, John, Mary)`.

The next group of triplets contains those which encode the relationship between a noun and its modifiers. These triplets begin with the abstract predicate `IS` which is then followed by the noun and its modifier. From the NP in Figure 2 we create the triplets `(IS, wolf, bad)` and `(IS, wolf, big)`.

In addition to generating these triplets, we ran a NER tagger on all sentences and the BART coreference solver (Versley et al., 2008) on support sentences and their context. After this, we added new triplets gained from these outputs. Based on the assumption that having two different people with the same last name in a sentence pair

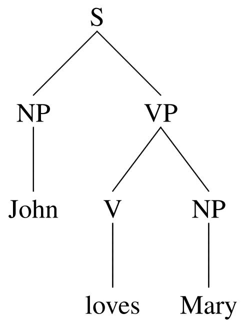

Figure 1: `(loves, John, Mary)`

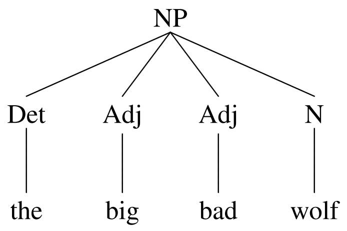

Figure 2: `(IS, wolf, bad)(IS, wolf, big)`

is a rare event, we also added simplified NER triplets that only contain last names. For example, from a sentence like „Al Bundy loves Peggy", we added `(NER, Al Bundy, PER)`, `(SIMP, Al Bundy, Bundy)` and `(loves, Bundy, Peggy)`.

On the triplets that only contained one word in a field (which is not a NER), we used WordNet (Miller, 1995) to add new triplets based on its synsets. We skipped fields with at least two words to avoid too many generated triplets. After all these steps, every sentence had about 20-500 triplets.

We used a simple machine learning approach to recognize entailment between the hypothesis-candidate sentence pairs. The following features were considered: number of triplets in the hypothesis and the candidates (ht and ct, respectively), the number of triplets that occur in both (hct) and the ratios $hr = hct/ht$ and $cr = hct/ct$. hct was computed in two different ways. The first method demanded exact lexical matching in all fields of the triplets, while the second method allowed partial matches; that is, two triplets were considered matching if each of their corresponding fields had at least one word in common.

We built decision stump classifiers on all features and selected the one with the highest information gain on the training set. hr turned out to be the most important attribute in each of our experiments. Since it performed better on the training set, we employed partial matching in our official run.

4.2 Results and discussion

The results of our official and ablation runs are presented in Table 4, along with the optimal hr values. Much to our surprise, most of the ablation runs outperformed the official one. We believe this is caused by overlearning, and take the high variance of hr as a proof.

Run	hr	**Prec.**	**Recall**	**F1**
Official run	0.44	12.90	32.06	18.40
No partial matching	0.44	13.35	31.22	18.71
No name norm.	0.44	13.49	25.93	17.75
No NER	0.01	13.59	35.34	19.63
No WordNet	0.022	14.51	19.74	19.51

Table 4: Micro-averaged precision achieved on the official and ablation runs

References

E. Agirre, A.X. Chang, D.S. Jurafsky, C.D. Manning, V.I. Spitkovsky, and E. Yeh. 2009. Stanford-UBC at TAC-KBP. In *Proceedings of Test Analysis Conference 2009 (TAC 09)*.

E. Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 132–139. Morgan Kaufmann Publishers Inc.

Bálint Daróczy, Zsolt Fekete, Mátyás Brendel, Simon Rácz, András Benczúr, Dávid Siklósi, and Attila Pereszlényi. 2009. Cross-modal image retrieval with parameter tuning. In Carol Peters, Danilo Giampiccol, Nicola Ferro, Vivien Petras, Julio Gonzalo, Anselmo Peñas, Thomas Deselaers, Thomas Mandl, Gareth Jones, and Nikko Kurimo, editors, *Evaluating Systems for Multilingual and Multimodal Information Access – 9th Workshop of the Cross-Language Evaluation Forum*, Lecture Notes in Computer Science, Aarhus, Denmark, September.

G.A. Miller. 1995. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41.

D. Varga and E. Simon. 2006. Hungarian named entity recognition with a maximum entropy approach. *Acta Cybernetica*, 16:293–301.

Y. Versley, S.P. Ponzetto, M. Poesio, V. Eidelman, A. Jern, J. Smith, X. Yang, and A. Moschitti. 2008. BART: A modular toolkit for coreference resolution. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Demo Session*, pages 9–12. Association for Computational Linguistics.