The University of Amsterdam at Senseval-3: Semantic Roles and Logic Forms

David Ahn Sisay Fissaha Valentin Jijkoun Maarten de Rijke
Informatics Institute, University of Amsterdam
Kruislaan 403
1098 SJ Amsterdam
The Netherlands
{ahn,sfissaha,jijkoun,mdr}@science.uva.nl

Abstract

We describe our participation in two of the tasks organized within Senseval-3: Automatic Labeling of Semantic Roles and Identification of Logic Forms in English.

1 Introduction

This year (2004), Senseval, a well-established forum for the evaluation and comparison of word sense disambiguation (WSD) systems, introduced two tasks aimed at building semantic representations of natural language sentences. One task, Automatic Labeling of Semantic Roles (SR), takes as its theoretical foundation Frame Semantics (Fillmore, 1977) and uses FrameNet (Johnson et al., 2003) as a data resource for evaluation and system development. The definition of the task is simple: given a natural language sentence and a *target* word in the sentence, find other fragments (continuous word sequences) of the sentence that correspond to elements of the semantic frame, that is, that serve as arguments of the predicate introduced by the target word.

For this task, the systems receive a sentence, a target word, and a semantic frame (one target word may belong to multiple frames; hence, for real-world applications, a preliminary WSD step might be needed to select an appropriate frame). The output of a system is a list of frame elements, with their names and character positions in the sentence. The evaluation of the SR task is based on precision and recall. For this year's task, the organizers chose 40 frames from FrameNet 1.1, with 32,560 annnotated sentences, 8,002 of which formed the test set.

The second task, Identification of Logic Forms in English (LF), is based on the LF formalism described in (Rus, 2002). The LF formalism is a simple logical form language for natural language semantics with only predicates and variables; there is no quantification or negation, and atomic predications are implicitly conjoined. Predicates correspond directly to words and are composed of the base form of the word, the part of speech tag, and a sense number (corresponding to the WordNet sense of the word as used). For the task, the system is given sentences and must produce LFs. Word sense disambiguation is not part of the task, so the predicates need not specify WordNet senses. System evaluation is based on precision and recall of predicates and predicates together with all their arguments as compared to a gold standard.

2 Syntactic Processing

For both tasks, SR and LF, the core of our systems was the syntactic analysis module described in detail in (Jijkoun and de Rijke, 2004). We only have space here to give a short overview of the module.

Every sentence was part-of-speech tagged using a maximum entropy tagger (Ratnaparkhi, 1996) and parsed using a state-of-the-art wide coverage phrase structure parser (Collins, 1999). Both the tagger and the parser are trained on the Penn Treebank Wall Street Journal Corpus (WSJ in the rest of this paper) and thus produce structures similar to those in the Penn Treebank. Unfortunately, the parser does not deliver some of the information available in WSJ that is potentially useful for our two applications: Penn functional tags (e.g., *subject*, *temporal*, *closely related*, *logical subject in passive*) and non-local dependencies (e.g., subject and object control, argument extraction in relative clauses). Our syntactic module tries to compensate for this and make this information explicit in the resulting syntactic analyses.

As a first step, we converted phrase trees produced by the parser to dependency structures, by detecting heads of constituents and then propagating the lexical head information up the syntactic tree, similarly to (Collins, 1999). The resulting dependency structures were labeled with dependency labels derived from corresponding Penn phrase labels: e.g., a verb phrase (VP) modified by a prepositional phrase (PP) resulted in a dependency with label 'VP|PP'.

Then, the information available in the WSJ (func-

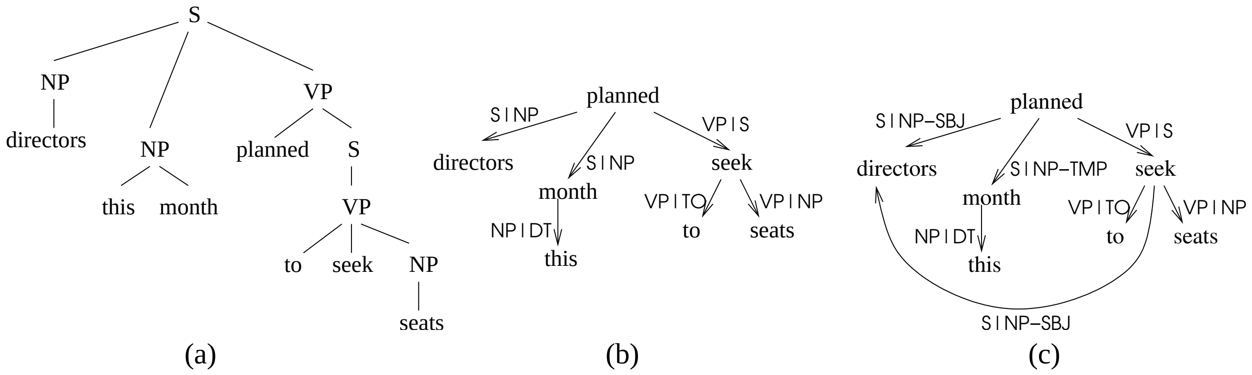

Figure 1: Stages of the syntactic processing: (a) the parser's output, (b) the result of conversion to a dependency structure, (c) final output of our syntactic module

tional tags, non-local dependencies) was added to dependency structures using Memory-Based Learning (Daelemans et al., 2003): we trained the learner to change dependency labels, or add new nodes or arcs to dependency structures. Trained and tested on WSJ, our system achieves state-of-the-art performance for recovery of Penn functional tags and non-local dependencies (Jijkoun and de Rijke, 2004).

Figure 1 shows three stages of the syntactic analysis of the sentence *Directors this month planned to seek seats* (a simplified actual sentence from WSJ): (a) the phrase structure tree produced by Collins' parser, (b) the phrase structure tree converted to a dependency structure and (c) the transformed dependency structure with added functional tags and a non-local dependency—the final output of our syntactic module. Dependencies are shown as arcs from heads to dependents.

3 Automatic Labeling of Semantic Roles

For the SR task, we applied a method very similar to the one used in (Jijkoun and de Rijke, 2004) for recovering syntactic structures and somewhat similar to the first method for automatic semantic role identification described in (Gildea and Jurafsky, 2002). Essentially, our method consists of extracting possible syntactic patterns (paths in syntactic dependency structures), introducing semantic relations from a training corpus, and then using a machine learning classifier to predict which syntactic paths correspond to which frame elements.

Our main assumption was that frame elements, as annotated in FrameNet, correspond directly to constituents (constituents being complete subtrees of dependency structures). Similarly to (Gildea and Jurafsky, 2002), our own evaluation showed that about 15% of frame elements in FrameNet 1.1 do not correspond to constituents, even when applying some straighforward heuristics (see below) to compensate for this mismatch. This observation puts an upper bound of around 85% on the accuracy of our system (with strict evaluation, i.e., if frame element boundaries must match the gold standard exactly). Note, though, that these 15% of "erroneous" constituents also include parsing errors.

Since the core of our SR system operates on words, constituents, and dependencies, two important steps are the conversion of FrameNet elements (continuous sequences of characters) into head words of constituents, and vice versa. The conversion of FrameNet elements is straightforward: we take the head of a frame element to be the word that dominates the most words of this element in the dependency graph of the sentence. In the other direction, when converting a subgraph of a dependency graph dominated by a word w into a continuous sequence of words, we take all (i.e., not only immediate) dependents of w, ignoring non-local dependencies, unless w is the target word of the sentence, in which case we take the word w alone. This latter heuristic helps us to handle cases when a noun target word is a semantic argument of itself. Several other simple heristics were also found helpful: e.g., if the result of the conversion of a constituent to a word sequence contains the target word, we take only the words to the right of the target word.

With this conversion between frame elements and constituents, the rest of our system only needs to operate on words and labeled dependencies.

3.1 Training: the major steps

First, we extract from the training corpus (dependency-parsed FrameNet sentences, with words marked as targets and frame elements) all shortest undirected paths in dependency graphs that connect target words with their semantic arguments. In this way, we collect all "interesting" syntactic paths from the training corpus.

In the second step, for all extracted syntactic paths and again for all training sentences, we extract all *occurences* of the paths (i.e., paths, starting from a target word, that actually exist in the dependency graph), recording for each such occurrence whether it connects a target word to one of its semantic arguments. For performance reasons, we consider for each target word only syntactic paths extracted from sentences annotated with respect to the same frame, and we ignore all paths of length more than 3.

For every extracted occurrence, we record the features describing the occurrence of a path in more detail: the frame name, the path itself, the words along the path (including the target word and the possible head of a frame element—first and last node of the path, respectively), their POS tags and semantic classes. For nouns, the semantic class of a word is defined as the hypernym of the first sense of the noun in WordNet, one of 19 manually selected terms (*animal*, *person*, *social group*, *clothes*, *feeling*, *property*, *phenomenon*, etc.) For lexical adverbs and prepositions, the semantic class is one of the 6 clusters obtained automatically using the K-mean clustering algorithm on data extracted from FrameNet. Examples of the clusters are: (*abruptly*, *ironically*, *slowly*, ...), (*above*, *beneath*, *inside*, ...), (*entirely*, *enough*, *better*, ...). The list of WordNet hypernyms and the number of clusters were chosen experimentally. We also added features describing the subcategorization frame of the target word; this information is straightforwardly extracted from the dependency graph. In total, the system used 22 features.

The set of path occurrences obtained in the second step, with all the extracted features, is a pool of positive and negative examples of whether certain syntactic patterns correspond to any semantic arguments. The pool is used as an instance base to train TiMBL, a memory-based learner (Daelemans et al., 2003), to predict whether the endpoint of a syntactic path starting at a target word corresponds to a semantic argument, and if so, what its name is.

We chose TiMBL for this task because we had previously found that it deals successfully with complex feature spaces and data sparseness (in our case, in the presence of many lexical features) (Jijkoun and de Rijke, 2004). Moreover, TiMBL is very flexible and implements many variants of the basic k-nearest neighbor algorithm. We found that tuning various parameters (the number of neighbors, weighting and voting schemes) made substantial differences in the performance of our system.

3.2 Applying the system

Once the training is complete, the system can be applied to new sentences (with the indicated target word and its frame) as follows. A sentence is parsed and its dependency structure is built, as described in Section 2. All occurences of “interesting” syntactic paths are extracted, along with their features as described above. The resulting feature vectors are fed to TiMBL to determine whether the endpoints of the syntactic paths correspond to semantic arguments of the target word. For the path occurences classified positively, the constituents of their endpoints are converted to continuous word sequences, as described earlier; in this case the system has detected a frame element.

3.3 Results

During the development of our system, we used only the 24,558 sentences from FrameNet set aside for training by the SR task organizers. To tune the system, this corpus was randomly split into training and development sets (70% and 30%, resp.), evenly for all target words. The official test set (8002 sentences) was used only once to produce the submitted run, with the whole training set (24,558 sentences) used for training.

We submitted one run of the system (with identification of both element boundaries and element names). Our official scores are: precision 86.9%, recall 75.2% and overlap 84.7%. Our own evaluation of the submitted run with the *strict* measures, i.e., an element is considered correct only if both its name and boundaries match the gold standard, gave precision 73.5% and recall 63.6%.

4 Logic Forms

4.1 Method

For the LF task, it was straightforward to turn dependency structures into LFs. Since the LF formalism does not attempt to represent the more subtle aspects of semantics, such as quantification, intensionality, modality, or temporality (Rus, 2002), the primary information encoded in a LF is based on argument structure, which is already well captured by the dependency parses. Our LF generator traverses the dependency structure, turning POS-tagged lexical items into LF predicates, creating referential variables for nouns and verbs, and using dependency labels to order the arguments for each predicate. We make one change to the dependency graphs originally produced by the parser. Instead of taking coordinators, such as *and*, to modify the constituents they coordinate, we take the coordinated constituents to be arguments of the coordinator.

Our LF generator builds a labeled directed graph from a dependency structure and traverses this graph depth-first. In general, a well-formed dependency graph has exactly one root node, which corresponds to the main verb of the sentence. Sentences with multiple independent clauses may have one root per clause. The generator begins traversing the graph at one of these root nodes; if there is more than one, it completes traversal of the subgraph connected to the first node before going on to the next node.

The first step in processing a node—producing an LF predicate from the node's lexical item—is taken care of in the graph-building stage. We use a base form dictionary to get the base form of the lexical item and a simple mapping of Penn Treebank tags into 'n', 'v', 'a', and 'r' to get the suffix. For words that are not tagged as nouns, verbs, adjectives, or adverbs, the LF predicate is simply the word itself.

As the graph is traversed, the processing of a node depends on its type. The greatest amount of processing is required for a node corresponding to a verb. First, a fresh referential variable is generated as the event argument of the verbal predication. The out-edges are then searched for nodes to process. Since the order of arguments in an LF predication is important and some sentence constitutents are ignored for the purposes of LF, the out-edges are chosen in order by label: first particles ('VP|PRT'), then arguments ('S|NP-SBJ', 'VP|NP', etc.), and finally adjuncts. We attempt to follow the argument order implicit in the description of LF given in (Rus, 2002), and as the formalism requires, we ignore auxiliary verbs and negation. The processing of each of these arguments or adjuncts is handled recursively and returns a set of predications. For modifiers, the event variable also has to be passed down. For referential arguments and adjuncts, a referential variable also is returned to serve as an argument for the verb's LF predicate. Once all the arguments and adjuncts have been processed, a new predication is generated, in which the verb's LF predicate is applied to the event variable and the recursively generated referential variables. This new predication, along with the recursively generated ones, is returned.

The processing of a nominal node proceeds similarly. A fresh referential variable is generated—since determiners are ignored in the LF formalism, it is simply assumed that all noun phrases correspond to a (possibly composite) individual. Out-edges are examined for modifiers and recursively processed. Both the referential variable and the set of new predications are returned. Noun compounds introduce some additional complexity; each modifying noun introduces two additional variables, one for the modifying noun and one for composite individual realizing the compound. This latter variable then replaces the referential variable for the head noun.

Processing of other types of nodes proceeds in a similar fashion. For modifiers such as adjectives, adverbs, and prepositional phrases, a variable (corresponding to the individual or event being modified) is passed in, and the LF predicate of the node is applied to this variable, rather than to a fresh variable. In the case of prepositional phrases, the predicate is applied to this variable and to the variable corresponding to the object of the preposition, which must be processed, as well. The latter variable is then returned along with the new predications. For other modifiers, just the predications are returned.

4.2 Development and results

The rules for handling dependency labels were written by hand. Of the roughly 1100 dependency labels that the parser assigns (see Section 2), our system handles 45 labels, all of which fall within the most frequent 135 labels. About 50 of these 135 labels are dependencies that can be ignored in the generation of LFs (labels involving punctuation, determiners, auxiliary verbs, etc.); of the remaining 85 labels, the 45 labels handled were chosen to provide reasonable coverage over the sample corpus provided by the task organizers. Extending the system is straightforward; to handle a dependency label linking two node types, a rule matching the label and invoking the dependent node handler is added to the head node handler.

On the sample corpus of 50 sentences to which our system was tuned, predicate identification, compared to the provided LFs, including POS-tags, was performed with 89.1% precision and 87.1% recall. Argument identification was performed with 78.9% precision and 77.4% recall. On the test corpus of 300 sentences, our official results, which exclude POS-tags, were 82.0% precision and 78.4% recall for predicate identification and 73.0% precision and 69.1% recall for argument identification.

We did not get the gold standard for the test corpus in time to perform error analysis for our official submission, but we did examine the errors in the LFs we generated for the trial corpus. Most could be traced to errors in the dependency parses, which is unsurprising, since the generation of LFs from dependency parses is relatively straightforward. A few errors resulted from the fact that our system does not try to identify multi-word compounds.

Some discrepancies between our LFs and the LFs provided for the trial corpus arose from apparent inconsistencies in the provided LFs. Verbs with particles were a particular problem. Sometimes, as in sentences 12 and 13 of the trial corpus, a verb-particle combination such as *look forward to* is treated as a single predicate (look_forward_to); in other cases, such as in sentence 35, the verb and its particle (*go out*) are treated as separate predicates. Other inconsistencies in the provided LFs include missing arguments (direct object in sentence 24), and verbs not reduced to base form (*felt*, *saw*, and *found* in sentences 34, 48, 50).

5 Conclusions

Our main finding during the development of the systems for the two Senseval tasks was that semantic relations are indeed very close to syntactic dependencies. Using deep dependency structures helped to keep the manual rules for the LF task simple and made the learning for the SR task easier. Also we found that memory-based learning can be efficiently applied to complex, highly structured problems such as the identification of semantic roles.

Our future work includes more accurate fine-tuning of the learner for the SR task, extending the coverage of the LF generator, and experimenting with the generated LFs for question answering.

6 Acknowledgments

Ahn and De Rijke were supported by a grant from the Netherlands Organization for Scientific Research (NWO) under project number 612.066.302. Fissaha, Jijkoun, and De Rijke were supported by a grant from NWO under project number 220-80-001. De Rijke was also supported by grants from NWO, under project numbers 365-20-005, 612.069.006, 612.000.106, and 612.000.207.

References

M. Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

W. Daelemans, J. Zavrel, K. van der Sloot, and A. van den Bosch, 2003. *TiMBL: Tilburg Memory Based Learner, version 5.0, Reference Guide*. ILK Technical Report 03-10. Available from http://ilk.kub.nl/downloads/pub/papers/ilk0310.pdf.

C. J. Fillmore. 1977. The need for a frame semantics in linguistics. *Statistical Methods in Linguistics*, 12:5–29.

D. Gildea and D. Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3):245–288.

V. Jijkoun and M. de Rijke. 2004. Enriching the output of a parser using memory-based learning. In *Proceedings of ACL 2004*.

C. Johnson, M. Petruck, C. Baker, M. Ellsworth, J. Ruppenhofer, and C. Fillmore. 2003. Framenet: Theory and practice. http://www.icsi.berkeley.edu/ framenet.

A. Ratnaparkhi. 1996. A maximum entropy part-of-speech tagger. In *Proceedings of the Empirical Methods in Natural Language Processing Conference*.

V. Rus. 2002. *Logic Form for WordNet Glosses*. Ph.D. thesis, Southern Methodist University.