On The Duality of Semantics and Syntax: The PP Attachment Case

Dimitar Kazakov, James Cussens and Suresh Manandhar
Department of Computer Science
University of York
{kazakov,jc,suresh}@cs.york.ac.uk

1 Background

The Prepositional Phrase (PP) attachment task is notorious within the computational linguists' community for its ambiguity. When a *verb NP PP* sequence is observed in a sentence, the task is to distinguish between (1) an adverbial prepositional phrase modifying the verb, and (2) an adjectival one modifying the noun phrase on its left side. The corresponding parse trees are as follows:

$$(VP\ verb\ (NP\ )(PP\ )) \tag{1}$$

$$(VP\ verb\ (NP\ (NP\ )(PP\ ))) \tag{2}$$

In many cases there is a single interpretation of the verb phrase in question. For instance, the phrase *to provide water for Texas* corresponds to parse tree 1, whereas *to be the Mayor of Atlanta* is an example of tree 2. However, sometimes phrases such as *to scratch a note on the wall* are truly ambiguous—*wall* can be either the medium of the note or the location where the note is written. For such examples ambiguity cannot be resolved without the help of the verb phrase context.

2 Method

2.1 Overview

PP attachment disambiguation can be based on word semantics. The different syntactic structures behind *to have sandwiches with beef* and *to have sandwiches with friends* are easily perceived by the human reader, who distinguishes between *beef:sem=food* as opposed to *friends:sem=person* [3].

It has been claimed that the set of features used for PP attachment disambiguation could be successfully restricted to four head words `<Verb,Noun,Prep,Noun>` [7]. For instance, <*have, sandwiches, with, friends*> would be used to classify

the phrase *to have some tuna fish sandwiches with those friends of mine.* Alternatively, given the parse tree

$$(VP\ (VB\ eat)\ (NP\ (NN\ toast)\ (PP\ (IN\ with)\ (NP\ (NNP\ Rama)))))) \quad (3)$$

one may expect *Rama* to be a kind of food rather than a person or tool.

The data set of such 4-tuples used in the mentioned article [7] was made freely available.[1] Since its release in 1994, it has been used in several works based on a rule-based approach [1], statistical backed-off models [2], or decision trees [8]. Some of the published approaches [1, 8] make use of semantic tagging. The former uses *all* word senses and their hypernyms to tag words, whereas the latter chooses the most probable tag and compares these results with a manually disambiguated data.

Both approaches use WordNet as a source of semantic tags. WordNet is a lexical database describing semantic relations between its lexical entries [5]. One of these relations, namely *hypernymy* is particularly useful for semantic tagging, because the 'IS_A' relationship that it represents, can be directly used to cluster words corresponding to the same more general word (or *hypernym*).

Each of word's ancestors in the hypernym hierarchy can be used as a semantic tag. Too coarse a tag may not be sufficient to distinguish between different concepts. On the other hand, the choice of fine-grain tags might result in an insufficient number of examples for each of them. Due to homonymy and lexical ambiguity, a particular word form may have more than one hypernym. All that makes the choice of a set of semantic tags a non-trivial task. It has been argued that sense tagging is an intermediate task, and as such it should be evaluated according to the success of the more complex task of which it is a part [9].

In this work, the unsupervised tagging of words with all possible semantic tags as provided by WordNet is adopted to tag `<Verb,Noun,Prep,Noun>` 4-tuples. The semantically tagged 4-tuples are also labelled with the corresponding PP attachment class, 'V' for adverbial attachment and 'N' for the adjectival one. The Inductive Logic Programming (ILP) algorithms Progol [6] and CLOG [4] were then used to learn a first-order theory distinguishing between the two, possibly overlapping, classes. It will be shown that those rules are largely based on semantic labels, i.e., on word senses or hypernyms.

2.2 Using ILP

Inductive logic programming (ILP) is machine learning within a logic programming framework, where hypotheses (H) are induced from examples (E) and background knowledge (B). H, B and E are all logic programs. As input an ILP algorithm receives

Background knowledge B consisting of a finite set of definite clauses

Examples $E = E^- \cup E^+$ where

[1] ftp://ftp.cis.upenn.edu/pub/adwait/PPattachData

- E^+ are positive examples: a finite set of definite clauses (often ground facts)
- E^- are negative examples: a finite set of headless Horn clauses

Hypothesis language constraints which constrain the space of acceptable hypotheses.

As output, an ILP algorithm will produce an hypothesis H which 'explains' the examples with respect to the background knowledge:

- $H, B \models E^+$ (all positive examples are entailed)
- $H, B, E- \not\models E^-$ (consistency with negative examples)

To prevent 'overfitting' the algorithm will search for an hypothesis H which is as compact as possible. This prevents, for example, the set of positive examples E^+ be returned as an 'explanation' of the examples. In practice, some examples (positive and negative) are treated as noise, so the logical conditions on H are relaxed.

2.3 Using Progol to learn attachment rules

In this work, the ILP algorithm P-Progol 2.7.2 was used. This is a Prolog implementation[2] of the Progol algorithm which is explained in detail in [6]. Rules predicting noun attachment and rules predicting verb attachment were learnt separately. In the noun attachment case, positive examples (E^+) were of the form

```
n('Accrued',interest,on,refund).
n(adopting,attitude,of,flexibility).
```

Negative examples (E^-) of noun attachment were simply reformulated positive examples of verb attachment

```
:- n(accuse,'Cohen',of,wimping).
:- n(approve,sale,to,'Indosuez').
```

Unusually for ILP, the bulk of the background knowledge consisted of large tables of ground facts. Firstly, we used predicates mapping nouns and verbs to their base lexical entries:

```
nlx(address,address). nlx(addresses,address).
vlx(add,add). vlx(added,add). vlx(adding,add).
```

We then mapped lexical entries to the set of all WordNet senses for that lexical entry, including all hypernyms. This was done via an intermediary identifier, as shown in Figure 1.

[2] P-Progol was developed by Ashwin Srinivasan (Oxford University)

```
nsemid(address,id2460).
idflathyp(id2460,['100001740','100009457',
'100013018','100014887','100017487','100018604',
...
'106351684']).
vsemid(add,id197).
idVflathyp(id197,['200083947','200125286',
...
'201788486','201880057']).
```

Figure 1: `nsemid/2` and `idflathyp/2` map noun lexical entries to word senses via an `id` identifier. `vsemid/2` and `idVflathyp/2` do the same for verbs

These identifiers identify semantically equivalent lexical entries[3], and also allowed a more compact representation of the semantic information by eliminating redundancy. WordNet tags have been quoted so that they are stored as Prolog atoms rather than integers. This approximately halved the amount of memory required to store the predicates.

Finally the central background predicates `noun_sense/2` and `verb_sense/2` were defined as follows:

```
noun_id(Noun,ID) :- nlx(Noun,LE), nsemid(LE,ID).
noun_sense(Noun,Sense) :- noun_id(Noun,ID),  idflathyp(ID,List),
  member(Sense,List).
verb_id(Verb,ID) :- vlx(Verb,LE),   vsemid(LE,ID).
verb_sense(Verb,Sense) :- verb_id(Verb,ID),  idVflathyp(ID,List),
  member(Sense,List).
```

Predicate	Facts	Kbytes
`n/4` (+)	10752	401
`v/4` (-)	9831	360
`nlx/2`	6561	173
`vlx/2`	3315	83
`nsemid/2`	4896	117
`vsemid/2`	1573	35
`idflathyp/2`	4460	894
`idVflathyp/2`	1542	263
Total	42930	2326

Table 1: Background knowledge and example statistics.

Two learning strategies were used with Progol. In one case the complete set of examples and a greedy search strategy were used. Progol induces an hypoth-

[3]That is, with exactly the same list of possible senses, e.g., *to shut* and *to close*.

esis one clause at a time. In the greedy approach to learning, once a clause is added to the theory, all positive examples covered by the clause are removed. This speeds up learning since the training set of positive examples becomes smaller. On the other hand, this makes it impossible to tell accurately how many positive examples a clause entails, since some might have been previously removed. Knowing how many positive and negative training examples a clause entails is important for gauging the accuracy of the clause, which is especially important in the presence of noise.

To avoid the shortcomings of the greedy search, a non-greedy approach was carried out on half of the data. In this approach,[4] Progol takes each positive example (5451 for noun attachment, 4978 for verb attachment) and uses it as a 'seed' to find the 'best clause' that entails that example. This produced 5001 and 3951 distinct clauses respectively. One advantage of this approach is that the process of inducing the best clause for any particular seed example is completely independent of this process for any other example. This allows one to parallelise the induction process. The search for a theory for noun attachment was split up into 5 P-Progol processes, and that for verb attachment split into 8. Unfortunately, this parallelisation made it difficult to work out the precise amount of CPU time required to induce the theory. However, we can say that each noun attachment clause required on average 163 seconds of CPU time to induce, and each verb attachment clause required 177 seconds. These average timings were used to produce the estimated times in Table 2.

2.4 Using CLOG to learn attachment rules

CLOG is a system for learning first-order decision lists. Details of the algorithm can be found in [4], where CLOG was applied to the learning of morphological rules for Slovene.

First-order decision lists are essentially Prolog clauses in which every clause ends in a cut (!). More generally, decision lists can be thought of as an ordered set of clauses with specific clauses at the top and general clauses at the bottom. The following in an example of a decision list that generates the past tense of English verbs

```
past(A,B) :- split(A,C,[e,p]), split(B,C,[e,t]),!.
...
past(A,B) :- split(B,A,[d]), split(A,C,[e]),!.
past(A,B) :- split(B,A,[e,d]),!.
```

Thus the principal difference to Progol is that the rules learnt by Progol are pure Prolog programs with no ordering on them while the rules learnt by CLOG are ordered from specific-to-general (going from to top to bottom) and only the first (most specific) applicable rule will apply. This means that CLOG PP attachment rules are intended to guess the most likely attachment given the

[4] Using the `induce_max` command. See the P-Progol manual at `http://www.comlab.ox.ac.uk/oucl/groups/machlearn/PProgol/ppman_toc.html` for details

context (*i.e.* <*verb,noun,prep,noun*> tuple). This also provides an important justification for choosing another ILP system apart from Progol - we wanted to verify if a decision-list representation provided a more accurate PP attachment disambiguation.

In our PP attachment learning experiments, CLOG used exactly the same background knowledge as Progol and used all the 10752+9831 = 20583 examples[5] in a single run. Thus, the reader is referred to section 2.3 for details of the setting of the experiment.

3 Results

Examples of rules found by Prolog can be found in Figure 2 (CLOG rules are similar). In Figure 3 we translate some of the rules in Figure 2 into quasi-English. One advantage of ILP's logical framework is that such translations are easily done. Statistics on the number of rules learnt and the time taken can be found in Tables 2–4.

```
n(A, B, _, B) :- verb_sense(A, '200063646').
n(A, B, _, C) :- verb_sense(A, '200045966'),
  noun_sense(C, '100020056'), noun_sense(B, '110843624').
n(A, B, _, C) :- verb_sense(A, '200054862'), %<-
  noun_sense(C, '100019671'), noun_sense(B, '100013243').
n(A, B, _, C) :- verb_sense(A, '200116137'),
  noun_id(C, id4461), noun_sense(B, '105085885').
n(A, B, _, _) :- vlx(A, attract),
  noun_sense(B, '100261466').
n(A, _, about, B) :- verb_sense(A, '201619317'),
  noun_sense(B, '100003731').
```

Figure 2: Example rules learnt by Progol

```
n(V,N,_,N):- verb_sense(V,synset=[grow,develop,produce,acquire]).
n(V,N1,_,N2):- verb_sense(V,synset=[suffer,sustain,have,get,expose]),
  noun_sense(N1,synset=[time_period]),
  noun_sense(N2,synset=[measure,amount,quantity]). %<-
n(V,N1,_,_):- vlx(V,attract), noun_sense(N1,synset=[activity]).
n(V,_,about,N2):- verb_sense(V,synset=[interact]),
  noun_sense(N2,synset=[cause,causal_agent]).
```

Figure 3: Quasi-English version of noun attachment rules learnt by Progol

[5]The actual syntax of the background knowledge and the induced rules used by Progol and CLOG differed slightly but they were equivalent.

Predicate	Rules	Time (est. hours)	Prolog
n/4	1542	226	Yap
v/4	1996	194	Yap

Table 2: Statistics on Progol theories induced from half of the data (non-greedy search, no stopping rule).

Predicate	Rules	Time (hours)	Prolog
n/4	257	301	Sicstus
v/4	541	415	Sicstus

Table 3: Statistics on Progol theories induced from all of the data (greedy search, stopping rule).

3.1 Using attachment rules to perform semantic tagging

The learnt rules predict either noun or verb attachment based on possible semantic tags for the words in the four-tuple. Here we use the rules in reverse. *Given* a four-tuple of head words with known noun or verb attachment, use the rules to provide semantic tags for the words. Consider the 4-tuple (had,million,of,loans) which we know is an example of noun-attachment. Had we not known this, the third rule in Figures 2 and 3 would have predicted this attachment on the basis of the semantics of "had", "million" and "loans". We then use this in reverse: given that (had,million,of,loans) is a noun attachment, the correct tags are those given in the third rule, namely 200045966, 100020056 and 110843624, respectively. Exactly the same argument applies to both CLOG and Progol rules.

In the case of Progol, a number of rules can cover a 4-tuple, giving multiple semantic taggings. Also note that not all rules make use of semantic information for all 3 non-prepositions in the 4-tuple. In such cases, the rule will not constrain possible semantic tags for all 3 non-prepositions. In Clog, due to the decision-list representation, only the first applicable rule is employed, but it may still be the case that multiple senses of a word correspond to the same hypernym chosen by a Clog rule in which case the tuple is not fully disambiguated.

Rules	Time (days)	Prolog
338	15	Sicstus

Table 4: Statistics on CLOG decision list induced from all of the data.

3.2 Evaluation

Using a semantically tagged part of the Brown corpus, test examples were created with both attachment and lexical semantic information:

```
n(receive:'201513366',portion:'100702886',of,funds:'109618851').
v(enter:'201683922',candidate:'107136302',in,race:'105556569').
```

There were 428 noun attachment test examples, and 336 verb attachment examples. For each 4-tuple, we then compared the semantic tags chosen by our theories with the one in the test set. Each rule that covers a test example produces exactly one tag for each word in the example. This is the most specific sense that is (i) consistent with the WordNet information for that word (ii) consistent with the rule. Sometimes the tag so produced was a dummy tag 'top' indicating a virtual most general hypernym. Since CLOG produces a decision list each example is covered by at most one CLOG rule, so only one reading is produced. To provide a baseline we also performed semantic tagging using no rules.

The distance (d) between an hypothesised sense (S_1) for a word and the true sense (S_2) was defined as in [8]. Let S' be the most specific word sense that is a hypernym of both S_1 and S_2. Let L_i be the distance (in edges) between the nodes S_i and S' in the hypernymy graph. Finally, let L be the distance between S' and the topmost element of the hypernymy graph. Then $d(S_1, S_2) = \frac{\frac{L_1}{L+L_1} + \frac{L_2}{L+L_2}}{2}$ The *score* for an hypothesised sense was S_1 simply $1 - d(S_1, S_2)$, where S_2 was the sense given in the test data set. The score for each 4-tuple was simply the average score for the verb and two nouns in the 4-tuple. Note that even when we use no rules, we can achieve a positive score, since some words are semantically unambiguous. Figure 4 gives an example of scoring, where a rule has correctly tagged "receive", made no progress with "portion" and had some success with "funds".

```
n(receive:'201513366',portion:'100702886',of,funds:'109618851').
reading: ['201513366',top,of,'109616555']
score: 0.51
```

Figure 4: Scoring hypothesised semantic tagging

Theory	Mean score	Mean readings/example
None	0.15	1.00
Progol (non-greedy)	0.19	6.18
Progol (greedy)	0.17	1.09
CLOG	0.21	1.00

Table 5: Evaluation of semantic tagging for 4-tuples with noun attachment

Theory	Mean score	Mean readings/example
None	0.15	1.00
Progol (non-greedy)	0.21	3.02
Progol (greedy)	0.18	1.12
CLOG	0.22	1.00

Table 6: Evaluation of semantic tagging for 4-tuples with verb attachment

The results indicate that further work is required to optimise semantic tagging using this approach. Our rules provide a comprehensible argument for an hypothesised tagging (see Figure 3) and achieve more accurate tagging than our baseline, but the level of accuracy is less than desired. In the case of Progol, one possibility would be to rank the rules that cover a test example, and only use the tagging from that ranked best.

4 Conclusion

The approach described here learns rules for syntactic or lexical semantic disambiguation. Given: (i) a set of phrases, (ii) a semantic lexicon (e.g., WordNet), (iii) a number of conflicting grammar rules, each of which covering the phrases (1), e.g., $VP \rightarrow verb\ NP$, $VP \rightarrow verb\ NP\ PP$, (iv) and the correct parsing, i.e., the grammar rule which has to be applied, we specialise the grammar rules (3) using lexical and semantic constraints to reduce the overlap of their scopes.

The learnt rules can be used in two ways: given some phrase and the semantic lexicon, the correct parse of the phrase is suggested. In this work, the relationship between the correct parse of a phrase and the meaning of its terminals is utilised in the opposite way.

Given a phrase tagged with all possible semantic tags (or *readings*) from WordNet, the rules reject the combinations of semantic tags leading to wrong parses, and from the remaining readings favour those corresponding to a rule parsing the input phrase correctly. A specific feature of the rules learnt is that the more ambiguous the data covered by a rule, the stronger the semantic constraints imposed by the rule. For instance, if all training 4-tuples containing the preposition *of* were classified as a noun attachment, then the rule learnt would be `n(V,N1,of,N2):-true.`, i.e., allowing any possible semantic tagging of each of the nouns and the verb. On the other hand, in more ambiguous context, where semantic tags are really needed for disambiguation, the rules may specify restrictions on the set of semantic tags for all 3 open-lexicon words.

The learning of rules for PP-attachment disambiguation described here can be generalised to cover a complete grammar. If the parser is seen as an automaton with a set of states and transitions between them, then the nondeterministic states of the parser will correspond to a set of conflicting grammar rules [3]. For each such set of grammar rules, the approach described here can be repeated.

Our method can be used in a number of ways. It is possible to use it for a fully automatic tagging, partially reducing in such a way the amount of semantic ambiguity in the data. The approach can be especially useful in an interactive framework, where the text is being manually annotated, since our rules can suggest semantic taggings according to the context.

References

[1] E. Brill and P. Resnik. A rule-based approach to prepositional phrase attachment disambiguation. Technical report, Department of Computer and Information Science, University of Pennsylvania, 1994.

[2] Michael Collins and James Brooks. Prepositional phrase attachment through a backed-off model. In *Proceedings of the Third Workshop on Very Large Corpora*, 1995.

[3] Dimitar Kazakov. An inductive approach to natural language parser design. In Kemal Oflazer and Harold Somers, editors, *Proceedings of the Second International Conference on New Methods in Natural Language Processing*, pages 209–217, Ankara, Turkey, September 1996. Bilkent University. ISBN 975-7679-16-X.

[4] Suresh Manandhar, Sašo Džeroski, and Tomaž Erjavec. Learning Multilingual Morphology with CLOG. In *The Eighth International Conference on Inductive Logic Programming (ILP'98)*, Madison, Wisconsin, USA, 1998.

[5] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller. Introduction to WordNet: An on-line lexical database. Technical report, University of Princeton, 1993.

[6] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[7] Adwait Ratnaparkhi, Jeff Reynar, and Salim Roukos. A maximum entropy model for prepositional phrase attachment. In *Proceedings of the ARPA Human Language Technology Workshop*, pages 250–255, 1994.

[8] Jiri Stetina and Makoto Nagao. PP ambiguity resolution through semantic matching.

[9] Yorick Wilks and Mark Stephenson. The grammar of sense: Is word-sense tagging much more than part-of-speech tagging? cmp-lg/9607028.