Syntactic Parsing: Assignment 3

Shifei Chen

---

1 Arc-eager Oracle Error Analysis

As stated in the instruction, "Ideally, the trees output by your oracle parser should be identical to the original trees from the treebank...", but the reality is that there are some errors from my arc-eager oracle. Thinking of this example,

I	PRON	5	nsubj	I	PRON	5	nsubj
was	VERB	5	cop	was	VERB	5	cop
on	ADP	5	case	on	ADP	5	case
my	PRON	5	nmod:poss	my	PRON	5	nmod:poss
way	NOUN	0	root	way	NOUN	0	root
to	ADP	8	case	to	ADP	8	case
my	PRON	8	nmod:poss	my	PRON	8	nmod:poss
wedding	NOUN	5	nmod	wedding	NOUN	5	nmod
fearing	VERB	1	acl	fearing	VERB	0	root
death	NOUN	9	dobj	death	NOUN	9	dobj
,	PUNCT	9	punct	,	PUNCT	9	punct
basically	ADV	9	advmod	basically	ADV	9	advmod
.	PUNCT	5	punct	.	PUNCT	0	root
”	PUNCT	5	punct	”	PUNCT	0	root

One the left hand side we have the dependency tree from the golden standard, while on the right hand side it's the tree predicted from my oracle. The word "fearing" and the last two punctuation, "." and the quotation mark have the incorrect dependencies and labels.

In general, errors happen in a non-projective tree, when a word has a long arc to a previous word that has already been removed from the stack. In the case above, when the oracle was trying to build a left arc for the word "fearing", it's configuration at that moment looked like this,

BUFFER
fearing | death | , | basically | . | ”
STACK
ROOT | wedding

The head word of "fearing", "I" was already been removed when building the left arc (way, nsubj, I) hence at this moment it was not possible to find the head word "I" in the stack again. Also, in an ideal case the head word should stay in the stack if there are dependents later in the sentence, however as the program was predicting and parsing the sentence rule by rule at the same time, the `trasition()` function can never be able to know how many dependents are still remain for a specific head word.

2 Arc-Standard Parser

The non-projective problem went even worse when I was implementing the arc-standard parser as it cannot handle that kind of sentences at all. As a workaround I had to raise an exception and warn the user in the `transition()` function.

Another thing is when building a right arc in the arc-standard parser, we need to loop through all the existing arcs to find if there's any non-implemented arc (S_1, l, k) for the top word in the stack S_1.

I've also rewriten the argument parsing part in `transition.py` so now it can parse sentences in both the arc-eager and arc-standard way (enabled by the argument `-s`).