Assignment 2

Dependency parsing & Part-of-speech tagging

Mourad Akandouch
INFOY004
Academic year: 2017-2018
ULB - VUB

TABLE DES MATIERES

UNIVERSITÉ LIBRE DE BRUXELLES
ULB
VUB
VRIJE UNIVERSITEIT BRUSSEL

Section 1

Introduction

The aim of our second assignment was to use some techniques in order to describe and formalize the syntactic structure of given sentences. The considered sentences for this assignment were the followings:

- *I gave an apple to the teacher*
- *Mary missed her train to work*
- *John gave the teacher a very heavy book*
- *The sun shines*
- *This is the dog that chased the cat*
- *I saw the doctor this morning who is treating me*
- *This is the cat that the dog chased*
- *John is eager to please*

In order to do so, we had to learn and write a program that implement one of the standard approaches for the creation of a dependency parsing: *The transition based dependency parser.* The algorithm runs in linear time is quite straightforward. It needs some data structures to work efficiently: A stack where we will put the words in which we will the dependency relations, an input buffer that contains the remaining word of the current sentence to process and a dependency relations list that will contains the head-dependent relations. During the process, we use standard operations in order to process the data's (*shift, leftarc, righarc).* Among those data structures, we had to implement an Oracle that our parser will consult in order to take and apply the correct operations in order to generate a valid dependency parsing.

In addition, in order to train the Oracle, we had to create feature templates based on the first four sentences. The annotation of the sentences had to be done in the first part of the assignment.

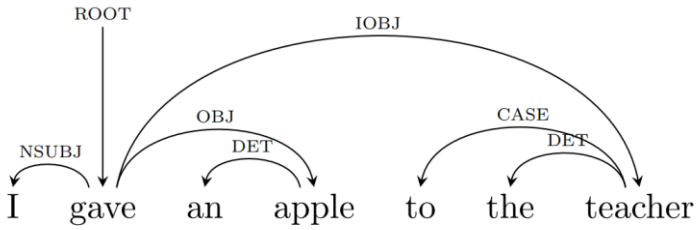
Finally, in the third part of the assignment, we had to run our algorithm with the first four sentences and compare the results against a gold standard. The next sections contains further details and answers to the questions of the assignment.

Section 2

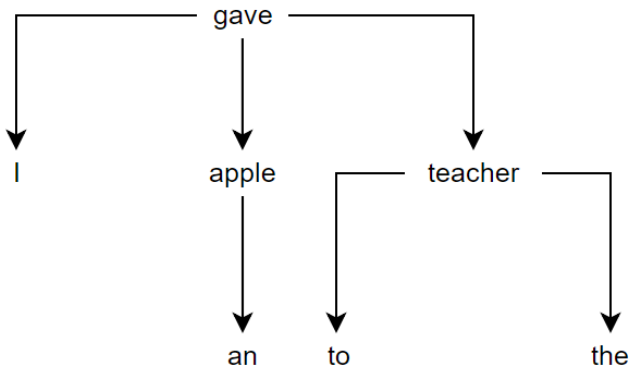
PART 1 - ANNOTATION WITH DEPENDENCY RELATIONS

In the first part of the assignment, we had to annotate the eight given sentences with the universal part-of-speech tags and universal dependency relations. In the archive of the assignment, you will find a file named *annotation.txt* in the CoNNL-U format where I mention all the annotation alongside the head-dependency relations and more useful informations.

For the first sentence, we must also provide a dependency diagram and tree. Here is what I obtain for the dependency diagram:



In addition, the dependency tree:



Discussions

Question 1

Are there ambiguous sentences with multiple possible parses? If yes, what form does the ambiguity take?

Yes there is. For instance, my parser parses the second sentence[1] differently from what I did but leads anyway to the same dependency relations. This is due to the order of the configurations and the actions that the algorithm did. When I manually did the shift reducing algorithm, I put the relation (Mary <- missed) at the step 3 while the parser did it in at the step 11. The way I created my features leaded to the different parsing since I use a priority order, for instance *s1.t o s2.t* has more priority than *s1.t o b1.t*. This is discussed more in the second part of the assignment.

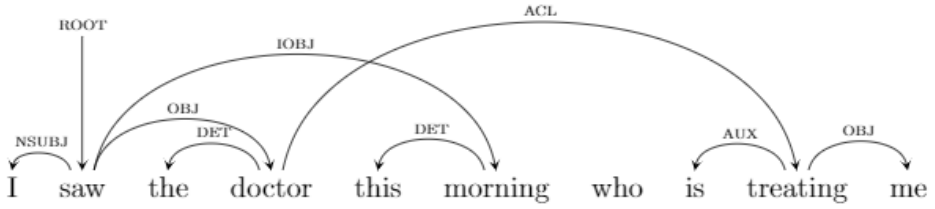
Question 2

Which sentence contains non-projective dependency relations? Which relations exactly and why are they non-projective?

Quoted from the reference book, section 14.2.1 – Projectivity: *"An arc from a head to a dependent is said to be projective if there is a path from the head to every word that lies between the head and the dependent in the sentence. A dependency tree is then said to be projective if all the arcs that make it up are projective." – p.248*

The sentence that is not projective according to that definition is the sixth, ie. *I saw the doctor this morning who is treating me.*

Here is the dependency diagram in order to have a better visualization of the relations:



We can see that there is a relation between *doctor* and *treating* but we cannot find any path from *doctor* to *who*. In addition, we can directly see that there is non-projectivity if there is crossing edges in the diagram, which is the case here.

[1] *Mary missed her train to work*

Section 3

PART 2 - IMPLEMENTATION OF A TRANSITION-BASED DEPENDENCY PARSER

In the second part, we had to implement a transition based dependency parser in Python or Lisp. I chose Python. In order to write that parser, we first had to manually create a general features set on which the Oracle will base its choices. In order to create the features, we had to manually compute the shift reduce algorithm over the first four sentences. Here is a sample of the work done:

Step	Stack	Word List	Action	Relation Added	s1.t	s2.t	b1.t	Suggested feature
0	[root]	[I gave an apple to the teacher]	Shift		root		pron	<s1.t=root, b1.t=pron, Shift>
1	[root, I]	[gave, an, apple, to, the, teacher]	Shift		pron	root		<s1.t=pron, s2.t=root, Shift>
2	[root, I, gave]	[an, apple, to, the, teacher]	LeftArc	(gave, I)	verb	pron		<s1.t=verb, s2.t=pron, LeftArc>
3	[root, gave]	[an, apple, to, the, teacher]	Shift		verb		det	<s1.t=verb, b1.t=det, Shift>
4	[root, gave, an]	[apple, to, the, teacher]	Shift		det	verb		<s1.t=det, s2.t=verb, Shift>
5	[root, gave, an, apple]	[to, the, teacher]	LeftArc	(apple, an)	noun	det		<s1.t=noun, s2.t=det, LeftArc>
6	[root, gave, apple]	[to, the, teacher]	RightArc	(gave, apple)	noun	verb		<s1.t=noun, s2.t=verb, RightArc>
7	[root, gave]	[to, the, teacher]	Shift		verb	root	adp	<s1.t=verb, b1.t=adp, Shift>
8	[root, gave, to]	[the, teacher]	Shift		adp	verb		<s1.t=det, b1.t=noun., Shift>
9	[root, gave, to, the]	[teacher]	LeftArc	(teacher, the)	det	adp	noun	<s1.t=det, b1.t=noun, LeftArc>
10	[root, gave, to, the, teacher]	[]	LeftArc	(teacher, to)	noun	det		<s1.t=noun, s2.t=det, LeftArc>
11	[root, gave, to, teacher]	[]	LeftArc	(gave, teacher)	noun	adp		<s1.t=noun, s2.t=adp, LeftArc>
12	[root, gave, teacher]	[]	RightArc	(root, gave)	noun	verb		<s1.t=verb, s2.t=root, RightArc>
12	[root, gave]	[]	RightArc	(root, gave)	verb	root		<s1.t=verb, s2.t=root, RightArc>
13	[root]	[]	Done		root			<s1.t=root, Done>

We can see that I put at the end of the table the features I think would be interesting for the Oracle. We can also see in the *relation added* field that I use ordered pair (head, dependent) but it is just for me. In the assignment, I do it in the same way it is asked, i.e. with an arrow pointing to the dependent. The features I created only depend on the part-of-speech in order to stay generic as mentioned in the assignment.

As design choice, I implemented three classes:

- TransitionBasedDependencyParser – Its role is to parse a given sentence in the CoNNL-U-format. Thus, it contains a stack, an input buffer and some other data structures that are required to generate the output files.
- Oracle – It is the oracle. We give it the path to a features set file and it has a *consult(self, state)* method that give us the action to do.
- ConLLUToken – Simple wrapper of a word in the CoNNL-U-format.

Regarding the behaviour of the Oracle, I coded it so that it gives priority to some features when there is multiple possible choices. For instance, a feature whose template is *s1.t o s2.t* will be important than an *s1.t o b1.t* but it is not always the case so my Oracle is fulfilled with many nested *if/elif/else* branching operators. For example, there is scenarios where it is correct to choose *shift* than is the usual cases. When the input buffer is not empty and the top of the stack is a determiner, in that case, I give priority to *Shift* because one of my feature could break the parsing. Suppose we have a determiner at the top of the stack and the Root under it. In normal cases according to my features, the *leftarc* operation will be chosen but it may lead to an invalid parsing since the input buffer may contain the noun attached to that determiner.

Section 4

PART 3 - EVALUATION

Final part of the assignment. We have to run our parser over the four first sentences and evaluate the accuracy of our parser given the Oracle we have built. For all the sentences, I got the same dependency relations as when I did it manually. However, the parsing of the second sentence gave me a different configuration order but leads to the same dependency relations. You can see it by visualizing the file *conftable.txt* alongside the table given in the previous section.

Regarding the *unlabelled attachment accuracy,* I am not sure about what it is but if I interpret correctly, then my parser gives me the exact same dependency relations than when I did it manually so I can say that the accuracy is 100%.

I think it worth to mention that the parser failed the first times I executed it because it took choices that are not correct. Since then, I implemented priority rules in order to make better choices. Fun fact, I executed the parser across all the sentences in order to test it and even with the rules I coded, it fails at the sixth sentence I think. The accuracy would be better if we had the permission to take triple-word features or quadruple-word.

Little note: since I had many deadlines during the previous weeks, I excessively rushed this assignment and it will not be as well documented as the previous one.