NLP Assignment 2: Dependency Relations & Transition-based Parser

Alberto Parravicini

1 Introduction

Among the many formalisms to represent the structure of a sentence, **Dependency grammars** provide a flexible and intuitive way to describe how words relate to each other.
Dependency grammars describe the dependencies between words, e.g. to which *noun* a certain *adjective* refers.
Each sentence is represented as a directed tree in which each word $\{w_1, w_2, \ldots, w_n\}$ is a vertex of the tree, and there exists an arc from word w_i to word w_j if and only if word w_j depends on word w_i. In this relation, w_i is referred as **head**, while w_j is the **dependent**.
By convention, we say that the main verb of a sentence is the **root**, and has no incoming arcs.

As a simple example, in the sentence *you sing* the verb *sing* will have an arc to the pronoun *you*, as the latter is a subject of the former.
Furthermore, arcs can be labelled, in order to explicitly say what type of dependency, or **grammatical function**, occurs between two words: in the previous example, the arc would be labelled as **nsubj**, as *you* is the subject of *sing*. The full set of dependency relations can be found here.

This model, by definition, is purely *descriptive*, i.e. it doesn't say how relations between words should be assigned.
To do so, it will be necessary to build a **parser**, an algorithm which scans the sentence and assigns the appropriate relations between each pair of words.
The construction of a parser will be the focus of *section 2* onwards; first, however, we can look at some examples of dependency relations, to get a better understanding of how *dependency grammars* work.

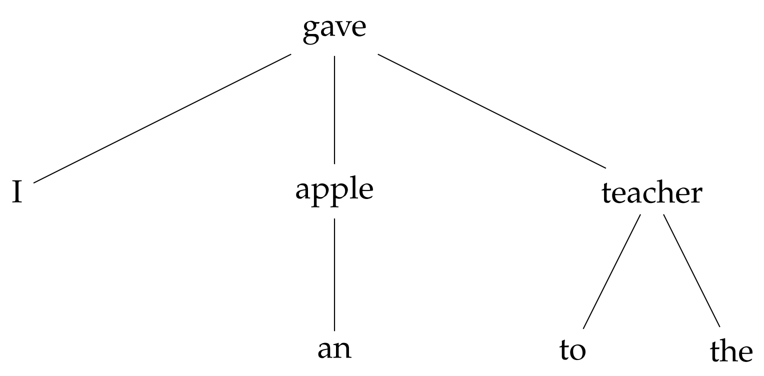
2 Annotation of sentences

Given a small set of sentences, we can annotate them as described in the **CoNNL-U** format. For each word, we denote its *lemma*, its *part-of-speech* (**POS**, according to the universal part-of-speech tags), its *head* and *dependency relation*, as defined above. The **CoNNL-U** format contains other fields, but for our purpose they can be omitted.

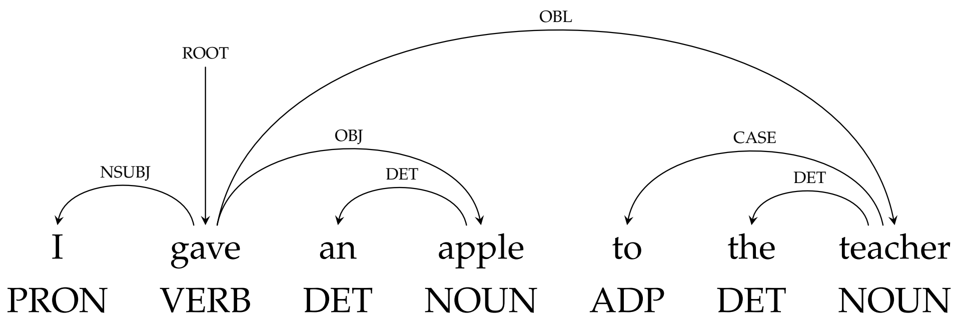
The sentences that we consider are, in order:

1. I gave an apple to the teacher
2. Mary missed her train to work
3. John gave the teacher a very heavy book
4. The sun shines
5. This is the dog that chased the cat
6. I saw the doctor this morning who is treating me
7. This is the cat that the dog chased
8. John is eager to please

The full annotaion of each sentence if given in the file **annotation.txt**. For the first sentence (*I gave an apple to the teacher*) are provided a *dependency tree* and a *dependency diagram*, which also shows the type of grammatical relation between the words.



Dependency tree of "I gave an apple to the teacher".


OBL
ROOT
OBJ
CASE
NSUBJ
DET
DET
I gave an apple to the teacher
PRON VERB DET NOUN ADP DET NOUN

The structure of the sentence is rather simple, and can be represented as **subject** + **verb** + **object** + **indirect object**; note that the indirect object is defined *oblique*, and the **to** apposition is called *case*.

As for the other sentences, we can make the following observations:

- In some sentences, certain words could have more than one *part-of-speech* tag. However, in most cases only one tag is grammatically correct, given the context.
- As an example, in the second sentence the word *work* can be seen as a noun or as a verb, but given the context only the former interpretation is valid. The same applies to sentence 4 (*shines* can be a - rather unusual - noun).
- **That** can be either a *determinant* (as in *that book*) or a *subordinating conjunction*, as in sentence 5.
- *This morning* can be seen as a *determinant* followed by a *noun*, or as a single *temporal adverb*, composed by 2 words. In this case, both options are grammatically correct.
- Sentence 6 could be seen as *non-projective*: specifically, if we say that *treating* depends from *doctor*, then the arc that joins the 2 words will be *non-projective*, as there is no path from *doctor* to *this morning*.
  However, it is also possible to interpret the sentence in a way that *treating* depends from the main verb *saw*, and in this case the sentence would be projective.
  In this case, the non-projectivity arises from the fact that *this morning* can be placed rather freely in a sentence: it could be at the start, in the middle, or at the end, and in every case the sentence would be grammatically correct.

3 Implementation of the parser

In order to assign correct dependency relations to each word in the sentence, we have to build a parser: in our case, the parser will take the form of a greedy *shift-reduce* algorithm.
At each step, the algorithm will decide which action to take: whether to read a word from the input buffer, or to assign a *dependency relation*. For simplicity, we consider only **left** arcs, **right** arcs, and **shifts**. No other dependency relation is specified.

Words that are read from the input buffer are placed in a stack: then, the algorithm will decide which action to take, based on the content of the stack and of the input buffer. If a dependency relation is added, a word is removed from the stack. The stack is initialized with a special element denoted as **__ROOT__**.

Now, there are two issues to discuss.

First, when should a *left* or *right* action be taken?
When a *left* arc is assigned, the dependent word is removed from the stack. As such, we can assign left arcs without many complications: the only borderline case is when the root is the second element in the stack.
In that case, it is forbidden to assign a *left* arc, as it would imply that the root is dependent from some other word!
Right actions are more complex to handle: the assignment of a right arc will lead to the removal of te **head**, and as a consequence we must be sure that all the words dependent on that head have already been marked with a *left* arc; otherwise it will be impossible to flag them correctly, as their head has been removed.
In our algorithm, the assignment of right arcs is problematic as we can be sure that a word has no other dependents only after reading the entire sentence. Without being able to do so, we must be able to predict efficiently if a word will have some other dependent further down the sentence.

The second issue is related to how the action to be made is chosen. Ideally, we would have an omniscient oracle that always spits out the right action to be performed.
More realistically, our oracle will have to predict the best action given the current state of the parser, i.e. the stack and the input buffer.
This prediction can be done by extracting features from the state, such as the *part-of-speech* tags of the words in the stack and in the input buffer.

The features can on paper be computed by the parser itself, and fed as input to the oracle. However, a more flexible approach consists in passing the state of the parser to the oracle, and leave the feature construction to the oracle itself.
This approach has the advantage of isolating completely the oracle from the

parser: it is easily possible to use different kinds of oracles, without having to modify at all the parser algorithm.

In the proposed implementation, we initially experiment with a vary basic oracle that will provide random actions (given some reasonable constraints, such as not shifting if the buffer is empty), then move to a **rule-based** oracle.

The rule based oracle is built by inspecting the structure of the first 4 sentences given before. To provide more versatility, the oracle will look at the *part-of-speech* tags of the words, not at the words themselves.
Moreover, the oracle will look at the first and second words on the stack and in the input buffer.
Rules are provided as *if-else* statements, and as such there is no reason no explicitly build 2-words features (i.e. features build by using two words), as we can simply combine the single word features inside the *if-else* statements.

For the explanation of the meaning and the role of each single rule, please refer to the comments in the code. Still, it is possible to make some more general considerations regarding the structure of the rules.

These rules are quite similar to the ones used by **Classification rules** algorithms, and the same line of thoughts can be applied to the construction of the oracle's rules.

- A general idea, is that the first rules to be considered, i.e. the ones at the top, should be the most restrictive ones, and match the most specific cases. As an example, it is considered the rules that matches the pattern **"verb"** + **"det"** + **"adv/adj"** + **"adv/adj"**, which is usually followed by a nominal to which the last two elements refer (as in sentence 3); as such, it is appropriate to shift.

- To cover situations that are not matched by any rule, it is important to have a default rule: in our case, a random action will work, provided that no constraint is violated (in that case, the action is changed appropriately).

- In general, it would be preferrable to have non-overlapping rules: overlaps occur when a single input is matched by more than one rule: in this case, the first rule to appear is the one that will be used.
  Measuring if rules overlap is quite difficult, but as a rule of thumb having restricting rules at the top should avoid unwanted behaviours.

On a side note, the features generated by the **rule-based** oracle have been saved to a file (**feattemp.txt**), along with the action that correspond to each feature vector. This dataset has been used to train a simple decision tree, to see if it would be able to reconstruct, or even improve, the results of the rule-based oracle.

However, the results are very poor, as the tree is unable to predict *right arcs*, as seen by the drawing of the tree structure (in the file *tree.pdf*).

3.1 *Execution traces*

Below are reported the execution traces of the parsing of the first 4 sentences.

Step	Stack	Buffer	Action	Output
0	[__ROOT__]	[I, gave, an, apple, to, the, teacher]	SHIFT	
1	[__ROOT__, I]	[gave, an, apple, to, the, teacher]	SHIFT	
2	[__ROOT__, I, gave]	[an, apple, to, the, teacher]	LEFT	(I $\rightarrow$ gave)
3	[__ROOT__, gave]	[an, apple, to, the, teacher]	SHIFT	
4	[__ROOT__, gave, an]	[apple, to, the, teacher]	SHIFT	
5	[__ROOT__, gave, an, apple]	[to, the, teacher]	LEFT	(an $\leftarrow$ apple)
6	[__ROOT__, gave, apple]	[to, the, teacher]	RIGHT	(gave $\rightarrow$ apple)
7	[__ROOT__, gave]	[to, the, teacher]	SHIFT	
8	[__ROOT__, gave, to]	[the, teacher]	SHIFT	
9	[__ROOT__, gave, to, the]	[teacher]	SHIFT	
10	[__ROOT__, gave, to, the, teacher]	[]	LEFT	(the $\leftarrow$ teacher)
11	[__ROOT__, gave, to, teacher]	[]	LEFT	(to $\leftarrow$ teacher)
12	[__ROOT__, gave, teacher]	[]	RIGHT	(gave $\rightarrow$ teacher)
13	[__ROOT__, gave]	[]	RIGHT	(__ROOT__ $\rightarrow$ gave)
14	[__ROOT__]	[]	DONE	

Table 1: Execution trace for *I gave an apple to the teacher*.

Step	Stack	Buffer	Action	Output
0	[__ROOT__]	[Mary, missed, her, train, to, work]	SHIFT	
1	[__ROOT__, Mary]	[missed, her, train, to, work]	SHIFT	
2	[__ROOT__, Mary, missed]	[her, train, to, work]	LEFT	(Mary $\leftarrow$ missed)
3	[__ROOT__, missed]	[her, train, to, work]	SHIFT	
4	[__ROOT__, missed, her]	[train, to, work]	SHIFT	
5	[__ROOT__, missed, her, train]	[to, work]	LEFT	(her $\leftarrow$ train)
6	[__ROOT__, missed, train]	[to, work]	RIGHT	(missed $\rightarrow$ train)
7	[__ROOT__, missed]	[to, work]	SHIFT	
8	[__ROOT__, missed, to]	[work]	SHIFT	
9	[__ROOT__, missed, to, work]	[]	LEFT	(to $\leftarrow$ work)
10	[__ROOT__, missed, work]	[]	RIGHT	(missed $\rightarrow$ work)
11	[__ROOT__, missed]	[]	RIGHT	(__ROOT__ $\rightarrow$ missed)
12	[__ROOT__]	[]	DONE	

Table 2: Execution trace for *Mary missed her train to work.*

Step	Stack	Buffer	Action	Output
0	[__ROOT__]	[John, gave, the, teacher, a, very, heavy, book]	SHIFT	
1	[__ROOT__, John]	[gave, the, teacher, a, very, heavy, book]	SHIFT	
2	[__ROOT__, John, gave]	[the, teacher, a, very, heavy, book]	LEFT	(John $\leftarrow$ gave)
3	[__ROOT__, gave]	[the, teacher, a, very, heavy, book]	SHIFT	
4	[__ROOT__, gave, the]	[teacher, a, very, heavy, book]	SHIFT	
5	[__ROOT__, gave, the, teacher]	[a, very, heavy, book]	LEFT	(the $\leftarrow$ teacher)
6	[__ROOT__, gave, teacher]	[a, very, heavy, book]	RIGHT	(gave $\rightarrow$ teacher)
7	[__ROOT__, gave]	[a, very, heavy, book]	SHIFT	
8	[__ROOT__, gave, a]	[very, heavy, book]	SHIFT	
9	[__ROOT__, gave, a, very]	[heavy, book]	SHIFT	
10	[__ROOT__, gave, a, very, heavy]	[book]	LEFT	(very $\leftarrow$ heavy)
11	[__ROOT__, gave, a, heavy]	[book]	SHIFT	
12	[__ROOT__, gave, a, heavy, book]	[]	LEFT	(heavy $\leftarrow$ book)
13	[__ROOT__, gave, a, book]	[]	LEFT	(a $\leftarrow$ book)
14	[__ROOT__, gave, book]	[]	RIGHT	(gave $\rightarrow$ book)
15	[__ROOT__, gave]	[]	RIGHT	(__ROOT__ $\rightarrow$ gave)
16	[__ROOT__]	[]	DONE	

Table 3: Execution trace for *John gave the teacher a very heavy book*.

Step	Stack	Buffer	Action	Output
0	[__ROOT__]	[The, sun, shines]	SHIFT	
1	[__ROOT__, The]	[sun, shines]	SHIFT	
2	[__ROOT__, The, sun]	[shines]	LEFT	(The $\leftarrow$ sun)
3	[__ROOT__, sun]	[shines]	SHIFT	
4	[__ROOT__, sun, shines]	[]	LEFT	(sun $\leftarrow$ shines)
5	[__ROOT__, shines]	[]	RIGHT	(__ROOT__ $\rightarrow$ shines)
6	[__ROOT__]	[]	DONE	

Table 4: Execution trace for *The sun shines*.

4 Parser Evaluation

After parsing the sentences, we can inspect what is the accuracy of our parser. We can notice how the parser is able to predict the dependency relations with 100% accuracy, and (obviously) values of 100% also in terms of precision and recall relatively to the 3 actions.
With such a small test set it is hard to say if some rules were over-specific, or if the results are explained by the rather simple structure of the 4 sentences that were tested.

To get some further insight of the performances of the model, we could test it against the other 4 sentences that we were given.
It should be immediately pointed out that the results will be mediocre at best, as our **rule-based** oracle doesn't keep into account some of the *part-of-speech* tags that appear in the other sentences.
Still, this test could shed some light on the weaknesses of the model we built.

The input and the output of this test are saved as **input_test.txt** and **output_test.txt**.

On these test sequences, we can note the following:

- The oracle is unable to identify a *determiner* as subject of the fifth sentence. It will not be able to recognize the main verb as root of the sentence, so it will either read the entire input buffer and proceed to assign dependencies at random (or by recognizing some simple pattern, such as *determiner* + *noun*), or it will pick the verb of a subordinate clause as main verb.
- In the fifth and sixth sentences the oracle is unable to handle the subordinate clauses. Specifically, it can't handle the fact that the verb of the subordinate is dependent on the main verb.
- The seventh sentence has similar problems as before: a determiner as subject, and a relative clause.
- In the sixth and in the eight sentences, the parser cannot recognize the role of the auxiliary verb. In the first case the oracle can still recognize the second part of the verb, but in the last sentence the auxiliary verb is used as copula, and the parser cannot identify the main verb of the sentence.

On the test sentences, we can measure an overall accuracy of $\frac{15}{31} = 0.48$, including the root arcs. Moreover, we have an accuracy of $\frac{8}{15} = 0.53$ for the left arcs and of $\frac{5}{12} = 0.41$ for the right arcs (root arcs are not included in this case).

The data seems to partially confirm the intuition that right arcs are more difficult to predict, even tough the sample size at our disposal is too small to draw any definitive conclusion.