Natural Language Processing

Lab 9: Dependency Parsing

1 Introduction

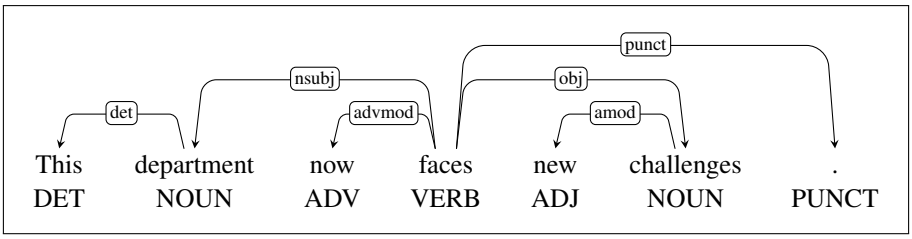
In this lab, we will train and evaluate a transition-based dependency parser using data from the Universal Dependencies (UD) project.[1] The input to parsing will be text that has been tokenized, tagged and lemmatized as in earlier labs, so by putting all these tools together you can build your own pipeline for grammatical analysis of English text. We will use the MaltParser system to build the parser and the MaltEval tool for evaluation and error analysis.

2 Data

We will use the training and development set of the English web treebank, as well as the 10 sentences that you annotated in Lab 8, which all come from the UD project. The files can be found in `/local/kurs/nlp/syntax/`. For the purpose of this assignment, we have removed some of the information in the original annotation so that the parsing will be based only on information that can be predicted using tools previously explored in the course, as shown below.

```
1    This          this          DET     DET     _    2    det      _    _
2    department    department    NOUN    NOUN    _    4    nsubj    _    _
3    now           now           ADV     ADV     _    4    advmod   _    _
4    faces         face          VERB    VERB    _    0    root     _    _
5    new           new           ADJ     ADJ     _    6    amod     _    _
6    challenges    challenge     NOUN    NOUN    _    4    obj      _    _
7    .             .             PUNCT   PUNCT   _    4    punct    _    _
```

The format used to represent dependency trees is known as the CoNLL-X format. Part-of-speech tags and lemmas are found in columns 4–5 and 3, respectively. The indices in column 7 encode an unlabeled dependency tree by pointing to the head of each word, while the labels in column 8 specify the function of each word.[2] Below is a more conventional visualization of the same dependency tree.



3 Install MaltParser

Go to the MaltParser website at `http://maltparser.org`. Go to the Download page and download the latest version `maltparser-1.9.1.tar.gz`. Open a terminal and unpack the compressed archive by running:

```
tar xvf maltparser-1.9.1.tar.gz
```

Then change directory:

```
cd maltparser-1.9.1
```

Then run MaltParser:

```
java -jar maltparser-1.9.1.jar
```

This should produce output starting as follows:

```
----------------------------------------------------------------------------
                         MaltParser 1.9.1
----------------------------------------------------------------------------
         MALT (Models and Algorithms for Language Technology) Group
             Vaxjo University and Uppsala University
                              Sweden
----------------------------------------------------------------------------
```

If you don't get this output, notify the the lab instructor.

[1] `http://universaldependencies.org`

[2] For more information about the CoNLL-X format, see `http://ilk.uvt.nl/conll/#dataformat`. UD actually uses an evolved version of CoNLL-X called CoNLL-U, but the differences are irrelevant for this lab.

4 Train a parser

To train a parser, run a command like the following:

```
java -jar -Xmx2g maltparser-1.9.1.jar -c myparser -m learn -i ewt-train.conll
```

This command can be broken down into two parts, where the first is:

```
java -jar -Xmx2g maltparser-1.9.1.jar
```

This tells the Java Virtual Machine on your computer to run MaltParser with a heapspace of 2GB. (The heapspace is set by the flag `-Xmx2g`. If you forget this flag, the process will probably run out of memory.) The rest of the command consists of specific MaltParser parameters:

- `-c` The name of the parsing model you are creating, which can be anything you like. If you name your model `myparser`, it will be saved in a file could `myparser.mco`.
- `-m` The processing mode. This should be `learn` when you are training a model.
- `-i` The input file. This should be the name of your training set (`ewt-train.conll` in our example).

Training a model should take somewhere between 10 seconds and a few minutes, depending on the size of the training set. If it takes longer (or crashes), notify the lab instructor.[3]

5 Parse some new data

To parse a new set of sentences using the trained parser, run a command like the following:

```
java -jar -Xmx2g maltparser-1.9.1.jar -c myparser -m parse -i ewt-dev.conll -o out.conll
```

The first part of the command is the same as for training, telling the JVM to run MaltParser, but the MaltParser parameters are slightly different:

- `-c` The name of the parsing model, which has to be trained beforehand. For example, if we specify `myparser`, the model saved as `myparser.mco` will be used.
- `-m` The processing mode. This should be `parse` when you are parsing new data.
- `-i` The input file to be parsed (`ewt-dev.conll` in our example).
- `-o` The output file where parses will be saved (`out.conll` in our example).

6 Evaluate the parser

To evaluate the parse results, you first need an evaluation tool. We are going to use `MaltEval.jar`, which you can copy from `/local/kurs/nlp/syntax/`. You can then evaluate your parser by running:

```
java -jar -Xmx2g MaltEval.jar -g ewt-dev.conll -s out.conll
```

The output given is the labeled attachment score (LAS) and the number of tokens included in the evaluation. There are a number of flags that can be used to get more output from MaltEval. For example, to get precision and recall for different dependency relations, add the flag:[4]

```
--GroupBy Deprel:all
```

7 Do an error analysis

MaltEval can also be used to visualize dependency trees with color coding of parse errors. To make the task more manageable, first run the parser on the 10 sentences in `en10.conll`, saving the output in a file called `out10.conll`. To visualize the parse trees and compare them to the gold standard, run:

```
java -jar -Xmx2g MaltEval.jar -g ewt-dev.conll -s out.conll -v 1
```

Use the buttons next to the list of the sentences at the bottom to navigate either to the next or previous sentence or to the next or previous parse error. Go through all 10 sentences and discuss the parse errors you see. If you have time left, you can continue by discussing parse errors in the larger file `out.conll`.

[3] MaltParser has a large number of options that can be used to tune the parser. For more information, see the MaltParser website.

[4] For more information about the options available, see the MaltEval user guide, available on the MaltParser website.