INF5830, Fall 2017
Assignment 3: Dependency Parsing (Project A)

Andrey Kutuzov

October 9, 2017

Deadline 27 Oct., at 18:00, to be delivered in Devilry

Goals

- Get familiar with the *CoNLL-U* dependency treebank format.
- Learn to train and evaluate dependency parsing models with one of the mainstream parsers.
- Learn to change the training data according to your needs.

Introduction

Modern algorithms of dependency parsing (i.e., analyzing syntactic structure of a sentence) serve as the building bricks for many automated systems dealing with natural language. They provide the data necessary for further 'understanding' of text. The knowledge of how such algorithms work and what data is used to train them is indispensable for any person studying computational linguistics.

It can be profitable to try to implement a parsing algorithm from scratch in your favorite programming language (we encourage you to do this); however, in real life tasks, it is usually much more convenient to employ already available implementations and train your own models with them, or even re-use some of the pre-trained models in your downstream tasks. Thus, the current assignment aims to make you familiar with some of this software. We recommend *MaltParser* and *UDPipe*, but you are free to use other existing tools: *MSTParser*[1], *Stanford CoreNLP*[2], *Spacy*[3] etc. Additionally, throughout the assignment the *Universal Dependencies* treebanks will be used as the source of training and testing data.

Please make sure you read through the entire assignment before you start. If you have any questions, please email *andreku@ifi.uio.no*, and make sure to take advantage of the group sessions. Solutions must be submitted through Devilry[4] by 18:00 on October 27. Please upload a single PDF file with your answers.

[1] `https://sourceforge.net/projects/mstparser/`
[2] `https://stanfordnlp.github.io/CoreNLP/depparse.html`
[3] `http://spacy.io/`
[4] `https://devilry.ifi.uio.no/`

Your (heavily commented) code and data files should be available in a separate private repository in the UiO in-house Git platform[5]. Make the repository private, but allow full access to me (`https://github.uio.no/andreku`). The PDF in Devilry should contain a link to your repository.

Recommended reading

1. **Speech and Language Processing**. Daniel Jurafsky and James Martin. 3rd edition draft of August 28, 2017. Chapter 14, 'Dependency parsing'[6].
2. **Maltparser: A data-driven parser-generator for dependency parsing**. Nivre, Joachim, et al., 2006.[7].
3. **Universal Dependencies v1: A Multilingual Treebank Collection**. Nivre, Joachim, et al., 2016.[8].
4. **Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe**. Straka, M., Straková, J., 2017.[9].
5. `http://www.maltparser.org/`
6. `http://ufal.mff.cuni.cz/udpipe`
7. `http://universaldependencies.org/`
8. `http://universaldependencies.org/conll17/`

1 Spreading the word

It seems that the Norwegian Wikipedia lacks an article describing data-driven natural language dependency parsing (`https://no.wikipedia.org/wiki/Parsing` is too general). Your task is to fix this.

This is a collective assignment: all Norwegian-speaking students in the group should work on it. The outcome should be a separate Wikipedia page in Norwegian (linked from `https://no.wikipedia.org/wiki/Parsing`) about natural language dependency parsing, containing a brief gist of what you have learned from this course. There are no restrictions on minimum or maximum size of this article, but it should give an arbitrary reader an idea about what data-driven parsing is.

It is your responsibility to organize and decide who writes which part of the article. Once you are done with the text, you should create the corresponding Wikipedia page. Then, each of you should log in under his/her Wikipedia username and edit the page, adding his/her part. In the report file, you should mention your Wikipedia username (different for each of you) and the address of the newly created page (the same for all).

[5] `https://github.uio.no/`
[6] `http://web.stanford.edu/~jurafsky/slp3/14.pdf`
[7] `http://lrec-conf.org/proceedings/lrec2006/pdf/162_pdf.pdf`
[8] `https://nlp.stanford.edu/pubs/nivre2016ud.pdf`
[9] `http://www.aclweb.org/anthology/K17-3009`

In case you do not feel that your knowledge of Norwegian is good enough for this task, you can create (or enrich if it already exists) an article about data-driven natural language dependency parsing in your native language Wikipedia. The size of one paragraph will be enough. Please do not update the English Wikipedia, it already has much data on that topic.

Please take this task seriously and write responsibly: your text will be read and used by many people. Don't describe things about which you are unsure. Support your text with references, as per Wikipedia guidelines.

2 Tracing non-projectivity

Discontinuous or *non-projective* dependency trees are trees which contain crossing edges (arcs). In other words, it means that there exist patterns like $(d_1 d_2 h_1 h_2)$, with d_i denoting a dependent and h_i denoting its head. In some languages, such patterns are rare, while in others they abound.

Your task is to find out how widespread is non-projectivity in the languages of the world. For this, we will use syntactically annotated corpora (**treebanks**) from the *Universal Dependencies* project (UD).

1. Download the UD treebanks (`http://hdl.handle.net/11234/1-1983`, you will need the first archive – *ud-treebanks-v2.0.tgz*);
2. Extract the files from the archive; there should be 70 treebanks for 50 languages;
3. Look through the files; you will need those from the **train set** (for example, *en-ud-train.conllu*);
4. Implement a code in Python which takes as an input a treebank in *CoNLL-U* format and outputs its non-projectivity statistics:
   - What is the percentage of non-projective arcs?
   - What is the percentage of non-projective sentences?
5. You can optionally make use of any of the tools developed by the UD project (`http://universaldependencies.org/tools.html`).
6. Compute non-projectivity statistics for all the UD treebanks.
7. Can you see any patterns in these stats? Describe them.
8. Is there any correlation between the amount of non-projectivity in languages and the results that modern data-driven parsers are able to achieve on these languages? You can use the Table 10 from *Zeman, D. et al. 2017*[10] or the same data directly from `http://universaldependencies.org/conll17/results-treebanks.html`.

Your PDF report should contain detailed description of your experiments and the tables with the stats you computed. The code developed by you should be available at the UiO Github.

[10] `https://aclanthology.info/pdf/K/K17/K17-3001.pdf`

3 Using parsers

In this task, you will train and evaluate basic dependency parsing models. You can choose one or both of the following parsers:

1. *MaltParser*[11]
2. *UDpipe*[12]

MaltParser is a traditional transition-based dependency parser written in Java. *UDpipe* is a more modern parser written in C++. It is transition-based as well, but uses artificial neural networks to train its oracle (guide). You are also free to use any of other available parsers, if you want.

3.1 Training

1. Download the parser(s) of your choice.
2. Using it, train a model on **any two languages of your choice** from the UD treebank collection. Use languages from different language groups (not English).
   - for MaltParser, a model can be trained with *java -jar maltparser-1.9.1.jar -c MY_MODEL -i TRAINING_FILE -m learn*
   - for *UDpipe*, a parser model can be trained with *udpipe –train MY_MODEL –tokenizer=none –tagger=none –parser use_gold_tags=1 TRAINING_FILE*
   - Note that training a *MaltParser* model on a typical UD treebank takes 2 or 3 minutes, while training a *UDPipe* model on the same treebank can take 2 or 3 *hours*.
3. Read the documentation to the parser(s) you used. What features do they use by default?

3.2 Evaluation

In order to evaluate your freshly trained models, we need some test data.

1. Download the UD test treebanks (`http://hdl.handle.net/11234/1-2184`).
2. Extract the files from the archive.
   - The folder *input/conll17-ud-test-2017-05-09* contains test treebanks with missing heads and dependency labels.
   - The folder *gold/conll17-ud-test-2017-05-09* contains the same treebanks with the correct heads and dependency labels.
3. Employ your models to parse the corresponding test treebanks from the *input* folder:

[11] `http://www.maltparser.org/`
[12] `http://ufal.mff.cuni.cz/udpipe`

- for MaltParser, a *CoNLL* file can be parsed with *java -jar maltparser-1.9.1.jar -c MY_MODEL -i TEST_FILE -o PARSED_OUTPUT_FILE -m parse*
- for *UDpipe*, a *CoNLL* file can be parsed with *udpipe –parse MY_MODEL TEST_FILE –outfile=PARSED_OUTPUT_FILE*

4. Now you have the parsings produced by your models. You have to evaluate them against gold parsing with the same evaluation script that was used to find winners of the CoNLL 2017 Shared Task.
5. The script can be found in the UD test set archive which you have just downloaded: *evaluation_script/conll17_ud_eval.py*.
6. You should use the corresponding *treebanks* from the *gold* folder as a gold standard. Here is an example of running the evaluation script for English with *en.conllu* as the gold file and *file2test.conllu* as the parsing produced by the model we want to evaluate:

```
python conll17_ud_eval.py en.conllu file2test.conllu -v
Metrics    | Precision |    Recall |  F1 Score | AligndAcc
-----------+-----------+-----------+-----------+-----------
Tokens     |     98.86 |     98.48 |     98.67 |
Sentences  |     81.39 |     66.54 |     73.22 |
Words      |     98.86 |     98.48 |     98.67 |
UPOS       |     93.29 |     92.94 |     93.11 |     94.37
XPOS       |     92.60 |     92.25 |     92.42 |     93.67
Feats      |     94.15 |     93.80 |     93.97 |     95.24
AllTags    |     91.21 |     90.86 |     91.04 |     92.26
Lemmas     |     95.79 |     95.43 |     95.61 |     96.90
UAS        |     77.26 |     76.97 |     77.12 |     78.16
LAS        |     74.12 |     73.84 |     73.98 |     74.98
CLAS       |     69.15 |     68.56 |     68.85 |     69.65
```

Note that *MaltParser* can occasionally generate trees with multiple ROOT nodes. This raises an error in the CoNLL17 evaluation script. Thus, if you use *MaltParser*, you would probably want to comment out this check in the evaluation script (around line 180).

Evaluate the models you trained on the corresponding gold data. Include the values of UAS, LAS and CLAS in your PDF report. Which model was better, and thus which language was easier for the parser? Give your opinion on that.

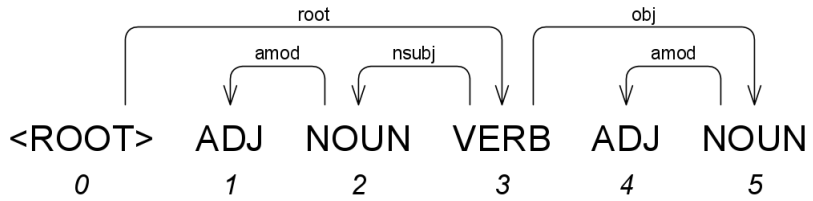
- If you used more than one parser:
  - describe the differences in their performance.
- If you used only one parser:
  - try to change at least one training parameter of the parser (read the documentation thoroughly) and train another set of models;
  - describe how these two sets differ in performance and why, in your opinion.

How far are your models below the state-of-the-art results for these languages (as expressed in the reports from the CoNLL 2017 Shared Task)?

4 Parsing one language with another

The models you trained in the previous task made use of many features to produce feasible dependency trees. However, sometimes one is interested in one particular feature. Suppose you conduct a study on the interplay between parts of speech (PoS) and dependency structures in different languages.

Intuitively, it is obvious that in many cases it is possible to infer a dependency tree from the sequence of PoS tags only, without any notion of word forms or semantics. For example, if we know that the language is English and we see the sequence like 'ADJ NOUN VERB ADJ NOUN', it is very probable that the corresponding dependency tree would be:



Thus, one can train a dependency parser on PoS tags only. Its performance will probably be lower than when training on the full set of features, but still better than random. But what's more interesting is that we can use this approach to find out how similar different languages are with respect to the interaction between PoS and dependency trees. As all the UD treebanks use one and the same **Universal PoS tagset**, we are technically able to use a trained model to parse a treebank in some language L_2 different from the language L_1 of the training treebank. Then we can check how accurate was this parsing, and in this way – how similar is L_2 to L_1 in this respect.

Your task is to conduct such an experiment on any 8 languages of your choice from the UD treebank and report the results. Once again you can use any of the parsers we mentioned (or several parsers). The necessary steps can be the following:

1. Choose the languages.
2. Choose the parser(s).
3. Train models for these languages using the UD treebanks as training data and *considering only PoS tags*.
4. Evaluate the resulting models on the test treebank from the language it was trained on **and** on the test treebanks from 7 other languages.
5. Describe the results and explain how they support (or undermine) the known linguistic facts about genetic and typological similarities between the languages you experiment with[13].

One of the simples way to train your model on PoS tags only is to pre-process the training CoNLL file to 'flatten' all the other features: for example, replacing

[13] See `https://en.wikipedia.org/wiki/Genetic_relationship_(linguistics)` and `https://en.wikipedia.org/wiki/Linguistic_typology`

them with '_'. Your code performing this task should also be available in your Git repository. Another option is to play with the parser settings. In this case, you have to describe it in your PDF report.

The report should include the values of UAS, LAS and CLAS for all combinations of your training and testing data. It makes sense to represent it as 3 matrices/tables, with training languages as rows and testing languages as columns. Any other good-looking visualizations of the results are encouraged.

Finally, compare the results of your models on the same language with all the features and with only PoS tags. Rank the languages by the difference in the performance of the full model and the PoS-only model. Does it correspond to your intuition about the nature of these languages?