

Dependency Parsing: Parsing with the Eisner Algorithm (Part 2)

Deadline: 22.05.2017. Please send your solutions (to the exercises in this tutorial and the ones last week) to jakub.waszczyk@phil.uni-duesseldorf.de with subject "dependency homework" and an attachment "eisner_lastname(s).pdf".

1 Introduction

This tutorial is a continuation of the tutorial started last week. Please continue working under linux.

The goals for today are to:

- Train our arc-factored projective parser
- Evaluate the resulting model
- Tweak model features so as to improve parsing results

Download the archive accompanying the today's session. It contains the extension of the source code provided last week, an external tool for parameter estimation, and datasets for training/testing the parser. Place the archive file in the home directory, open a terminal¹ session, and perform the following steps:

```
$ cd ~
$ unzip graph-based-package.zip
$ cd graph-based-package
```

2 Training the model

As a training material, we use an official Universal Dependencies `en-partut-ud-dev.conllu` file.²³ To train our model, we use an external tool which requires that input data is stored in a separate directory. Each file in this directory corresponds to one (sentence, dependency tree) pair and contains the dump of the parsing steps performed by the parser while processing the sentence. To convert our training material into the appropriate format, first create the target directory:

```
$ mkdir train_data
```

and then perform the following command:

```
$ ./source/main.py dump -i ud-data/en-partut-ud-dev.conllu -o train_data -l 40 -v
```

The `-l 40` option specifies that sentences of length exceeding 40 should be ignored (i.e. will not be used for training).⁴ The `-v` option specifies that chart items are to be dumped in their full, human-readable form.

¹Search for the application called *terminal*.

²See https://github.com/UniversalDependencies/UD_English-ParTUT

³It was slightly modified for the sake of this tutorial – namely, tokenization ambiguities were removed.

⁴You can drop this option, but then the training process will be slightly slower. The resulting model, however, should be more accurate, since trained on more material.

```
$ cat train_data/1 | grep "^1"
```

```
1 (0,0,L)
```

```
...
```

```
1 (4,4,R)
```

```
1 (0,1,B) (0,0,L), (1,1,R)
```

```
1 (2,3,B) (2,2,L), (3,3,R)
```

```
1 (2,3,R) (2,2,R), (2,3,B) relationships->your
```

```
1 (1,3,B) (1,1,L), (2,3,R)
```

```
1 (1,3,L) (1,3,B), (3,3,L) deepen->relationships
```

```
1 (1,4,B) (1,3,L), (4,4,R)
```

```
1 (1,4,L) (1,4,B), (4,4,L) deepen->.
```

```
1 (0,4,L) (0,1,B), (1,4,L) deepen
```

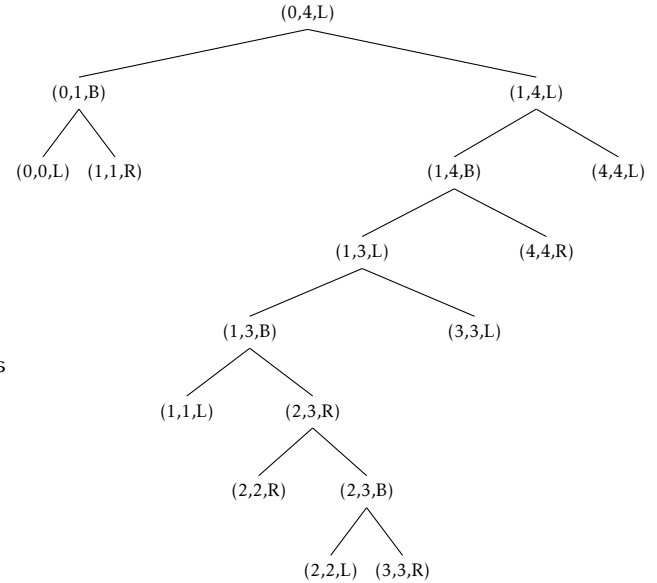


Figure 1: On the left, derivation tree encoded in the `train_data/1` dump. On the right, an alternative graphical representation of the same derivation tree (ignoring features).

After running the above command, have a look at the contents of `train_data/1`, which corresponds to the first sentence in `en.partut-ud-dev.conllu` (“*Deepen your relationships.*”). Each line in `train_data/1` consists of a tab-separated specification of a parsing step⁵:

- column 1: equal to 1 iff the step leads to the correct dependency tree
- 2: *conclusion* item, resulting from performing the parsing step
- 3: comma-separated *premise* items, required to perform the parsing step
- 4, 5, ...: *features* assigned to this parsing step

It is crucial to understand that `train_data/1` (and other files in `train_data`) includes both:

- The parsing steps needed obtain the correct dependency tree
- All the other steps that the parser will try to perform, even though they lead to incorrect dependencies

By providing the external parameter estimation tool with both “good” and “bad” examples, we allow it to learn which features are typically related to good steps (and should be assigned high weights), and which features are typically related to bad steps (and should be assigned low weights).

Getting back to `train_data/1`, if you filter the lines which contain 1 in the first column, you will obtain the derivation tree of the correct dependency tree, as shown in Fig. ?? . In particular, the following line:

```
1 (2,3,R) (2,2,R), (2,3,B) relationships->your
```

corresponds to the application of the *combine right* inference rule over chart items $(2,2,R)$ and $(2,3,B)$, which results in item $(2,3,R)$. Note the corresponding feature `relationships->your`, which tells that this parsing step creates a dependency arc from *relationships* to *your*.

Thanks to the fact that each file in `train_data` contains information about all the steps the parser *can* perform for a given sentence, as well as information about which of these steps actually *should* be

⁵Recall that a parsing step represents an application of an inference rule: *combine left*, *init left*, etc.

performed, it is possible to automatically estimate the weights of features in order to maximize the total weight of the correct derivations while keeping the weights of incorrect derivation trees relatively low.

To estimate the parameters, first repeat the dumping process without the `-v` (verbose) option:

```
$ ./source/main.py dump -i ud-data/en_partut-ud-dev.conllu -o train_data -l 40
```

and then run the parameter estimation procedure:

```
$ ./third-party/estimate-params sgd -t train_data -m model.txt -i 10
```

Training the model may take a couple of minutes.

Exercise 1. Is there any relation between the length of a sentence and the size of the corresponding dump file in the `train_data` directory?

2.1 Parameter estimation

WARNING: This subsection (??) summarizes the principles behind the parameter estimation processes implemented in the `estimate-params` tool. It is quite dense, but you can skip it during the first reading. We will get back to the question of parameter estimation on Thursday.

Let $\mathcal{F}(a)$ be the set of features assigned to a given parsing step a and $\theta_f \in \mathbb{R}$ be the parameter assigned to feature $f \in \mathcal{F}(a)$. Let also $\mathcal{D}(x)$ be the set of derivation trees that can be assigned to sentence x . Recall that each derivation tree corresponds to a particular dependency tree. Note also that the dump files created before encode (in a compact form) all the derivation trees for the individual sentences in the training dataset. The goal is to estimate parameters θ_f for the individual features f in our model.

The `estimate-params` tool adopts a *multiclass regression model* in which the probability of a derivation tree $d \in \mathcal{D}(x)$ given sentence x is defined as:

$$p(d|x) = \frac{\phi(d)}{Z(x)}, \quad (1)$$

where

$$\phi(d) = \exp\left(\sum_{\text{step } a \text{ in } d} \sum_{f \in \mathcal{F}(a)} \theta_f\right) \quad \text{and} \quad Z(x) = \sum_{d' \in \mathcal{D}(x)} \phi(d'). \quad (2)$$

Let T be a training dataset, i.e., a set of (sentence, correct derivation) pairs. Parameter estimation boils down to finding the *maximum likelihood* estimates $\hat{\theta}$ – parameters which maximize the likelihood $\ell(T)$ of the training dataset T :

$$\ell(T) = \prod_{(x,d) \in T} p(d|x). \quad (3)$$

In different terms, the goal of parameter estimation is to assign such parameters to features that the probability of the derivation trees (and, therefore, the corresponding dependency trees) in the training dataset is maximized. This is an optimization problem, which the `estimate-params` tool solves using *stochastic gradient descent*.

3 Parsing model

Once the parameter estimation process is over, have a look at the resulting `model.txt` file. Each line in this file specifies the parameter (weight) of a particular feature. Thus:

```
day->in    4.728
```

states that feature `day->in` has the weight 4.728. A consequence of this is that the weight of any dependency tree containing the arc `day->in` will be increased by 4.728.⁶

Exercise 2. Investigate the first and the last couple of lines in the `model.txt` file. Do they encode any linguistic generalizations one would expect the parser to learn from training data?

4 Parsing and evaluation

You can use the newly created model to parse sentences in the CONLLU format. For instance:

```
$ ./source/main.py parse -m model.txt \
-i ud-data/en_partut-ud-dev.conllu -o dev.system.conllu
```

will re-parse all the sentences in `ud-data/en_partut-ud-dev.conllu` according to the model and store the output sentences in `dev.system.conllu`. The tool uses the parsing algorithm defined in `source/eisner.py`, which we looked into last week.

To measure the quality of parsing, you can use the official CONLLU evaluation script, which can be downloaded from <http://universaldependencies.org/conll17/evaluation.html> (no need to download it, though, as it is already included in the tutorial package):⁷

```
$ ./evaluation_script/conll17_ud_eval.py ud-data/en_partut-ud-dev.conllu \
dev.system.conllu --verbose
```

We are interested in the two last lines which report UAS – *unlabeled attachment score* – and LAS – *labeled attachment score*. The LAS value will be always 0 in our case, since the parser only performs unlabeled parsing. The UAS value on the other hand, defined as the proportion of words for which the parser correctly assigned the head, should not be far from 100%, since we evaluate on roughly⁸ the same dataset as the one on which the model was trained.

Exercise 3. Are there any feature/value pairs which could be removed from the `model.txt` file without significantly changing the behavior of the parser? If so, remove them manually and verify that the evaluation results on `en_partut-ud-dev.conllu` are roughly the same.

Now, if we evaluate on another dataset – `en_partut-ud-test.conllu`, also included in the package – the score should be significantly lower:

```
$ ./source/main.py parse -m model.txt \
-i ud-data/en_partut-ud-test.conllu -o test.system.conllu
$ ./evaluation_script/conll17_ud_eval.py ud-data/en_partut-ud-test.conllu \
test.system.conllu --verbose
```

Exercise 4. Modify the `arc_features` function in the `source/feature.py` module so as to capture some linguistic generalizations. For instance, you may try to include features related to part-of-speech tags. Once you modify the source file, you will have to repeat all the steps described in this tutorial: convert training data, perform parameter estimation, and evaluate the resulting model on both `en_partut-ud-dev.conllu` (used for training) and `en_partut-ud-test.conllu`.

⁶Note that `day->in` is an atomic feature – simply a string – even though it is defined so as to make it clear that it represents a dependency arc from `day` to `in`.

⁷The version included in the package is slightly modified – namely, it allows dependency structures to have multiple roots.

⁸“Roughly”, because too long sentences were discarded.