NLP Assignment 2 Report

authors & contributors:
Petrus Kiendys, Amin Abu-Sneineh,
Oskar Berntorp, Azddin Bezberkan

Carrying out the experiments

Choose two (reasonable) sentences from the training set, and draw the tree structure by hand
Answer: We chose the following two sentences:

- "Individuell beskattning av arbetsinkomster" (line 1 from the files containing bracketing formatted phrase structure trees)
- "Kommunalskatteavdraget slopas" (line 17 from the files containing bracketing formatted phrase structure trees)

dependency trees
[insert diagram of sent1_deptree]
[insert diagram of sent2_deptree]

[cfg-pos] style trees
[insert diagram of sent1_cfg-pos]
[insert diagram of sent2_cfg-pos]

[cfg-dep] style trees
[insert diagram of sent1_cfg-dep]
[insert diagram of sent2_cfg-dep]

Does this algorithm do a good job of creating phrase-structure trees from a linguistic point of view?
Answer: It does not (or rather, the pseudocode describing it is too vague and does not define all operations that need to be executed in order to generate the same phrase-structure trees that are presented in the .cfg files). On the following images we can see sentence 1 being transformed from a dependency tree into a phrase-structure tree:

[insert diagram of sent1_deptree]
[insert diagram of sent1_cfg-dep]
[insert diagram of sent1_cfg-dep (via algo1)]

The following images show the general idea of how dependency trees are presented as phrase-structure trees according to Algorithm 1:

[insert algo1_01.png]
[insert algo1_02.png]

As we can see, by comparing the tree produced by algorithm 1 to the phrase structure trees of the bracketing formatted files, the algorithm is too simplistic and does a poor job of creating proper phrase structure trees.
It's important to note that phrase-structure trees adhere to constituency relations and dependency trees adhere to dependency relations. As such it is hard to transform a dependency tree to a phrase structure tree with simplicity.
Furthermore, the algorithm only specifies how nodes should be manipulated in the input dependency tree and does not specify how the dependency label that describe the relationship between each node should be presented on the phrase-structure tree. Simply adding a "P" suffix to a non-terminal node does not properly solve this issue as the label contains a word/terminal that was transformed from the dependency tree rather than a having a label consisting of a phrase (such as VB, NP, etc.).
As I explained earlier, some operations are not well-defined. The operation of "change node into a constituent node, and add a "P" suffix to its label" is what comes to mind. Does this operation entail taking the dependency relation label (placed on edges between vertices) and applying it as the value/label of the node?
Lastly, appending the SENT node/label for the whole tree is also omitted from the pseudocode.

With all of this said, despite the fact that the pseudocode might be too vague or incomplete to describe the process of obtaining phrase-structure trees (contained in the .cfg files) from dependency trees (contained in the .conll files), assuming that it does [describe the complete process], it does a pretty good job at creating phrase-structure trees.
The relationships between the nodes present in the dependency tree are still intact and present in the phrase-structure tree. Due to structuring nodes with constituents we can see that the phrase-structure trees are more fine-grained and as a result will produce more grammar rules.

Can you think of any improvements? Explain why this would be an improvement.
Answer: I'm not sure if there's any improvements that can be applied as far as to present a dependency tree in the structure of a phrase-structure tree.
If the sole goal of this task is to present a dependency tree as a phrase-structure tree then there is one "optimization" that comes to mind. Removing P-suffixed nodes/labels would produce a more coarse-grained phrase-structure tree that would bare more resemblance to a dependency tree as dependency trees are more compact/coarse-grained than constituency-based trees.
As a side-note this would reduce the amount of grammar rules generated by rule counters.
[insert diagram of sent1_cfg-dep (via algo1)]

How many different rules (grammar rules and lexicon rules) do extract for each parsing model?
Answer:
For parsing model dep we extracted 4489 grammar rules and 20618 lexicon rules.
For parsing model pos we extracted 3373 grammar rules and 13920 lexicon rules.

*Why do you think there such a big difference in number of rules between the two models (i.e., compare counts extracted from **[cfg-pos]** with those extracted from **[cfg-dep]**)?*
Answer: The key difference between generating grammar rules for the two different models is that parsing model dep always assigns a RTP label to the head word(**"root node" or head word?**) whereas parsing model pos assigns a part-of-speech tag. The predetermined construction of an RTP label for the root node(**"root node" or head word?**) will produce more grammar rules.
[**[explain diff in lexicon rules..]**]

Have a look at the output of the parser. Do you notices anything overtly incorrect?
Answer: Yes, there are a lot of sentences provided with test_input.txt that could not be parsed based on the grammar and lexicon rules that we trained BitPar on. These sentences/lines start with "No parse for: [SENT]" where [SENT] is the sentence that could not be parsed.

Evaluation

There is a lot of output of evalb. At the end of the output, is the overall summary and you are mainly interested in precision, recall and f1-measure. How do your parser models do?
Answer: [**insert answer**]

Error Analysis

Use the explanations on what the dependency and pos labels are for Talbanken. Consider the previous discussion in your report and try to explain the parse results (this should be at least a half page of explanations).
Answer: [**insert answer**]