Natuurlijke Taalmodellen & Interfaces
Tejaswini Deoskar
Part B (Step 2)

Lucas van Berkel, 10747958
Ruben Gerritse, 10760326
Gideon Mooijen, 10686290

1 Introduction

The goal of *Binarization* is to rephrase fragments in the treebank to CNF, *conjunctive normal form*, to make them suitable for the next step: *Markovization*. An example of binarization is given below. Input of the type:

(NP (NNP Rolls-Royce) (NNP Motor) (NNPS Cars) (NNP Inc.))

should be rewritten in the form:

(NP (NNP Rolls-Royce) (@NP→_NNP (NNP Motor)
(@NP→_NNP_NNP (NNPS Cars) (@NP→_NNP_NNP_NNPS (NNP Inc.)))))

After the treebank has been rewritten, Markovization is applied. Initially, vertical Markovization adds context to the nodes in the treebank. This context is necessary to correctly classify the nodes. As Klein and Manning stated[1]: *"a subject NP is 8.7 times more likely than an object NP to expand as just a pronoun. Having separate symbols for subject and object NPs allows this variation to be captured and used to improve parse scoring. One way of capturing this kind of external context is to use parent annotation, as presented in Johnson (1998). For example, NPs with S parents (like subjects) will be marked NP^S, while NPs with VP parents (like objects) will be NP^VP."*.

Where vertical Markovization adds depth regarding context, horizontal Markovization adds breadth. Depending of the scope, it adds context regarding the siblings of the node.

2 Methods

The code consists of several steps.

1. Read the text file and convert the sentences to lists
2. Perform vertical Markovization on all unbinarized sentences
3. Binarize all sentences (including the vertical Markovization)
4. Perform horizontal Markovization on all binarized sentences
5. Convert all binarized sentences to strings and write them to a text file

2.1 Converting the text file to lists

Before being able to apply binarization and Markovization, a usable representation of the sentences is needed. The received text file is read and every sentence is saved in *String* format. Those strings are to be converted to lists. Before a conversion, a couple of symbols need to be added and substituted. Every lexical unit in the sentence needs to be surrounded by double-quotes. Also, every node needs to be separated by a comma and the round brackets need to be replaced by square brackets. For example, the sentence:

[1] Klein, D., & Manning, C. D. (2003). *Accurate Unlexicalized Parsing*, Unpublished Manuscript, Stanford University, Stanford CA.

(NP (NNP Rolls-Royce) (NNP Motor) (NNPS Cars) (NNP Inc.))

will be represented as:

[”NP”, [”NNP”, ”Rolls-Royce”], [”NNP”, ”Motor”], [”NNPS”, ”Cars”], [”NNP”, ”Inc.”]]

By changing the String representation of the sentences, it is possible to convert the sentences to a list representation using the *literal_eval* function. After this conversion to lists, it is possible to apply binarization and Markovization on the sentences.

2.2 Vertical Markovization

With vertical Markovization, every node gets extra context history added to its name. The idea is that when extra information is given over the structure of the tree (not only the direct parent-child relation), a part of speech tagger will be capable of making better distinction between the better and the worse option. In this program, the user can decided to what degree the context history of a node should be added. The particular function that adds the context works recursively. When the entire sentence is offered as input, the function takes all children and adds its own name (of the node) to the name of the child, separated by a ”^”. If the number of ”^” exceeds the amount of vertical history the user wants to add, the eldest parents are cut off the name. From there the function calls itself, only to give the child as sentence-tree as input. The function terminates when the bottom of the (sub)tree is reached. This way every node is supplied with its history.

2.3 Binarization

The binarization function works, similar to the vertical and horizontal Markovization functions, recursively. The function takes the sentence as input and explores it depth-first. When the function finds a node with more than two children, after branching out the first child, the function will replace the second child until the last one with a new node with a new name, concatenated from its parent and sibling. From there, the first child of the new node is branched out, binarized, until the function returns to the unbinarized node. If there are still more than two children, the same principle will be applied. This way, the result will be a binarized sentence.

2.4 Horizontal Markovization

Horizontal Markovization is an algorithm which separates the nodes of the sentence-tree into two parts. Horizontal Markovization is about adding context information about the ’siblings’ of nodes in the tree. After the binarization two different nodes contained children, original nodes and created nodes. The algorithm explores the sentence depth-first, again by applying recursion, checking if a node is of the first or second order. When the node is an original one, there has to be checked if it has a sibling on its left side. If so, a new node is created, taking place between the second node and its parent, named after the parent and the sibling. Every first child is left untouched. When a node of the second order, so created during binarization, the name of the node is checked if the number of names of nodes after the ”->” does not exceed the maximum number of horizontal context information given. If so, the eldest siblings are cut off the name. Through recursion every node is visited once, resulting in a completely horizontalized sentence.

2.5 Converting the sentences to a text file

After binarization and Markovization, it is needed to write the sentences back to a text file. Therefore a conversion from list to String format is needed. This is achieved by executing the opposite of the steps explained in section 2.1.

3 Results

```
[Current]  P: 50.0 R: 57.14 F1: 53.33 EX: 0.0
[Average (up to 1034)]  P: 79.48 R: 73.73 F1: 76.49 EX: 20.4
[Average]  P: 79.48 R: 73.73 F1: 76.49 EX: 20.4
```

Figure 1: No Markovization, *H0V0*

```
Guess:
(ROOT
  (S
    (NP (EX There))
    (VP (VBD were)
      (NP (DT no) (JJ major) (NN Eurobond)
        (CC or)
        (NP (JJ foreign) (NN bond) (NNS offerings)))
      (PP (IN in)
        (NP (NNP Europe) (NNP Friday))))
    (. .)))

Gold:
(ROOT
  (S
    (NP (EX There))
    (VP (VBD were)
      (NP (DT no) (JJ major) (NNP Eurobond)
        (CC or)
        (JJ foreign) (NN bond) (NNS offerings))
      (PP (IN in)
        (NP (NNP Europe)))
      (NP (NNP Friday)))
    (. .)))

[Current]  P: 57.14 R: 57.14 F1: 57.14 EX: 0.0
[Average (up to 1034)]  P: 79.08 R: 72.11 F1: 75.43 EX: 16.63
[Average]  P: 79.08 R: 72.11 F1: 75.43 EX: 16.63
```

```
Guess:
(ROOT
  (S
    (NP (EX There))
    (VP (VBD were)
      (NP (DT no) (JJ major)
        (ADJP (JJ Eurobond)
          (CC or)
          (JJ foreign))
        (NN bond) (NNS offerings))
      (PP (IN in)
        (NP (NNP Europe) (NNP Friday))))
    (. .)))

Gold:
(ROOT
  (S
    (NP (EX There))
    (VP (VBD were)
      (NP (DT no) (JJ major) (NNP Eurobond)
        (CC or)
        (JJ foreign) (NN bond) (NNS offerings))
      (PP (IN in)
        (NP (NNP Europe)))
      (NP (NNP Friday)))
    (. .)))

[Current]  P: 57.14 R: 57.14 F1: 57.14 EX: 0.0
[Average (up to 1034)]  P: 81.89 R: 78.56 F1: 80.19 EX: 28.91
[Average]  P: 81.89 R: 78.56 F1: 80.19 EX: 28.91
```

Figure 2: Left: *H1V1*. Right: *H1V2*

```
Guess:
(ROOT
  (S
    (NP (EX There))
    (VP (VBD were)
      (NP (DT no) (JJ major) (NN Eurobond)
        (CC or)
        (JJ foreign) (NN bond) (NNS offerings))
      (PP (IN in)
        (NP (NNP Europe) (NNP Friday))))
    (. .)))

Gold:
(ROOT
  (S
    (NP (EX There))
    (VP (VBD were)
      (NP (DT no) (JJ major) (NNP Eurobond)
        (CC or)
        (JJ foreign) (NN bond) (NNS offerings))
      (PP (IN in)
        (NP (NNP Europe)))
      (NP (NNP Friday)))
    (. .)))

[Current]  P: 66.66 R: 57.14 F1: 61.53 EX: 0.0
[Average (up to 1034)]  P: 78.89 R: 72.9 F1: 75.78 EX: 16.53
[Average]  P: 78.89 R: 72.9 F1: 75.78 EX: 16.53
```

```
Guess:
(ROOT
  (S
    (NP (EX There))
    (VP (VBD were)
      (NP (DT no) (JJ major) (NN Eurobond)
        (CC or)
        (JJ foreign) (NN bond) (NNS offerings))
      (PP (IN in)
        (NP (NNP Europe) (NNP Friday))))
    (. .)))

Gold:
(ROOT
  (S
    (NP (EX There))
    (VP (VBD were)
      (NP (DT no) (JJ major) (NNP Eurobond)
        (CC or)
        (JJ foreign) (NN bond) (NNS offerings))
      (PP (IN in)
        (NP (NNP Europe)))
      (NP (NNP Friday)))
    (. .)))

[Current]  P: 66.66 R: 57.14 F1: 61.53 EX: 0.0
[Average (up to 1034)]  P: 81.38 R: 78.67 F1: 80.0 EX: 30.07
[Average]  P: 81.38 R: 78.67 F1: 80.0 EX: 30.07
```

Figure 3: Left: *H2V1*. Right: *H2V2*

The highest F1-score is achieved by *H1V2*, the highest percentage of exact matches is achieved by *H2V2*. Both *H1V1* and *H2V1* perform worse than the runs without Markovization.

4 Conclusion

As expected, the runs with the greatest Markovization perform best. Contradictory to intuition, the runs with little vertical Markovization, *H1V1 & H2V1*, perform even worse than the run without Markovization. The fact that the two runs with the highest vertical Markovization *(H1V2, H2V2)* have the highest F1-scores show that in this stage vertical Markovization has a greater influence than horizontal. Even *H1V2* has a higher F1-score than *H2V2*. This might give reason to consider declaring that horizontal Markovization offers no increase in performance. This is not the case, as it does increase the F1-scores for the cases where the vertical Markovization are 1. Secondly, there is a significant increase in the exact matches between *H1V2* and *H2V2*. In short: adding context, both vertically and horizontally, improves the performance of the POS-tagger, but the scope of the vertical Markovization must be greater than one.

5 How to run

```
BerkelGerritseMooijen_5.py -hor [value] -ver [value] -input [non-binarized] -output [binarized]
```