Finite state clause segmentation

Anders Berglund
Lund Technical Highschool
d98ab@efd.lth.se

Abstract

This report describes an attempt to perform clause segmentation in Swedish using a method published by Eva Ejerhed in (András, 1999) pp. 140-151.

1 Credits

The method I used is basically the method developed and published by Eva Ejerhed, the grammar I use is hers entirely and this very project wouldn't have been possible without her paper.

2 Introduction

Ejerhed's idea was to use a finite state automaton that triggers on certain patterns of word tags and inserts clause boundaries. The automaton is easy to specify and easy to implement, the algorithm isn't too computationally intensive and the results good, Ejerhed reports a degree of correctness of 96% with the method applied to manually tagged text.

3 Basic Overview

The finite state automaton in my implementation takes a tagged text as input, metaclassifies the tags (the automaton has a metatag "FIN" meaning a finite verb, all words classified as verbs being in a finite form gets metaclassified as "FIN") as a first step, then runs the rules given in the automaton using simple pattern matching (the automaton rules can be found in the appendix).

The metatags are quite few, around 10, and they do not overlap. As they are that few, a simple solution is giving each an integer value of its own and using those integers in the computations, making it possible to avoid string compares in the second step.

In the second step my pattern matcher simply runs *"are the next n tags (tag x tag y tag z)? then tag c after tag x"* in a long loop with different patterns of different length, advancing the automaton one step after each iteration of the loop.

4 Results

Eva Ejerhed remarks in her paper that there are many open questions concerning the definition of the clause units to have as targets for clause segmentation. One aspect she notes is whether a clause should have *at most* or *exactly* one finite verb per clause.

Eva Ejerhed chooses to have at most on finite verb per clause, and this leads to a clause segmentation that looks sometimes looks odd. An example of sentence from the *aa01* text from the SUC1A-corpus tagged with clause delimiters given by the state machine (quite thoroughly debugged, it *does* follow the rules):

```
<C1A> Don Kerr , en av de
politiska tänkarna pá det ansedda
analysinstitutet IISS , <C1C> är
inte páfallande optimistisk
<C2Bg> när han talar om saken .
```

The <C1C>-clause segmentation is strange

(<C1C> means a cl ause segmentation marker that was produced by rule 1C). In my book the entire sentence is a single clause (at least the part before the <C2Bg>-tag (the entire part before the <C2Bg> constitutes the subject of the phrase)), the automaton "overrecognizes" when it encounters something that fits rule 1C. A solution might be having a "flag" in the automaton that keeps track of whether the preceding stretch of words starting with the last DL MID-tag contained any verb, if not, then rule 1C shouldn't be applied. This flag would be set to *false* at every encounter of a delimiter and set to true when encountering a verb.

Such a change would mean a change of focus to having *exactly* one finite verb per clause. Eva Ejerhed reports very few *overrecognized* clauses, using a manually annotated corpus (which the SUC1A-corpus is) she finds *zero* overrecognized clauses. The Swedish construct above, what in (at least) Swedish grammatic terminology for German is called an *apposition* [1], is quite common, and I think it has been overrecognized in this case.

This is directly at odds with Ejerhed's results (*she*: zero overrecognizations, *I*: common construct overrecognized), but I'm reluctant to take a stand and I choose to leave the question of who is right as an exercise to the reader.

5 Conclusion

The complaint above apart, the method works very nicely for most cases, is easy to implement (and debug!) and not very computationally expensive. The performance attained with such simple functionality is quite impressive.

References

András Kornai (Ed.) 1999. *Extended finite state models of language*. Cambridge University Press.

[1] An example of an apposition: *Berlin, the German capital, is big*. The string *the German capital* is an attribute to the preceding noun *Berlin*, an attribute without a verb written between commas. As there is no verb it is not a clause of its own, and it should be treated as were it an adjective on the top-level.

Appendix: The automaton rules from Ejerhed (András, 1999) pp. 140-151

Clause segmentation rules

```
1 PUNCTUCATION
1a1) <h>XX -> <h><c>XX
1a2) <p>XX -> <p><c>XX
1b) DL-MAD XX -> DL-MAD <c> XX,
where XX is not end tag
1c) DL-MID FIN -> DL-MID <c> FIN
1d) DL-MID XX FIN -> DL-MID <c>
XX FIN, where XX=PN, NN, PM or AB

2 COMPLEMENTIZERS
2as) XX KN SN -> XX <c> KN SN
2ag) XX SN -> XX <c> SN
2bs) XX KN HX -> XX <c> KN HX
2bg) XX HX -> XX <c> HX

3 KN + FINITE VERB
3s) XX KN FIN -> <c> XX KN FIN,
where XX is a closed class of
finite forms of the verbs be, go,
stand, sit
3g) XX KN FIN -> XX <c> KN FIN

4 KN + XX + FINITE VERB, where
XX=PN, NN, PM or AB
4s) YY KN XX FIN -> <c> YY KN XX
FIN, if YY=XX
4g) YY KN XX FIN -> YY <c> KN XX
FIN, if YY!=XX

5 SEQUENCES OF FINITE VERBS
5a) CASE: 0 WORDS BETWEEN FINITE
VERBS
FIN FIN -> FIN <c> FIN
5b) CASE: 1 WORD BETWEEN FINITE
VERBS
FIN XX FIN -> FIN XX <c> FIN
5c) CASE: 2 WORDS BETWEEN FINITE
VERBS
5cs) FIN YY XX FIN -> FIN YY <c>
XX FIN, where XX=PN, NN or PM
5cg) FIn YY XX FIN -> FIN YY XX
<c> FIN
```

ABBREVIATIONS
<h> = head
 = paragraph
</h> = end head
 = end paragraph
DL MAD = major delmiter (.?!)
DL MID minor delimiter (,-:)
FIN = finite verb; VB PRS AKT, VB PRS SFO, AB PRT AKT, VB PRT SFO, VB SUP AKT, VB SUP SFO, VB IMP AKT
PN = PN ... SUB, PN ... SUB/OBJ (subject forms of pronouns)
NN = NN ... NOM (nouns)
PM = PM NOM (proper nouns)
AB = AB, AB POS, AB KOM, AB SUV (adverbs)
KN = conjunction
SN = subjunction
HX = HA, HD ..., HP ..., HS ... (Wh: adverbs, determiners, pronouns, possesives)