Programming exercise: A machine-learning algorithm for co-reference resolution

Ina Rösiger, Arndt Riester

Universität Stuttgart
Institut für maschinelle Sprachverarbeitung

Preamble

- This programming exercise is based on the paper by Soon et al. (2001): A Machine Learning Approach to Coreference Resolution of Noun Phrases
- It is meant as a chance for you to better understand the different steps that are involved in co-reference resolution, not as a coding competition
- You can code in Python or Java (please comment a lot!)
- You can do the exercise alone or in a team of two. If you choose to do it in a team, please specify who is responsible for which sections. Please do not copy code written by other teams.
- Note that we are not working with the same corpus as in the original paper. Some details that are given in the paper (e.g. on markables, etc.) may not be applicable in our case.
- Solutions for step 1 (markable extraction) and step 2 (feature extraction) will have to be handed in by everyone and will not be graded
- Work on further steps only if you choose the programming exercise for your final grade
- For those doing the whole programming exercise:

- Please document the steps you implemented and the results you achieved in a separate, short text document, giving details on challenges and problems you came across as well as possible solutions (2-3 pages)
- Not every step in the implementation has to be perfect
  if you encounter difficulties, just describe them
  the mark you'll get will be based on both implementation and presentation of results
- You might have to present your ideas/difficulties/solutions in a very short talk (depending on how much time we've got left)
- Steps marked with *bonus* are optional (you should do them after your pipeline is finished when there is time left ...)
- When you hand in the exercise: please, create a README that gives instructions on how to use your script(s) –commands, arguments, etc. Please create a folder *output* that contains the output of every step (the required outputs for every step are described in the different sections). We should be able to use your scripts on IMS linux machines in order to check and compare results.
- **Deadline: to be specified**

1 Determination of Markables

Aim: Implement a module that extracts markables
Output: List of markables for every document

- Download the corpus from ILIAS: the corpus contains coreference clusters (in the last column) as well as gold annotations (parses, POS tags, named entities, etc.)
- Have a look at the corpus format and familiarise yourself with the annotation levels
- Markable extraction: have a look at the corpus to see which entities have been marked as coreferent. Please document which entities you extract from which annotation level. Typically this involves:
  - Parses: to extract noun phrases
  - POS tags: to extract personal and possessive pronouns (only if not already marked as NP in the parse: you should check this)

  - Named entity recognition: to extract organisation, person, location, date, time, money, and percent entities

- Nested noun phrase extraction: check if needed at all, depends on the parser and the gold annotations.
  In Soon et al. 2001 they added these nested noun phrases:
  his long-range strategy and **Eastern's** parent,
  wage reductions, **Union** representatives added as markables

- Think about which information you want to add to your list of markables: you will have to extract further information on your markables in the following steps, so you should be able to find your markable again in the CoNLL document

- *Bonus*: Evaluation of markables: can you compute recall and precision for your markable extraction module?

2 Generating Training Examples and Feature Vectors

Aim: Generate training examples for the classifier (positive/co-referent ones and negative/non-coreferent ones)
Output: List of positive and negative training examples (mention pairs) together with their features

- Create positive training instances for the training document as described in the paper

- Create negative training instances for the training document as described in the paper

- For all training instances (mention pairs): create feature values
  (you do not have to implement all features, just choose 6-8 features, some of them should be single features that only describe one mention, some of them should be pair features that compare two mentions)

  - distance of mention i and j in sentences
  - antecedent is a pronoun (personal, possessive or reflexive)
  - anaphor is a pronoun (personal, possessive or reflexive)

- string match (after removing articles and demonstrative pronouns)
  this license matches the license, that computer matches computer
- anaphor is definite noun phrase
- number agreement
- both proper names?
- alias feature: one is an alias of the other or vice versa
  - person: compare last names: Mr. Simpson and Bent Simpson
  - organisations: acronym match: IBM and INternational Business Machines Corp.
- semantic class agreement: "female," "male," "person," "organization," "location," "date," "time," "money, percent," and "object."
  - female and male as subclasses of person
  - all other classes subclasses of object
  - classes mapped to WordNet synsets
  - semantic class of markable first sense of the head noun of the markable
  - agreement if one is the parent of the other or they are the same
- gender agreement: true, false, unknown
- (if needed at all): appositive feature: if one is in apposition to the other
- *Bonus*: can you think of another (new) feature that could be helpful for the task?

3 Generating Test Pairs and Feature Vectors

Aim: Generate test pairs for the classifier
Output: List of test mention pairs together with their features

- For every markable in the test document (starting from the second markable): create anaphor-antecedent pairs
- *Bonus* Exclude nested markables: if anaphor is child or nested markables, then the antecedent cannot be a markable with the same root
  Mr. Tom's daughter ... **his** daughter's eyes
  → His daughter and his daughter's eyes are not possible antecedents for his, only Mr. Tom or Mr. Tom's daughter
- For each pair: create feature values

4 Classification

Aim: decision tree learns from training examples and applies learned rules on the test data
Output: list of classified test instances

We give example commands for the machine learning platform Weka, but it is up to you which machine learning system or classifier you want to use (there are lots of systems available: Mallet, Stanford classifier, etc.)

Weka: http://www.cs.waikato.ac.nz/ml/weka/
Mallet: http://mallet.cs.umass.edu/
Both systems have reasonable documentation online!

- Create the right format for your classifier
  Weka: ARFF format:

```
//header
@relation coreference

//Define the single attributes: list values or specify type
//here are some examples
@attribute 'distance' numeric
@attribute 'string match' {true,false}
@attribute 'number' {Sin,Plu,Unknown}
@attribute 'genderForPronouns' {Unknown,Masc,Fem,Neut}
@attribute 'semanticClassForPronouns' {Unknown,Female,Male,Person}
@attribute 'class' {coref,non-coref}

//Followed by data, one instance per line, features in order as specified abo
@data
4,true,Sin,Masc,Male,coref
2,false,Plu,Fem,Female,non-coref
```

  Mallet format: one instance per line, tab separated columns

```
name-of-instance1 coref feature1:value feature2:value feature3:value
name-of-instance2 non-coref  feature1:value feature2:value feature3:value
```

- Use classifier to learn from the examples and to annotate the test instances
  you do not have to use decision trees, you can choose any ML you like

Example command for Weka J48 tree (C4.5 clone):

```
java -cp "./weka.jar" weka.classifiers.trees.J48
-t <training-file.arff> -T <test-file.arff> -c last
```

- Post-processing: for each anaphor: start from the immediately preceding markable and proceed backward until an antecedent is found
  → First pair that is considered coreferent is chosen

5 Evaluation

- Report on the performance that your classifier has achieved:
  CoNLL score: when you convert your system output to CoNLL format (the same format as the gold annotations), you can use the evaluation scripts (and compute the popular CoNLL score): http://conll.cemantix.org/2012/software.html

  What is the CoNLL score for your coreference resolution system?

6 *Bonus:* Automatic pre-processing

Aim: Enrich text with automatic annotations, repeat steps 1-5:
how does the performance of your classifier change?
Output: Pre-processed corpus and new evaluation results

- Pre-processing of the texts
  - part of speech (POS) tagging
  - constituency parsing
  - named entity recognition
  - morphological analysis (you can use a morphological analyzer or use POS tags to derive number information (lookup lists for pronouns, as the POS tags of pronouns do not contain morphological information)
  - *Bonus*: semantic class determination
- Replace gold annotations with automatic annotations

- Repeat steps 1-5: how does performance change?

It is up to you which tools you want to use for the pre-processing. (please mention them in your documentation). Here are some suggestions:

- Stanford Core NLP: (command-line or Java), http://stanfordnlp.github.io/CoreNLP/
- NLTK (python): http://www.nltk.org/
- IMS-internal tools
  - POS tagging: treetagger (in /corpora/cmd)
    tree-tagger-english (Input: one word per line, Latin 1 encoding)
  - Constituency parsing: BitPar (in /corpora/cmd)
    parse-english (Input: one word per line, sentences terminated by an empty line, Latin 1)
- A few others: UIMA, LingPipe, ...