NLP coursework II: n-grams

November 20, 2013

1 General

This coursework is worth 10% of the marks for the final module.

The deadline is Monday 2nd of December, 10 o'clock. Electronic submission.

2 Task definition

Compounds can be a trigram, i.e. a sequence of 3 nouns. Examples are *computer science lecturers, backup compiler disk, speech recognition system, exhibition ballroom dancer*. There are in theory two possible parses/interpretations for each trigram compound:

- Left-Branching (L): [[N1 N2] N3]
- Right-Branching (R): [N1 [N2 N3]]

The correct parse depends on the meaning of the compound. Thus, for example, the correct parse for *computer science lecturers* is left-branching [[computer science] lecturers] as the meaning of the compound is *lecturers in computer science*. In contrast, the correct parse for *exhibition ballroom dancer* is right branching [exhibition [ballroom dancer]] as the dancer is both an exhibition dancer and a ballroom dancer (and not in any way a dancer in exhibition ballrooms). Please copy all the files in `/home/csunix/scsnlp/compounds/` into your home directory to start.

A list of 244 trigram compounds extracted from newspaper text is given in the file `lauer.test`. Have a look at this file to see some more examples and make sure you understand the problem.[1]

Your task is to develop **unsupervised** n-gram based algorithms for compound bracketing. The algorithms should take as input the file `lauer.test` and output a file of the format in `lauer.gold` where L stands for left-branching and R for right-branching.

Your submission must be a single zip file with one report file `compound.pdf` of maximum 7 page length (if you do both questions, 5 pages if you only do Question 1) and other files as described in the following questions.

All submissions will be monitored for possible plagiarism. Any plagiarised report or program will be assessed as a fail mark and brought to the attention of the Director of Learning and Teaching.

2.1 Question 1 (20 marks)

2.1.1 Task

Develop an unsupervised algorithm for compound bracketing based on n-gram counts in a corpus.

The knowledge sources that your algorithm is allowed to use are

- The Brown corpus and NLTK

[1] We thank Mirella Lapata and Mark Lauer for making that list available to us. Some compounds occur more than once as they occurred more than once in the original texts.

- Thesauri like Roget's thesaurus or WordNet (although this is not necessary for the simplest algorithms)
- Literature on this topic as long as this is cited.

You are not allowed to use the following resources for algorithm development:

- The gold standard solutions in `lauer.gold` as the algorithm is supposed to be unsupervised
- Parsed corpora like the Penn Treebank
- Parsers

Implement your algorithm on the Brown corpus and run it on `lauer.test`. Evaluate performance using standard evaluation measures for classification tasks in comparison to the solutions in `lauer.gold`. For further evaluation, compare your algorithm to the expected performance of an unsupervised baseline that assigns right or left bracketing randomly and to a (supervised) baseline that always assigns the most frequent correct bracketing as given in `lauer.gold`. Discuss all design choices, the main problems your algorithm faces, both qualitatively (giving examples) and quantitatively, and discuss ideas for improvement (improvements need not be implemented).

2.1.2 Some implementation pointers

I will not give you hints on the algorithm development but just on some commands that allow you to count bigrams/trigrams in NLTK which you will need whatever n-gram-based method you develop.

The corpus needs to be imported via

```
>>> from nltk.corpus import brown
```

You can simply construct all bigrams in the Brown corpus by accessing

```
>>> from nltk import FreqDist
>>> from nltk.corpus import brown
>>> from nltk import bigrams
>>> thebigrams = bigrams(list(brown.words()))
>>> fdist=nltk.FreqDist(thebigrams) ##build a frequency distribution of bigrams
>>> fdist['big','car']   ## see how often "big car" occurred in the Brown corpus
```

thebigrams will now contain a list of all different bigram types in the Brown corpus (just as we in the first lab sheet constructed unigrams (vocabularies). Similar commands exist for other n-grams. You can then use `FreqDist` (which you already know) to get individual bigram counts etc.

2.1.3 Submission and Mark Scheme

A program `compound.ext` as well as a TXT file `compound.results` which contains the algorithm's decision on each line in the same format and order as `lauer.gold`. A section in your report describing and discussing your algorithm and its results. This section has a maximum of 5 pages.

Your submission will be assessed on

- Algorithm design and discussion (7 marks)
- Correct implementation of the algorithm (5 marks)
- Evaluation and discussion of algorithm results, including links to topics discussed in the module and/or the literature, evaluation of resources used etc. Comparison to lower bound. (8 marks)

2.2 Question 2: optional/formative, non-credit

Explore how your algorithm without any design changes would fare on a bigger corpus. The Web1T n-gram corpus at `/home/csunix/scsnlp/Web1T_5-gram/current/data` gives you precomputed n-gram frequencies from a 1 trillion web corpus sorted into unigram, bigram etc directories. For example

```
grep -P ``big car\t'' 2gm-0021
```

yields 42683 hits for the occurrence of "big car" on the portion of the web indexed by this corpus.

The best option is to run your whole algorithm for all examples on the google n-gram corpus but please note that the (only) programming problem here is that using a grep on all n-gram files will take very very long as they are huge so to run all your examples you will need to access the n-gram tables efficiently. As a hint, lower-cased bigrams start in 2gm-0021 and the file 2gm.idx tells you the starting bigram for each file.

Should you struggle with programming, then instead choose (intelligently) some examples that your Brown corpus algorithm could not handle for various reasons (10-15 examples) and simulate your algorithm on the Google n-gram corpus for these examples (using grep or similar).

Submission: a TXT file `compound2.results` which contains your algorithm's decisions on each line in the same format and order as `lauer.gold` for all examples or hand-simulated results for your chosen problematic examples. A section in your report describing and discussing your results and the impact of the changed corpus (max 2 pages).