Segmentation and tokenisation

Segmentation

Extract 50 paragraphs of text at random from a Wikipedia dump of a language of your choice and compare two sentence segmenters. You can choose any two segmenters you like (including segmenters you've written yourself!).

Data

You will need the file called `-pages-articles.xml.bz2`. You can find it on the WikiMedia dumps site (http://dumps.wikimedia.org%5D). Choose an XXwiki or YYYfolder, where XX is the 2-letter language code or YYY is the 3-letter language code.

To extract the text, you can use WikiExtractor (https://github.com/apertium/WikiExtractor). Then use the segmenters to segment the raw text output into sentences.

Suggested segmenters

pragmatic segmenter (https://github.com/diasks2/pragmatic_segmenter)

Requires ruby, which you should install using your package manager. To install `pragmatic_segmenter`, do the following:

```
$ git clone https://github.com/diasks2/pragmatic_segmenter
```

This will check out the code.

```
$ cd pragmatic_segmenter/lib
```

You will need to put this code into a file, call it `segmenter.rb`:

```
require 'pragmatic_segmenter'

lang = "en"
if ARGV[0]
    lang=ARGV[0]
end

STDIN.each_with_index do |line, idx|
    ps = PragmaticSegmenter::Segmenter.new(text: line, language: lang)
    ps.segment
    for i in ps.segment
        print(i,"\n")
    end
end
```

You can then run it like this:

```
$ echo "This is a test.This is a test. This is a test." | ruby -I .
segmenter.rb
This is a test.
This is a test.
This is a test.
```

If you get errors like `cannot load such file -- unicode (LoadError)`, then you need to install `ruby-unicode`, on apt-based systems, you can use:

```
$ sudo apt-get install ruby-unicode
```

NLTK's Punkt (https://kite.com/python/docs/nltk.tokenise.punkt)

Install with your package manager; Debian and Ubuntu `sudo apt-get install python3-nltk`.

For example usage, see this tutorial (https://textminingonline.com/dive-into-nltk-part-ii-sentence-tokenise-and-word-tokenise) starting at "How to use sentence tokenise in NLTK?"

Report

The comparison should include:

- Brief description of each segmenter used.
- Quantitative evaluation: Accuracy percentage (how many sentence boundaries were detected correctly)
- Qualitative evaluation: What kind of mistakes does each segmenter make?

Tokenisation

First download the UD treebank for Japanese (`UD_Japanese-GSD`) from the Universal Dependencies (http://github.com/UniversalDependencies/) GitHub repository. Then implement the left-to-right longest-match algorithm (also known as `maxmatch`). For a description of the algorithm see Section 3.9.1 in Jurafsky and Martin.

Hint: you might write a python program which

- reads sentences to tokenise from standard input
- reads the dictionary from a file specified as an argument;
- writes each word separated by newlines to standard output

so tokenisation looks like:

```
$ cat > dictionary-file
sentence
to
tokenise
^D

$ echo 'sentence to tokenise.' | python maxmatch.py dictionary-file
sentence

to

tokenise
.
```

Procedure:

- Extract a dictionary of segmented surface forms from `UD_Japanese-GSD/ja_gsd-ud-train.conllu`. The dictionary should contain 15,326 forms.
- Test how well the left-to-right longest match algorithm is capable of segmenting the text in `UD_Japanese-GSD/ja_gsd-ud-test.conllu` To perform the evaluation, you can use some Word Error Rate calculating code that you find online, e.g.
  - WER in Python (https://github.com/zszyellow/WER-in-python)
  - apertium-eval-translator (http://svn.code.sf.net/p/apertium/svn/trunk/apertium-eval-translator/)

If you have time, test the algorithm with other treebanks for languages which do not use word separators, e.g. Chinese, Thai.

Report

Submit:

- your implementation of maxmatch,
- instructions on how to use it,
- brief description of its performance, with examples to support your findings