НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ

Tokenisation and word segmentation

Francis M. Tyers

ftyers@hse.ru
https://www.hse.ru/org/persons/209454856

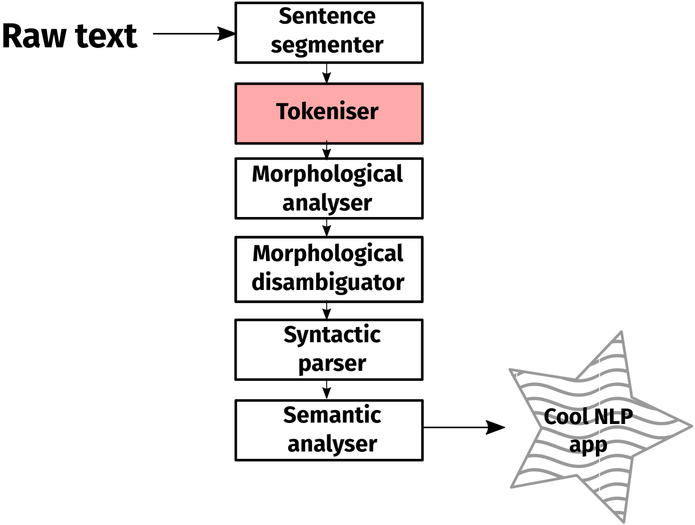
Национальный исследовательский университет
«Высшая школа экономики» (Москва)

1 октября 2018 г.

Introduction

- Sentences

Pipeline



Raw text

Sentence segmenter

Tokeniser

Morphological analyser

Morphological disambiguator

Syntactic parser

Semantic analyser

Cool NLP app

Token separators

ARMA·VIRVMQVE·CANO·TROIAE·QVI·PRIMVS·AB·ORIS
ITALIAM·FATO·PROFVGVS·LAVINIAQVE·VENIT
LITORA·MVLTVM·ILLE·ET·TERRIS·IACTATVS·ET·ALTO
VI·SVPERVM·SAEVAE·MEMOREM·IVNONIS·OB·IRAM

- ...
- Vast majority of languages use some kind of whitespace-based word separator

But what is a token?

Some questions:

- Multiword expressions
  - *только что* or *только·что* ?
- Named-entities
  - *Нижный Новгород* or *Нижный·Новгород* ?
- Numeral expressions
  - *150 000,0* or *150·000,0*

And how about abbreviations:

- *Гипотеза была выдвинута Каролем Борсуком в 1933 г.*
  - Is there one '.' here or two?
- *В 1933 г. гипотеза была выдвинута Каролем Борсуком.*
  - And here?

More questions

Clitics:

-

Tasks

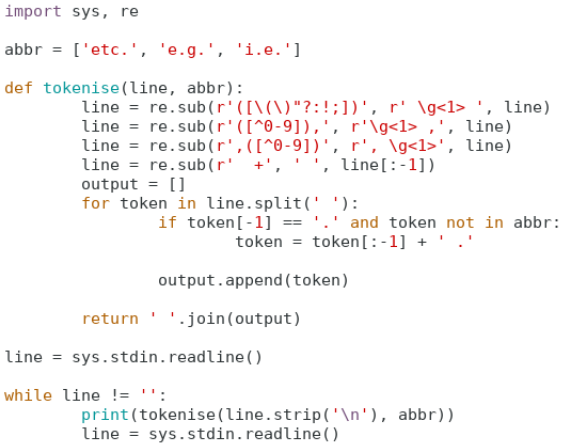
The ideal tokenisation may depend on the task.[1]

- Russian–Arabic MT:
    - Split off clitics[2]

[1] And also on the language (pair)!

[2] Zalmout and Habash (2017) “Optimizing Tokenization Choice for Machine Translation across Multiple Target Languages”

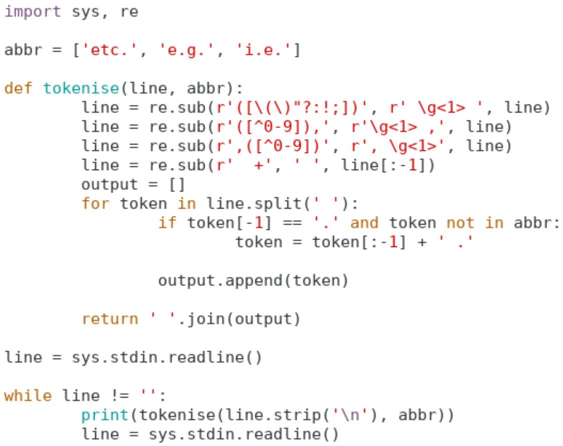
Approaches



```
import sys, re

abbr = ['etc.', 'e.g.', 'i.e.']

def tokenise(line, abbr):
        line = re.sub(r'([\(\)"?:!;])', r' \g<1> ', line)
        line = re.sub(r'([^0-9]),', r'\g<1> ,', line)
        line = re.sub(r',([^0-9])', r', \g<1>', line)
        line = re.sub(r'  +', ' ', line[:-1])
        output = []
        for token in line.split(' '):
                if token[-1] == '.' and token not in abbr:
                        token = token[:-1] + ' .'

                output.append(token)

        return ' '.join(output)

line = sys.stdin.readline()

while line != '':
        print(tokenise(line.strip('\n'), abbr))
        line = sys.stdin.readline()
```

- Split off always-separating punctuation

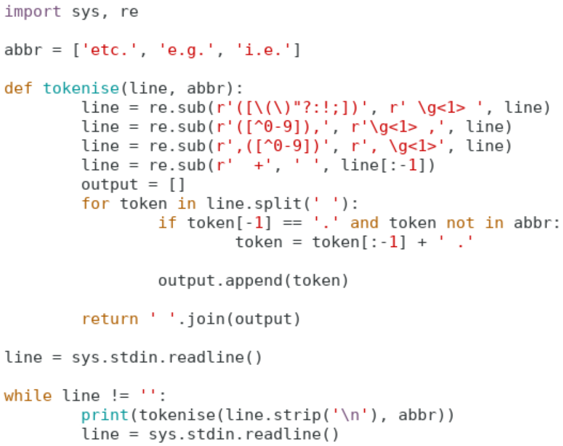
Approaches



```
import sys, re

abbr = ['etc.', 'e.g.', 'i.e.']

def tokenise(line, abbr):
        line = re.sub(r'([\(\)"?:!;])', r' \g<1> ', line)
        line = re.sub(r'([^0-9]),', r'\g<1> ,', line)
        line = re.sub(r',([^0-9])', r', \g<1>', line)
        line = re.sub(r'  +', ' ', line[:-1])
        output = []
        for token in line.split(' '):
                if token[-1] == '.' and token not in abbr:
                        token = token[:-1] + ' .'

                output.append(token)

        return ' '.join(output)

line = sys.stdin.readline()

while line != '':
        print(tokenise(line.strip('\n'), abbr))
        line = sys.stdin.readline()
```

- Split off commas not part of numeral expressions

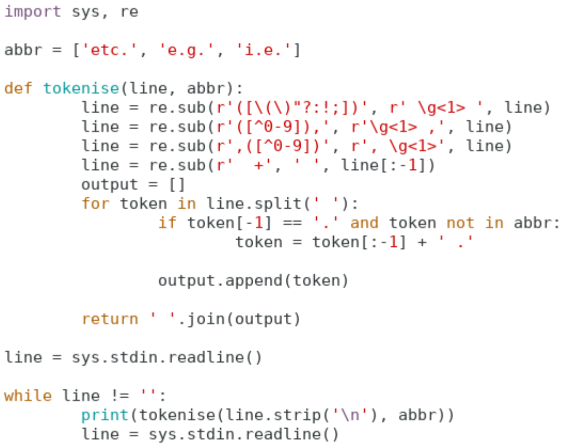
Approaches



```
import sys, re

abbr = ['etc.', 'e.g.', 'i.e.']

def tokenise(line, abbr):
        line = re.sub(r'([\(\)"?:!;])', r' \g<1> ', line)
        line = re.sub(r'([^0-9]),', r'\g<1> ,', line)
        line = re.sub(r',([^0-9])', r', \g<1>', line)
        line = re.sub(r'  +', ' ', line[:-1])
        output = []
        for token in line.split(' '):
                if token[-1] == '.' and token not in abbr:
                        token = token[:-1] + ' .'

                output.append(token)

        return ' '.join(output)

line = sys.stdin.readline()

while line != '':
        print(tokenise(line.strip('\n'), abbr))
        line = sys.stdin.readline()
```

- Collapse multiple spaces

Approaches



```python
import sys, re

abbr = ['etc.', 'e.g.', 'i.e.']

def tokenise(line, abbr):
        line = re.sub(r'([\(\)"?:!;])', r' \g<1> ', line)
        line = re.sub(r'([^0-9]),', r'\g<1> ,', line)
        line = re.sub(r',([^0-9])', r', \g<1>', line)
        line = re.sub(r'  +', ' ', line[:-1])
        output = []
        for token in line.split(' '):
                if token[-1] == '.' and token not in abbr:
                        token = token[:-1] + ' .'

                output.append(token)

        return ' '.join(output)

line = sys.stdin.readline()

while line != '':
        print(tokenise(line.strip('\n'), abbr))
        line = sys.stdin.readline()
```

- Split of full stops not part of abbreviations

Caveats

- Let’s meet at 17:45
- No one uses Yahoo! any more

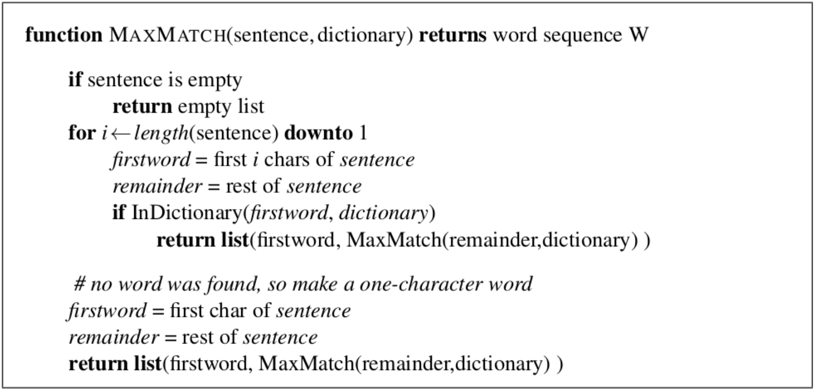
Space-free languages

- Some languages are written without spaces
- Tokenisation more difficult
- A number of algorithms available

Max-match algorithm

- Rule-based algorithm
- Requires some kind of lexicon/dictionary
    - From a corpus
    - From a wordlist

Max-match algorithm/2



```
function MAXMATCH(sentence, dictionary) returns word sequence W

    if sentence is empty
        return empty list
    for $i \leftarrow length$(sentence) downto 1
        firstword = first $i$ chars of sentence
        remainder = rest of sentence
        if InDictionary(firstword, dictionary)
            return list(firstword, MaxMatch(remainder,dictionary) )

     # no word was found, so make a one-character word
    firstword = first char of sentence
    remainder = rest of sentence
    return list(firstword, MaxMatch(remainder,dictionary) )
```

- Start at beginning of string
- Iteratively look up the longest word in the dictionary
- If no word is found, output a single character

Caveats

wecanonlyseeashortdistanceahead
we canon l y see ash ort distance ahead

– Alan Turing

- Works pretty well for some languages (e.g. Chinese)
- Not so great for others
- Why? Length of words

Evaluation

Word segmentation systems can be evaluated using Word Error Rate (WER):

- Edits (insertions, deletions, substitutions)

wecanonlyseeashortdistanceahead
Ref: we can only see a short distance ahead
Test: we canon l y see ash ort distance ahead
WER =

Viterbi approach

Language model

- Downside: requires an existing tokenised corpus

Graph-based word segmentation

Decoding

A note on encodings

Watch out for characters other than space (U+0020):

- Non-breaking spaces (U+2060, U+FEFF, …)
- Soft-hyphen (U+00AD)
- En quad (U+2000)
- En space (U+2002)

Plus 20 or so other characters.

Practical

Implement the `maxmatch` algorithm and test it on Japanese:

- Extract surface form dictionary from the training corpus
- Run the algorithm with the dictionary on the test corpus
- Write script to calculate WER for the segmentation.
  - Feel free to use a library for this

https://github.com/UniversalDependencies/UD_Japanese-GSD