НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Tokenisation and word segmentation

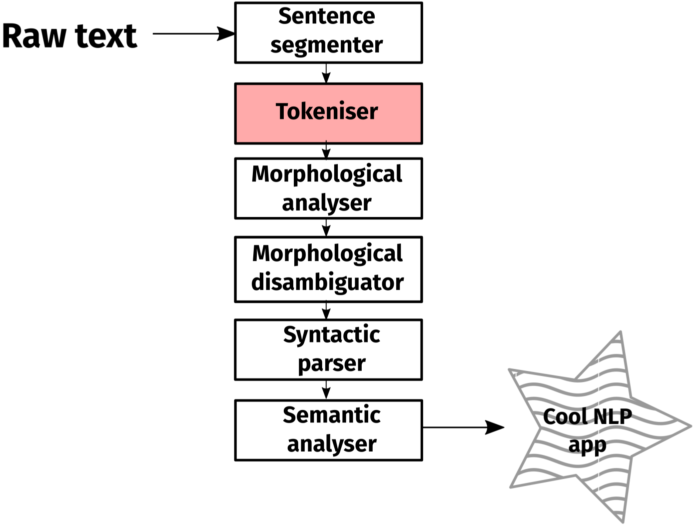
Francis M. Tyers

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

Национальный исследовательский университет
«Высшая школа экономики» (Москва)

13 октября 2018 г.

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

- Peoplecangenerallyreadsentencesevenifthereisnowhitespace, but:
  - The vast majority of languages use some kind of whitespace-based word/token 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?

Clitics:

- Romance: *explicándoselo* [explicándo se lo], *j'ai* [j' ai]
- Germanic: *don't* [do n't], *I'm* [I 'm], *Nastya's* [Nastya 's]
- Slavic: *mógłbym* [mógł bym], *razumeću* [razume ću]
- Finnic: *voisiko* [voisi ko]
- Turkic: *чыгачакмы* [чыгачак мы]

Can be ambiguous:

- Nastya's cool → Nastya is
- Nastya's broke the record again → Nastya has
- Nastya's cat is cute → Nastya's

Is resolving this the job of a tokeniser ?

Tasks

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

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

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

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

A simple approach

```
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

A simple approach

```
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

A simple approach

```
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

A simple approach

```
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

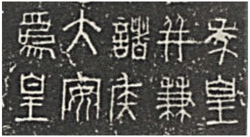
Caveats

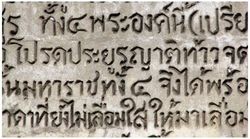
- Time and numeral expressions:
  - Let's meet at 17:45
- Proper names:
  - I'm surprised that Yahoo! are still solvent
  - Therapy? is one of the best bands from Northern Ireland
- Emoticons:
  - haters gonna hate ;____;

Machine-learning approaches

@@@

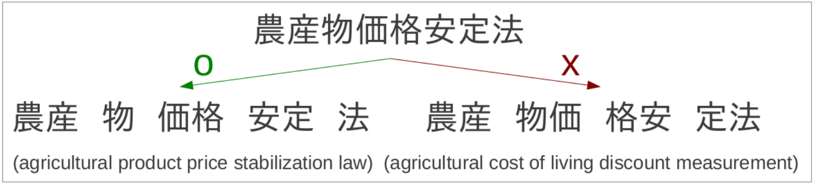
Space-free languages





- Some languages are written without spaces
- Tokenisation more difficult — space is a strong signal
- Issue of ambiguity — more than one possible interpretation
- A number of algorithms available

Ambiguity



- Many analyses for each sequence
- How do we choose the correct one(s) ?

Graphics by Graham Neubig (NAIST)

(A tale of) two algorithms

Maxmatch:

- Rule-based algorithm
  - Requires some kind of dictionary — from wordlist or corpus

Graph-based:

- Statistical algorithm
  - Requires pre-segmented corpus

Maxmatch algorithm

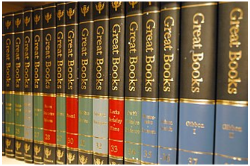
```
function MAXMATCH(sentence, dictionary) returns word sequence W

    if sentence is empty
        return empty list
    for i ← 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

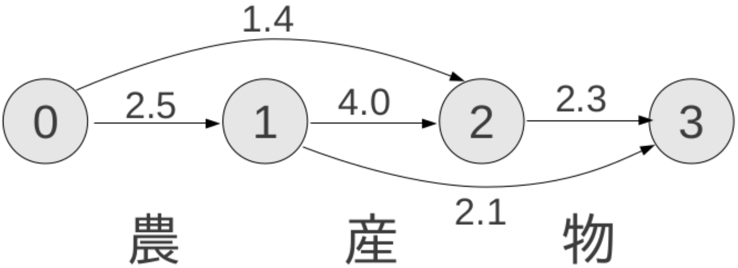



wecanonlyseeashortdistanceahead
we canon l y see ash ort distance ahead

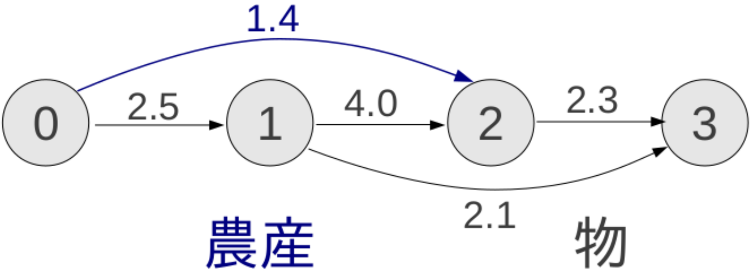
– Alan Turing

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

A graph-based approach

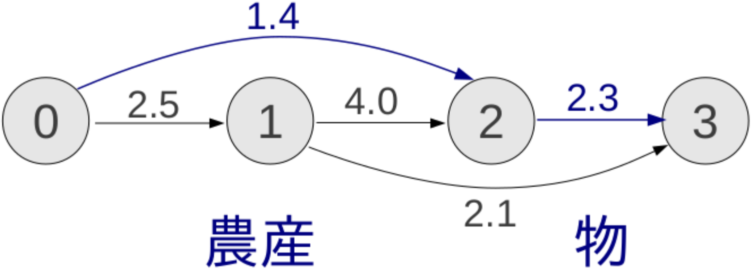


A graph-based approach



- Each edge is a word (segmentation decision)
- Each edge weight is a unigram probability

A graph-based approach



- Each path is a segmentation
- The path weight is the unigram probability of the sentence

Language model

$P($ 農産 物 価格 安定 法 $) = 4.12 * 10^{-23}$

$P($ 農産 物価 格安 定法 $) = 3.53 * 10^{-24}$

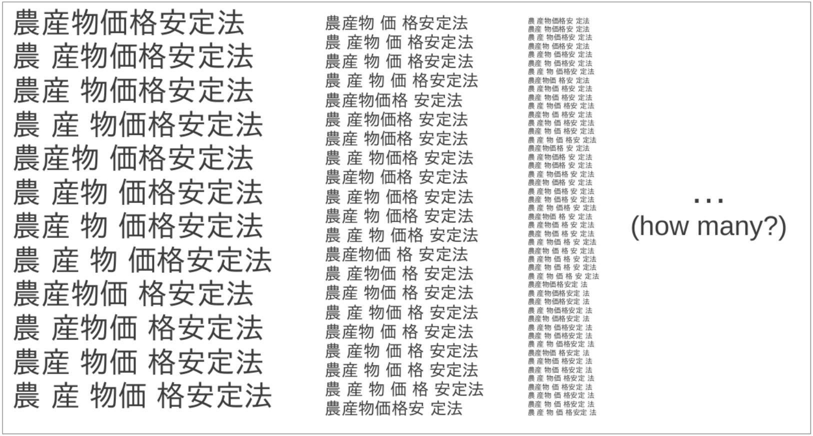
$P($ 農産 物 価 格安 定法 $) = 6.53 * 10^{-25}$

$P($ 農産 物 価格 安 定法 $) = 6.53 * 10^{-27}$

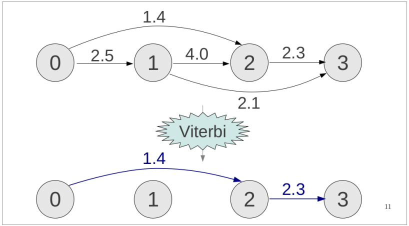
...

- Language model to choose sequence with highest probability
- Requires an existing tokenised corpus
- Maximum likelihood estimation (MLE):
    - Probability of a word: frequency / number of tokens in corpus

Combinatorial explosion



Finding the best segmentation



Viterbi algorithm:

- Forward step:
  - Search the graph left-to-right, picking the highest prob arc at each step,
  - leaving a pointer to the previously visited node
- Backward step:
  - Follow the pointers back to read off the path

[Algorithm covered in Lecture 3 *Morphological disambiguation*.]

Unknown words

What to do with out-of-vocabulary (OOV) items?

Smoothing:

- Add-one smoothing[3], basically just add 1 to every count
  - Assigns too much probability mass to unknown words
- Use the Good-Turing method (see Manning and Schütze, p.212)

[3] If you do this, avoid a man called Ken Church.

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.

Formats

Extra-spaces format:

```
Крайняя западная точка расположена в районе Фленсбурга (9°10′ в.д.).
```

↓

```
Крайняя западная точка расположена в районе Фленсбурга ( 9°10′ в.д. ) .
```

- Widely used
- Basically just add in (extra) spaces to indicate tokenisation
- Easy to process
- Loses original text

Formats

CoNLL-U:

```
# text = Крайняя западная точка расположена в районе Фленсбурга (9°10′ в.д.).
1    Крайняя       _  _  _  _  _  _  _  _
2    западная      _  _  _  _  _  _  _  _
3    точка         _  _  _  _  _  _  _  _
4    расположена   _  _  _  _  _  _  _  _
5    в             _  _  _  _  _  _  _  _
6    районе        _  _  _  _  _  _  _  _
7    Фленсбурга    _  _  _  _  _  _  _  _
8    (             _  _  _  _  _  _  _  SpaceAfter=No
9    9°10′         _  _  _  _  _  _  _  _
10   в.д.          _  _  _  _  _  _  _  SpaceAfter=No
11   )             _  _  _  _  _  _  _  SpaceAfter=No
12   .             _  _  _  _  _  _  _  _
```

- More complex to process
- Original text can be reconstructed
- universaldependencies.org/format.html

Evaluation

Most common evaluation method is Word Error Rate (WER):

- Number of edits divided by number of tokens in the reference
  - Edits: insertions (I), deletions (D), substitutions (S)
- Implemented with Levenshtein distance

wecanonlyseeashortdistanceahead

```
REF: we can   only   see a   short distance ahead
HYP: we canon l    y see ash ort   distance ahead
EVA:    S     S    I     S   S
```

WER: $\frac{5}{8} = 62.5\%$

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 or existing code for this

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