



НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ

Tokenisation and word segmentation

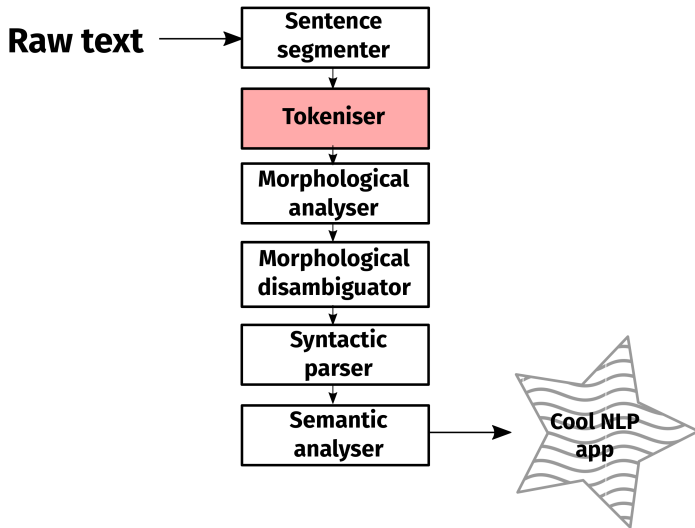
Francis M. Tyers

ftyers@hse.ru

<https://www.hse.ru/org/persons/209454856>

Национальный исследовательский университет
«Высшая школа экономики» (Москва)

13 октября 2018 г.



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

- People can generally read sentences even if there is no whitespace, but:
 - The vast majority of languages use some kind of whitespace-based word/token separator

Some questions:

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

And how about abbreviations:

- *Гипотеза была выдвинута Каролом Борсуком в 1933 г.*
 - Is there one " " here or two?
- *В 1933 г. гипотеза была выдвинута Каролом 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 ?

The ideal tokenisation may depend on the task.¹

- Russian–Arabic MT:
 - Split off clitics²
- Sentiment analysis:
 - Definitely want emoticons to be tokenised as a single unit ;)

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

```

1 import sys, re
2
3 abbr = ['etc.', 'e.g.', 'i.e.']
4
5 def tokenise(line, abbr):
6     line = re.sub(r'([\(\)\":?;!;])', r'_\g<1>_', line)
7     line = re.sub(r'([0-9])', r'_\g<1>_', line)
8     line = re.sub(r'([0-9])', r'_\g<1>_', line)
9     line = re.sub(r'_+', r'_', line[:-1])
10
11     output = []
12     for token in line.split('_'):
13         if token[-1] == '.' and token not in abbr:
14             token = token[:-1] + '_'
15         output.append(token)
16
17     return '_'.join(output)
18
19 line = sys.stdin.readline()
20 while line != '':
21     print(tokenise(line.strip('\n'), abbr))
22     line = sys.stdin.readline()

```

Split off always-separating punctuation [6]

A simple approach

```

1 import sys, re
2
3 abbr = ['etc.', 'e.g.', 'i.e.']
4
5 def tokenise(line, abbr):
6     line = re.sub(r'([\(\)\":?;!;])', r'_\g<1>_', line)
7     line = re.sub(r'([0-9])', r'_\g<1>_', line)
8     line = re.sub(r'([0-9])', r'_\g<1>_', line)
9     line = re.sub(r'_+', r'_', line[:-1])
10
11     output = []
12     for token in line.split('_'):
13         if token[-1] == '.' and token not in abbr:
14             token = token[:-1] + '_'
15         output.append(token)
16
17     return '_'.join(output)
18
19 line = sys.stdin.readline()
20 while line != '':
21     print(tokenise(line.strip('\n'), abbr))
22     line = sys.stdin.readline()

```

Split off commas not part of numeral expressions [7,8]

A simple approach

```

1 import sys, re
2
3 abbr = ['etc.', 'e.g.', 'i.e.']
4
5 def tokenise(line, abbr):
6     line = re.sub(r'([\(\)\":!;])', r'_\g<1>_', line)
7     line = re.sub(r'([0-9])', r'_\g<1>_', line)
8     line = re.sub(r'([0-9])', r'_\g<1>_', line)
9     line = re.sub(r'_+', r'_', line[:-1])
10
11     output = []
12     for token in line.split('_'):
13         if token[-1] == '.' and token not in abbr:
14             token = token[:-1] + '_'
15         output.append(token)
16
17     return '_'.join(output)
18
19 line = sys.stdin.readline()
20 while line != '':
21     print(tokenise(line.strip('\n'), abbr))
22     line = sys.stdin.readline()

```

Collapse multiple spaces [9]

A simple approach

```

1 import sys, re
2
3 abbr = ['etc.', 'e.g.', 'i.e.']
4
5 def tokenise(line, abbr):
6     line = re.sub(r'([\(\)\":?;!;])', r'_\g<1>_', line)
7     line = re.sub(r'([0-9])', r'_\g<1>_', line)
8     line = re.sub(r'([0-9])', r'_\g<1>_', line)
9     line = re.sub(r'_+', r'_', line[:-1])
10
11     output = []
12     for token in line.split('_'):
13         if token[-1] == '.' and token not in abbr:
14             token = token[:-1] + '_'
15         output.append(token)
16
17     return '_'.join(output)
18
19 line = sys.stdin.readline()
20 while line != '':
21     print(tokenise(line.strip('\n'), abbr))
22     line = sys.stdin.readline()

```

Split of full stops not part of abbreviations [12–14]

- 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 ;____;

Example:

В 1933 г. гипотеза была выдвинута Каролом Борсуком.

В	1	9	3	3	г	.	г	и	п	о	т	е	з	а	...
+	-	-	-	+	-	+	-	-	-	-	-	-	-	+	...

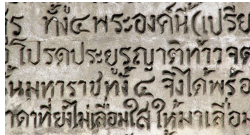
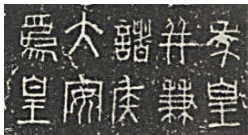
Two classes:

- (+) Token boundary follows
- (-) No boundary

Process:³

- Read through the sentence character by character
- Classify each character into 'token boundary or not?'

³Straka, Hajič and Straková (2016) "UDPipe: Trainable Pipeline for Processing CoNLL-U Files Performing Tokenization, Morphological Analysis, POS Tagging and Parsing".



- 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



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

Graphics by Graham Neubig (NAIST)

Maxmatch:

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

Graph-based:

- Statistical algorithm
 - Requires pre-segmented corpus

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

if sentence is empty

return empty list

for $i \leftarrow \text{length}(\text{sentence})$ **downto** 1

firstword = first i chars of *sentence*

remainder = rest of *sentence*

if InDictionary(*firstword*, dictionary)

return list(*firstword*, MaxMatch(*remainder*, dictionary))

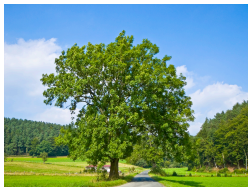
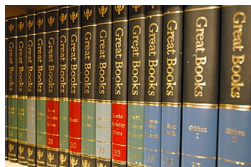
no word was found, so make a one-character word

firstword = first char of *sentence*

remainder = rest of *sentence*

return list(*firstword*, MaxMatch(*remainder*, dictionary))

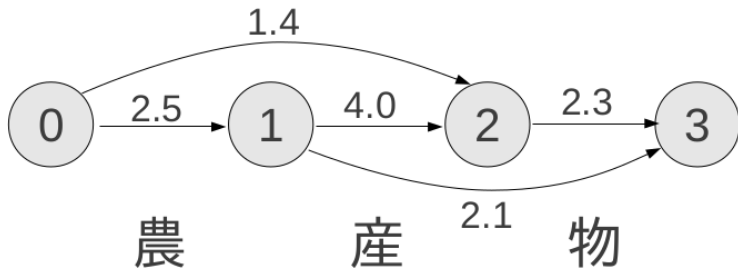
- Start at beginning of string
- Iteratively look up the longest word in the dictionary
- If no word is found, output a single character

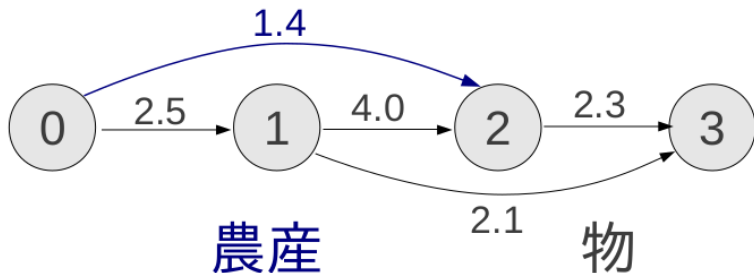


we can only see a short distance ahead
we **canon** ly see **ash ort** distance ahead

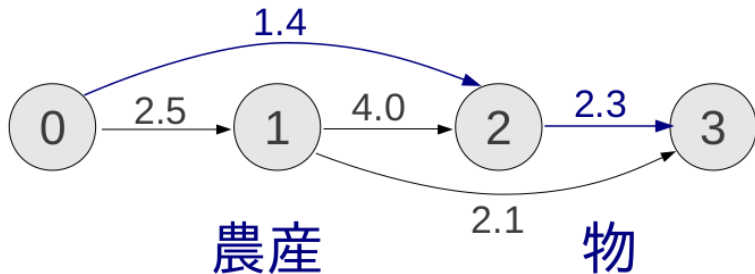
– Alan Turing

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





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



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

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

$$P(\text{農産 物 価格 安定 法}) = 3.53 \cdot 10^{-24}$$

$$P(\text{農産 物 価格 安定 法}) = 6.53 \cdot 10^{-25}$$

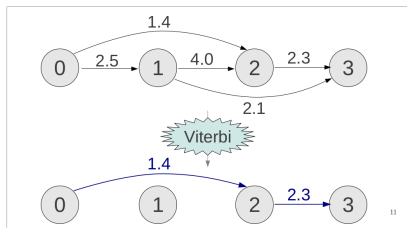
$$P(\text{農産 物 価格 安定 法}) = 6.53 \cdot 10^{-27}$$

...

- 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

[illegible]

17 / 23



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.*]

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

Smoothing:

- Add-one smoothing⁴, 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)

⁴If you do this, avoid a man called Ken Church. See: Gale and Church (1994) "What's wrong with adding one?".

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.

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

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

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:	w	e		c	a	n		o	n	l	y		s	e	e		a		s	h	o	r	t		d	i	s	t	a	n	c	e		a	h	e	a	d		
HYP:	w	e		c	a	n		l			y		s	e	e		a	s	h		o	r	t				d	i	s	t	a	n	c	e		a	h	e	a	d
EVA:				S				S					I				S																							

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

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