

# When should I submit my project?

- May/June
- September

# When should I submit my project?

- **May/June**
- September

# 92586 Computational Linguistics

## Lesson 1. Tokens

Alberto Barrón-Cedeño

Alma Mater Studiorum-Università di Bologna

a.barron@unibo.it

@albarron\_

18/02/2020



# Table of Contents

1 Words

2 Normalisation

3 Representations

# Words

# Words

What is a word?

# Words

What is a word?

Speech The smallest sequence of phonemes that can be uttered in isolation with objective or practical meaning

Text Sequences of graphemes (“letters”) [...] delimited by spaces [...] or by other graphical conventions

<https://en.wikipedia.org/wiki/Word>

# Words

What is a word?

Speech The smallest sequence of phonemes that can be uttered in isolation with objective or practical meaning

Text Sequences of graphemes (“letters”) [...] delimited by spaces [...] or by other graphical conventions

<https://en.wikipedia.org/wiki/Word>

## **Simplistic definition**

A word is a sequence of characters surrounded by spaces



# Words

What is a word?

Speech The smallest sequence of phonemes that can be uttered in isolation with objective or practical meaning

Text Sequences of graphemes (“letters”) [...] delimited by spaces [...] or by other graphical conventions

<https://en.wikipedia.org/wiki/Word>

## **Simplistic definition**

A word is a sequence of characters surrounded by spaces

Arguable, as multiple scholars claim; in particular across languages (Bender, 2013; Haspelmath, 2011)

# Words

## Lexicon

The set of all tokens (words!) in document  $d$  (or a corpus  $c$ )

# Words

## Tokenisers

We have a tokeniser, kindly provided by Church (1994)

```
tokens = re.findall('[A-Za-z]+', txt)
```

---

<sup>1</sup>Example borrowed from Lane et al. (2019, p. 34)

# Words

## Tokenisers

We have a tokeniser, kindly provided by Church (1994)

```
tokens = re.findall('[A-Za-z]+', txt)
```

Python provides a “similar” tool

```
tokens = txt.split()
```

---

<sup>1</sup>Example borrowed from Lane et al. (2019, p. 34)

# Words

## Tokenisers

We have a tokeniser, kindly provided by Church (1994)

```
tokens = re.findall('[A-Za-z]+', txt)
```

Python provides a “similar” tool

```
tokens = txt.split()
```

What if `txt` is the following?<sup>1</sup>

```
txt = """Thomas Jefferson started building Monticello at  
the age of 26."""
```

---

<sup>1</sup>Example borrowed from Lane et al. (2019, p. 34)

# Words

## Tokenisers

A better regular expression<sup>2</sup>

```
tokens = re.split(r'([-\\s.,;!?])+', txt)
```

---

<sup>2</sup>Borrowed from Lane et al. (2019, p. 43)

# Words

## Tokenisers

A better regular expression<sup>2</sup>

```
tokens = re.split(r'([-\\s.,;!?])+', txt)
```

Now let us force the regexp to match entire expressions, rather than just splitting

```
tokens = re.split(r'([-\\s.,;!?])+', txt)
```

---

<sup>2</sup>Borrowed from Lane et al. (2019, p. 43)

# Words

## Tokenisers

A better regular expression<sup>2</sup>

```
tokens = re.split(r'([-\\s.,;!?])+', txt)
```

Now let us force the regexp to match entire expressions, rather than just splitting

```
tokens = re.split(r'([-\\s.,;!?])+', txt)
```

What if we have the following text?

```
txt = "Monticello wasn't designated as UNESCO World Heritage  
Site until 1987"
```

---

<sup>2</sup>Borrowed from Lane et al. (2019, p. 43)



# Words

NLTK

<http://www.nltk.org/>

# Words

## NLTK

- Leading platform for building Python programs to work with human language data.

<http://www.nltk.org/>

# Words

## NLTK

- Leading platform for building Python programs to work with human language data.
- Easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet

<http://www.nltk.org/>

# Words

## NLTK

- Leading platform for building Python programs to work with human language data.
- Easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet
- Suite of text processing libraries for classification, **tokenization**, stemming, tagging, parsing [...]

<http://www.nltk.org/>

# Words

## Installing NLTK

```
$ pip install --user -U nltk  
$ pip install --user -U numpy  
$ python  
>>> import nltk
```

<http://www.nltk.org/>

# Words

## Installing NLTK

```
$ pip install --user -U nltk
$ pip install --user -U numpy
$ python
>>> import nltk
```

## Using (one of) the NLTK tokenisers

```
from nltk.tokenize import TreebankWordTokenizer
tokenizer = TreebankWordTokenizer()
sentence = "Monticello wasn't designated as UNESCO World
Heritage Site until 1987"
tokenizer.tokenize(sentence)
```

<http://www.nltk.org/>

# Normalisation

# Normalisation

## Case folding

Ignoring differences in the spelling of a word which involves only capitalisation (Lane et al., 2019, p. 54)



# Normalisation

## Case folding

Ignoring differences in the spelling of a word which involves only capitalisation (Lane et al., 2019, p. 54)

# We know how to deal with this, don't we?

# Normalisation

## Case folding

Ignoring differences in the spelling of a word which involves only capitalisation (Lane et al., 2019, p. 54)

# We know how to deal with this, don't we?

PROS Now TEA==tea; the vocabulary is smaller

CONS The Joker is not a character

# Normalisation

## Stemming

“Eliminate the small meaning differences of pluralisation or possessive endings of words or [...] verb form” (Lane et al., 2019, p. 57)

# Normalisation

## Stemming

“Eliminate the small meaning differences of pluralisation or possessive endings of words or [...] verb form” (Lane et al., 2019, p. 57)

```
import re
def stem(phrase):
    return ' '.join([re.findall('^(.*ss|.*?)(s)?$',
                               word)[0][0].strip("'") for word in phrase.lower()
                     .split()])

stem('houses')
stem("Doctor House's calls")
stem("stress")
```

# Normalisation

## Stemming: Porter and Snowball

Once again, people have developed (and released) more sophisticated stemming algorithms

<https://tartarus.org/martin/PorterStemmer/>

<http://snowball.tartarus.org/>

# Normalisation

## Stemming: Porter and Snowball

Once again, people have developed (and released) more sophisticated stemming algorithms

<https://tartarus.org/martin/PorterStemmer/>

<http://snowball.tartarus.org/>

```
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
' '.join([stemmer.stem(w).strip("'") for w in
"dish washer's washed dishes".split()])
```

# Normalisation

## Lemmatisation

Associating several words down to their semantic common root (adapted from (Lane et al., 2019, p. 59))

PROS Stemming might alter the meaning of a word

CONS It is more expensive; it requires a knowledge base of synonyms and endings, and POS

# Normalisation

Lemmatisation: re-use, re-use!

```
import nltk
nltk.download('wordnet')

from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

lemmatizer.lemmatize("better")
lemmatizer.lemmatize("better", pos="a")
```



What kind of NLP are we using here?

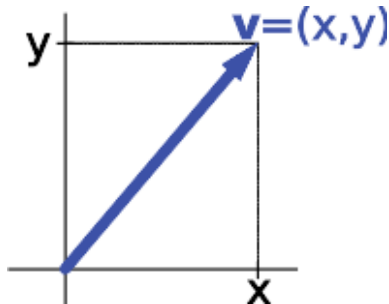
**rule-based** or **statistical**

# Representations

# Representations

## Vectors

An (Euclidean) vector is an entity endowed with a magnitude (the length of the line segment  $(A, B)$ ) and a direction (the direction from  $A$  to  $B$ ).



[https://en.wikipedia.org/wiki/Vector\\_\(mathematics\\_and\\_physics\)](https://en.wikipedia.org/wiki/Vector_(mathematics_and_physics))

[https://en.wikipedia.org/wiki/Vector\\_space](https://en.wikipedia.org/wiki/Vector_space)

# Representations

## Bag of Words (BoW)

Turning words into numbers<sup>3</sup>

```
sentence = """Thomas Jefferson began building Monticello at  
the age of 26."""
```

```
sentence_bow = {}  
for token in sentence.split():  
    sentence_bow[token] = 1  
sorted(sentence_bow.items())
```

---

<sup>3</sup>From (Lane et al., 2019, p. 35)

# Representations

Bag of Words (BoW)

Using **pandas** (data structures for data analysis, time series, and statistics)<sup>4</sup>

```
import pandas as pd
```

```
sentences = """Thomas Jefferson began building Monticello at  
the age of 26.\n"""
```

```
sentences += """Construction was done mostly by local masons  
and carpenters.\n"""
```

```
sentences += "He moved into the South Pavilion in 1770.\n"
```

```
sentences += """Turning Monticello into a neoclassical  
masterpiece was Jefferson's obsession."""
```

```
corpus = {}
```

```
for i, sent in enumerate(sentences.split('\n')):
```

```
    corpus['sent{}'.format(i)] = dict((tok, 1) for tok in  
        sent.split())
```

```
df = pd.DataFrame.from_records(corpus).fillna(0).astype(int).T
```

# Representations

## One-Hot Vectors

Turning words into numbers<sup>5</sup>

```
import numpy as np
sentence = "Thomas Jefferson began building Monticello at
the age of 26."
token_sequence = str.split(sentence)
vocab = sorted(set(token_sequence))
', '.join(vocab)

num_tokens = len(token_sequence)
vocab_size = len(vocab)
onehot_vectors = np.zeros((num_tokens, vocab_size), int)
for i, word in enumerate(token_sequence):
    onehot_vectors[i, vocab.index(word)] = 1

', '.join(vocab)
onehot_vectors
```

# Representations

## One-Hot Vectors

Turning words into numbers<sup>6</sup>

```
import pandas as pd  
pd.DataFrame(onehot_vectors, columns=vocab)
```

---

<sup>6</sup>From (Lane et al., 2019, p. 35)

# References

Bender, E. M.

2013. *Linguistic Fundamentals for Natural Language Processing: 100 Essentials from Morphology and Syntax*. Morgan & Claypool Publishers.

Church, K.

1994. UNIX for poets.

Haspelmath, M.

2011. The indeterminacy of word segmentation and the nature of morphology and syntax. *Folia Linguistica*, 45.

Lane, H., C. Howard, and H. Hapkem

2019. *Natural Language Processing in Action*. Shelter Island, NY: Manning Publication Co.