Applied Natural Language Processing

Dr Jeff Mitchell, University of Sussex

Autumn 2025

Text Documents and Pre-processing

Week 2

This time

1. Introduction to the Document Retrieval scenario
2. Segmentation and tokenization
3. How many words are there?
4. Normalisation
5. Stemming and Lemmatisation

Introduction to Document Retrieval scenario

Part 1

Document Retrieval Scenario

Given a large digital collection of documents (generally referred to as a **corpus**), how do we automate finding all of the documents which contain a given word, e.g., *cat*?

X = "cat"

[D for D in docs if X in D]

Problem

- Given a corpus, humans automatically intelligently break it down (or segment it) into meaningful units:-
  - Documents
  - Paragraphs
  - Sentences
  - Words
  - Morphemes and/or syllables
  - Characters

- Given a corpus, computers automatically see:-
  - A sequence of characters
  - Represented as a sequence of zeros and ones

What exactly do we mean by word?

- Which of these sentences contain the word ‘cat’?
  - The cat sat on the mat.
  - Lots of cats sit on the mat.
  - Cats chase mice.
  - Which category of room would you like?
  - The cat, which is really fat, sat on the mat.
  - The seeds were scattered in the field
  - Mr Cat is a farmer.
  - Shall we give a home to a rescue cat?

Using a naïve string matching algorithm, which of the sentences above might give unexpected results?

Document Representation

How do we represent collections of documents so that we can search them efficiently? This is really important if your collection of documents is the www….

For document retrieval, preprocessing of documents may include:-

- Document and sentence segmentation
- Word tokenization
- Text normalization (case, numbers, stopwords and punctuation)
- Stemming / lemmatisation (cats == cat)
- Generation of "bag-of-words" representation, ignoring frequency and order
- Indexing (word → [document ids])

Segmentation and tokenization

Part 2

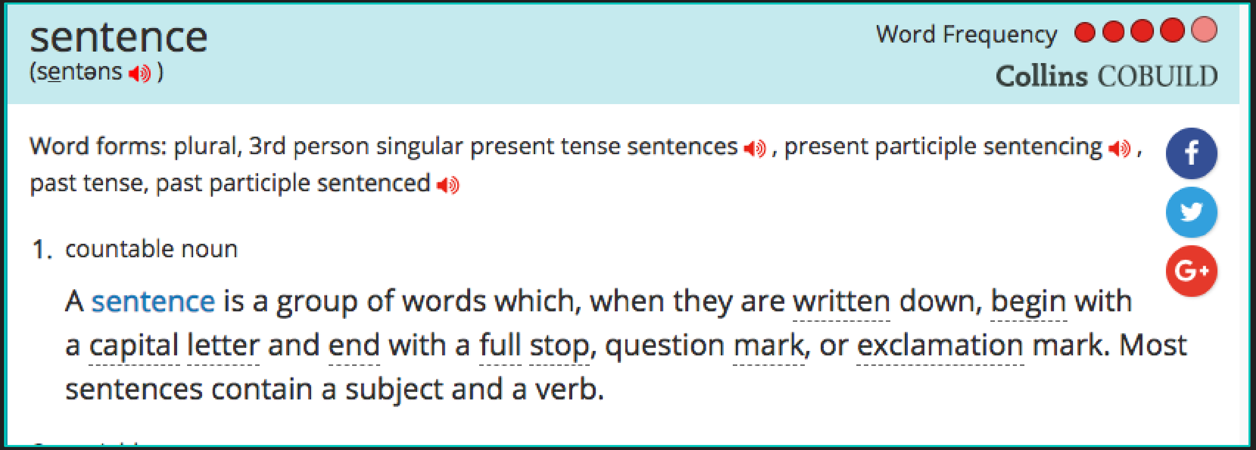
Document Segmentation

- A **corpus** is a collection of documents, which might be:
  - News articles
  - Essays
  - Books
  - Web pages
  - Tweets
  - …?

- Documents vary greatly in length so often useful to break them down further.

- A **corpus** might be stored as:
  - One document per file
  - In a single file using special characters as delimiters
  - In a single file one document per line, possibly with other meta-information about the document in csv format
- Important to know the format but usually straightforward to identify individual documents.

Sentence Segmentation 1



sentence
(sentəns)

Word Frequency

Collins COBUILD

Word forms: plural, 3rd person singular present tense sentences, present participle sentencing, past tense, past participle sentenced

1. countable noun

A sentence is a group of words which, when they are written down, begin with a capital letter and end with a full stop, question mark, or exclamation mark. Most sentences contain a subject and a verb.

At first sight, a simple problem which we might solve by:

- *writing a function which finds occurrences of full stops, question marks and exclamation marks and splits text accordingly.*

1. Do you want tea or coffee this morning?
2. Her son, who is called Mark, was taken to hospital.
3. Come here at once!
4. I went to Portsmouth last week.
5. Don't you remember?

Sentence Segmentation 2

Why doesn't this work very well?

- Ambiguous use of full stops (e.g., abbreviations)
- Ambiguous use of capitalization (e.g., proper nouns)
- Speech marks
- Incorrect use of punctuation or capitalization by writer

- I went to Portsmouth last week. Don't you remember?
- I want to go to the U.S.A. next Summer. Don't you?
- I want to go to the U.S.A. Don't you?
- I saw H.M.S. Victory in Portsmouth. It was impressive.
- "When shall we leave?" she asked.

Sentence Segmentation 3

State-of-the-art sentence tokenization methods work

- By building a binary classifier
- To decide whether a full stop is part of the word or is a sentence-boundary marker
- May be based on a sequence of rules or machine learning (trained on examples)
- Helps to have a dictionary of commonly used abbreviations

- I went to Portsmouth last week. Don't you remember?
- I want to go to the U.S.A. next Summer. Don't you?
- I want to go to the U.S.A. Don't you?
- I saw H.M.S. Victory in Portsmouth. It was impressive.
- "When shall we leave?" she asked.

Sentence Segmentation in Python

```
from nltk.tokenize import sent_tokenize
sent_tokenize(testtext)
```

```
['I went to Portsmouth last week.',
 'Don't you remember?',
 'I want to go to the U.S.A. next Summer.',
 'Don't you?',
 'I want to go to the U.S.A. Don't you?',
 'I saw H.M.S.',
 'Victory in Portsmouth.',
 'It was impressive.',
 '"When shall we leave?"',
 'she asked.']
```

https://www.nltk.org/_modules/nltk/tokenize.html

- Default sentence splitter in python's Natural Language Toolkit (NLTK) library is the Punkt sentence segmenter (Kiss and Strunk, 2006)

T. Kiss and J. Strunk (2006) Unsupervised Multilingual Sentence Boundary Detection, Computational Linguistics, 32(4). https://aclanthology.org/J06-4003/

Tokenization

- Task of segmenting running text into “words”
- Tokens might actually be:
  - Words
  - Items of punctuation
  - Numerical quantities
- Intuitively, tokens are meaningful sequences of characters delimited by whitespace
- But tokens can contain whitespace e.g.,
  - “£2, 314.99”
- Tokens might not be separated be whitespace e.g.,
  - “Ann, Bob and Chris went home.”
  - “I’m going home too.”

Question for you

- Which useful built-in Python function will take a string and return a list of the parts of that string as delimited by white space? For example,
  - input = “I went to Portsmouth.”
  - output = [“I”,”went”,”to”,”Portsmouth.”]

Tokenization in Python

3 main options

1. Use the split() function in very simple cases
2. Write and use regular expressions.
3. Use the NLTK regular expression tokenizer

```
#import the word_tokenize function from nltk
from nltk.tokenize import word_tokenize

# run the nltk tokeniser on a test sentence
test_sentence="The cat sat on the mat."
word_tokenize(test_sentence)
```

How many words are there?

Part 3

Question for you

- How many words are there in the first line of the Star Wars movies?

"A long time ago in a galaxy far, far away ...."

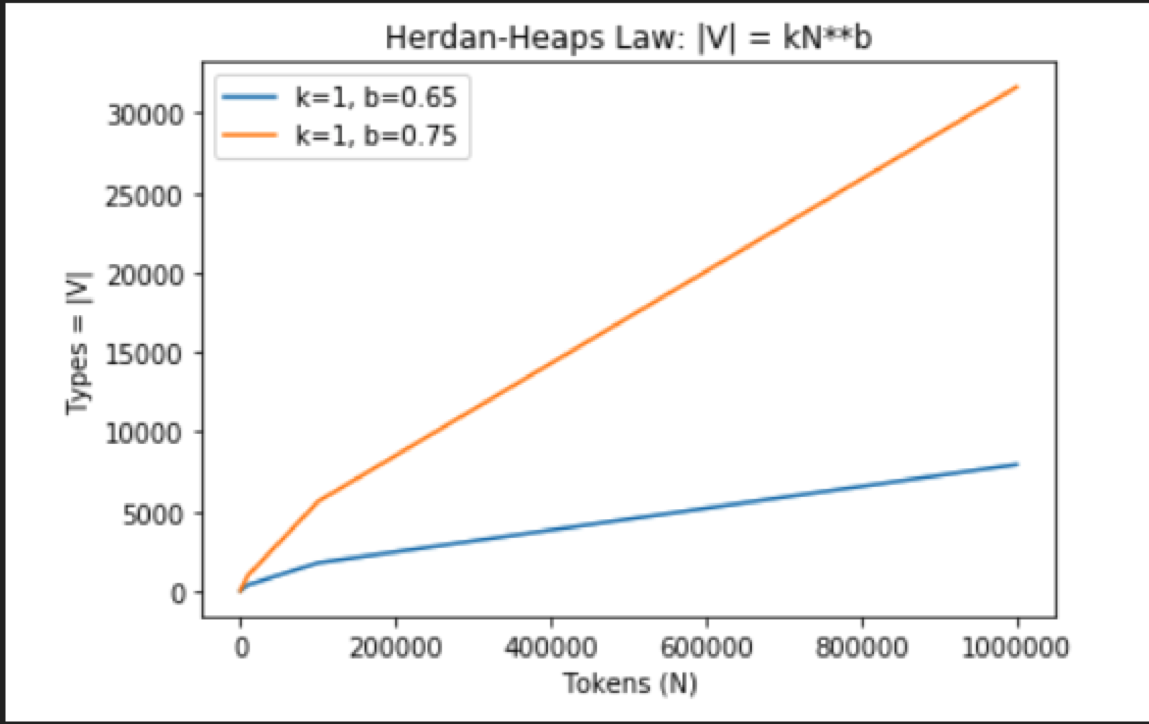
How many words in the English language?

- **Types** are the number of distinct words in a corpus; if the set of words in the vocabulary is V, the number of types is $|V|$
- **Tokens** are the total number N of running words.
- Ignoring punctuation and not carrying out case-normalisation, the Star Wars quote has 9 types and 10 tokens

Corpus	Tokens = N	Types = |V|
Shakespeare	884 thousand	31 thousand
Brown corpus	1 million	38 thousand
Switchboard telephone conversations	2.4 million	20 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13 million

Herdan-Heaps Law (Herdan, 1960; Heaps 1978)

- The larger the corpora, the more word types we find.



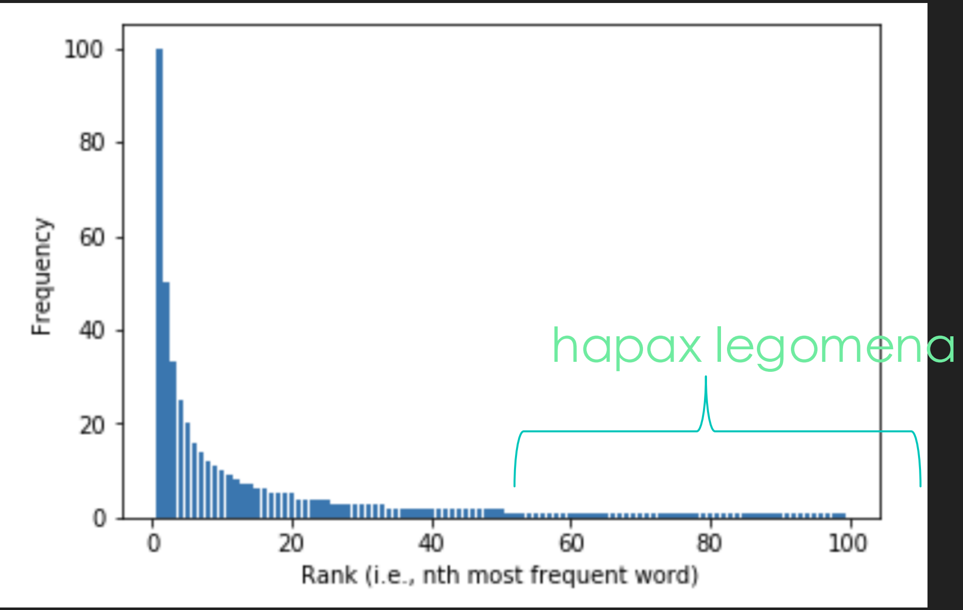
If k and β are positive constants with 0< β<1:

$$|V| = k\ N^{\beta}$$

- The values of k and β depend on the domain and genre.
- k ≈ 1, β ≈ 0.7 is typical for large general purpose corpora
- So, vocabulary size for text goes up faster than the square root of its length in words
- β will be lower for restricted domains such as the switchboard telephone conversations.

Zipf's Law

- According to Herdan-Heaps Law, average type frequency in 1M word corpus is 30.
- But frequency distribution of word types is not uniform, or even normal. Its **Zipfian**!
- This means half of the types will be ***hapax legomena***, meaning they occur only once!



- Zipf's Law states that "**the product of a word's frequency and its rank frequency is approximately constant.**"
  - So, if the most frequent word occurs 100 times,
  - the 2^{nd} most frequent word will occur 50 times,
  - the 3^{rd} most frequent word will occur 33 times,
  - And so on.

Normalisation

Part 4

Case normalisation

- In some applications (speech recognition) case isn't available.
- In others (document retrieval), lower and upper case distinctions are often considered irrelevant
- Advantages of ignoring case:
  - reduces the number of types to consider
  - increases the frequency of each type
- Disadvantages of ignoring case:
  - Increases problem of ambiguity e.g., where case is used to indicate a name

```
sent="A long time ago in a galaxy far far away"
tokens=sent.lower().split()
print(tokens)
```

```
['a', 'long', 'time', 'ago', 'in', 'a', 'galaxy', 'far', 'far', 'away']
```

Number normalisation

- How many numbers are there?
- Numbers can account for a lot of the rare words in a corpus
- In many applications, the exact number is irrelevant, so commonplace to replace numbers with a generic string like NUM
- See the lab

```
tokens="In 2018 , there are 157 undergraduates taking NLE .".split()
normalised =["NUM" if token.isdigit() else token for token in tokens]
print(normalised)
```

```
['In', 'NUM', ',', 'there', 'are', 'NUM', 'undergraduates', 'taking', 'NLE', '.']
```

Other rare words

- Might be:
  - Names
  - Scientific or other terminology
  - Spelling or other errors
- Commonplace to ignore rare words. Many applications
  - Use a lower frequency threshold
  - Or consider top n most frequent words

Punctuation and Stopword removal

- The most frequent words in English tend to be non-content bearing **function words**
- Also known as **stopwords**
- In certain applications (e.g., document retrieval), stopwords typically removed along with punctuation
  - Using a stopword list
  - Or an upper frequency threshold
- Drastically compresses the document representation
- Without affecting word-based search capability

Word	Frequency in BNC[1]
the	6 187 267
of	2 941 444
and	2 682 863
a	2 126 369
in	1 812 609
to	1 620 850
it	1 089 186
is	998 398
was	923 948
to	917 579

1. BNC = British National Corpus (100M tokens) https://www.kilgarriff.co.uk/BNClists/all.num.o5

Stemming and lemmatisation

Part 5

Morphology

When I search for articles about "*cats*", I probably want documents containing the word forms **"*cat*"** and **"*cats*"**

Morphology is the study of the way words are made up of smaller parts

Morphology Examples

- cats = cat + s
- kings = king + s
- started = start + ed
- starting = start + ing
- unnecessary = un + necessary
- antidistestablishmentarianism = anti+dis+establish+ment+arian+ism

Root or **stem**

affixes

Morphemes are the smallest meaningful unit of a word

L2.2

Inflectional morphology

Category	Distinction	Example
number	Singular vs plural	*cat* vs *cats*
gender	Masculine vs feminine	*bleu* vs *bleue* (French)
tense	Present vs past vs future	*love* vs *loved*
person	1^{st} vs 2nd vs 3rd	*I am* vs *you are* vs *he is*
case	Nominative vs accusative vs dative vs genitive	*Mark* vs *Mark's*

- Many inflectional variations follow regular rules but there are always exceptions
- What is the plural of:
  - man?
  - mouse?
  - child?

Derivational morphology

- Process of creating (or deriving) a new word from an existing word using affixes
  - Adds substantial new meaning
  - Often of a different part of speech (e.g. creating a noun from a verb)
- For example:
  - **start** (VERB) = cause to happen or begin
  - **starter** (NOUN) = thing that starts something
  - **restart** (VERB) = start again
  - **restartable** (ADJECTIVE) = capable of being restarted
- Less clear that we want to ignore distinctions (even in search applications)
- But analysing derivational morphology can help us to understand previously unseen words

STEMMING

- Removing unwanted affixes
- E.g., NLTK's Porter Stemmer
- Does not require a lexicon / dictionary
- Uses rewrite rules:
  - ATIONAL -> AT
  - FUL -> ε
  - SSES -> SS
- Errors of commission (false positives)
  - organization -> organ
- Errors of omission (false negatives)
  - urgency, European, sang, sung, mice, men, children ….

```python
from nltk.stem.porter import PorterStemmer
st = PorterStemmer()
words = ["relational","relate","hopeful",
         "caresses","organization","organ",
         "children","child"]
stems=[st.stem(word) for word in words]
for w,s in zip(words,stems):
    print("{} : {}".format(w,s))
```

```
relational : relat
relate : relat
hopeful : hope
caresses : caress
organization : organ
organ : organ
children : children
child : child
```

Lemmatisation

- Replace word with dictionary head word
- E.g., NLTK's WordNetLemmatizer
- Uses
  - A lexicon / dictionary (WordNet)
  - Knowledge of **inflectional** morphology
  - Exception list for irregulars
- Works by
  - Applying stemming rules
  - Comparing results to WordNet dictionary
  - If result is a real word, keep it, else use original word

```python
from nltk.stem.wordnet import WordNetLemmatizer
st = WordNetLemmatizer()
words = ["relational","relate","hopeful",
         "caresses","organization","organ",
         "children","child","mice","men","sang"]
lemmas=[st.lemmatize(word) for word in words]
for w,s in zip(words,lemmas):
    print("{} : {}".format(w,s))
```

```
relational : relational
relate : relate
hopeful : hopeful
caresses : caress
organization : organization
organ : organ
children : child
child : child
mice : mouse
men : men
sang : sang
```

More Python

Part 6

List Comprehensions

```
mysent = sents[1]
print(mysent)
lowered_sent=[token.lower() for token in mysent]
print(lowered_sent)
```

```
['They', 'told', 'Reuter', 'correspondents', 'in', 'Asian',
['they', 'told', 'reuter', 'correspondents', 'in', 'asian',
```

```
print(mysent)
lowered_sent=[]
for token in mysent:
  lowered_sent.append(token.lower())
print(lowered_sent)
```

```
['They', 'told', 'Reuter', 'correspondents', 'in', 'Asian',
['they', 'told', 'reuter', 'correspondents', 'in', 'asian',
```

Nested List Comprehensions

```
mysents=sents[0:10]
lowered_sents=[]
for sent in mysents:
  lowered_sent=[]
  for token in sent:
    lowered_sent.append(token.lower())
  lowered_sents.append(lowered_sent)

print(lowered_sents)
```

```
[['asian', 'exporters', 'fear'
```

```
mysents=sents[0:10]
lowered_sents=[[token.lower() for token in sent] for sent in myse
print(lowered_sents)
```

```
[['asian', 'exporters', 'fear', 'damage', 'from', 'u', '.', 's',
```

Less code ... but is it any faster?

```
[91] mysents=sents[0:1000]
```

```
%%time
lowered_sents=[]
for sent in mysents:
  lowered_sent=[]
  for token in sent:
    lowered_sent.append(token.lower())
  lowered_sents.append(lowered_sent)
```

```
CPU times: user 98.6 ms, sys: 3.86 ms, total: 102 ms
Wall time: 104 ms
```

```
[93] %%time
lowered_sents=[[token.lower() for token in sent] for sent in mysents]
```

```
CPU times: user 103 ms, sys: 4.81 ms, total: 108 ms
Wall time: 111 ms
```

Counting using Dictionaries

```
[100] def vocabulary(sentences):
          tok_counts = {}
          for sentence in sentences:
              for token in sentence:
                  tok_counts[token]=tok_counts.get(token,0)+1
          return tok_counts

      sample_vocab=vocabulary(sample)
```

```
print(sample_vocab['and'])
```

```
8
```

```
[104] print(len(sample_vocab.keys()))
```

```
225
```

Random Sampling

```
import random

pop_size=len(sents)
print("Number of sentence is {}".format(pop_size))

sample_size=10
ids=random.sample(range(pop_size),sample_size)
print(ids)
```

```
Number of sentence is 54716
[44100, 41655, 4638, 19745, 27116, 20267, 33963, 32512, 3447, 48698]
```

This sample will be different every time you run the code unless you set the random seed with
>> random.seed(someinteger)

```
sample=[sents[id] for id in ids]
print(sample)
```

```
[['The', 'sunflower', 'harvest', 'advanced', 'to', 'between', '20', 'and', '23', 'pct', 'of', 'to
```

Shell Commands and Getting Reuters to Work on Colab!

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

```
[nltk_data] Downloading package reuters to /root/nltk_data...
True
```

```
[47] !ls '/root/nltk_data/corpora/'
```

```
reuters.zip
```

```
!unzip '/root/nltk_data/corpora/reuters.zip' -d '/root/nltk_data/corpora'
```

```
[49] !ls '/root/nltk_data/corpora'
```

```
reuters   reuters.zip
```

```
[35] from nltk.corpus import reuters

     sents = reuters.sents()
     sample_size = 10

     for s in sents[1:10]:
         print(len(s))
```

The first time you use an nltk resource, either on Colab or on Anaconda, you need to download it. But for some reason, 'reuters' doesn't auto-extracton Colab

Shell commands can be used in notebooks if they are prefixed by !

- **!ls** to list a directory
- **!unzip** to unzip a file
- **-d** option to give the directory location for unzipped files

Making progress

- This week you should complete **all** of the exercises in **all 2 notebooks** for week 2 on Text Documents and Preprocessing:

  - [ ] Part 1: NLE2023_Lab_2_1.ipynb
  - [ ] Part 2: NLE2023_Lab_2_2.ipynb

Keywords Check

corpus		lemma	
document retrieval		lemmatisation	
pre-process		inflection	
segmentation		derivation	
tokenisation		case	
normalisation		stopword	
canonicalisation	= normalisation	content word	
morpheme		function word	
morphology		word token	
stemming		word type	