Language Technology

Chapter 9: Counting Words

Pierre Nugues

Pierre.Nugues@cs.lth.se

September 5 and 9, 2024

Structure of this Lecture

The lecture will consist of three parts:

1. Word segmentation and word count
2. Word and document representation
3. Document classification

Text Segmentation

Latin inscriptions on the *lapis niger*. *Corpus inscriptionum latinarum*. CIL I, 1. Picture from Wikipedia

Getting the Words from a Text: Tokenization

Arrange a list of characters:

```
[l, i, s, t, ' ', o, f, ' ', c, h, a, r, a, c, t, e, r, s]
```

into words:

```
[list, of, characters]
```

Sometimes tricky:

- Dates: 28/02/96
- Numbers: 9,812.345 (English), 9 812,345 (French and German) 9.812,345 (Old fashioned French)
- Abbreviations: km/h, m.p.h.,
- Acronyms: S.N.C.F.

Tokenizers use rules (or regexes) or statistical methods.

Tokenizing in Python: Using the Content

Simple program: The words

```
import regex as re

re.findall(r'\p{L}+', text)
```

The words and the rest

```
re.findall(r'\p{L}+|[^\s\p{L}]+', text)
```

Numbers and punctuation

```
re.findall(r'\p{L}+|\p{N}+|\p{P}|[^\s\p{L}\p{N}\p{P}]+', text)
```

Tokenizing in Python: Using the Boundaries

Simple program

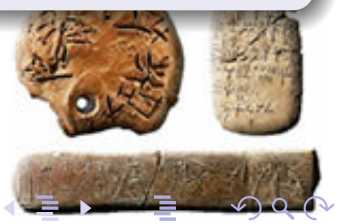
```
pattern1 = r'\s+'

re.split(pattern1, text)
```

Punctuation

```
pattern2 = r'([\p{S}\p{P}]+)'

re.split(
    pattern1,
    re.sub(pattern2, r' \1 ', text))
```



Improving Tokenization

The tokenization algorithm is word-based and defines a content
It does not work on nomenclatures such as Item #N23-SW32A, dates, or numbers
Instead it is possible to improve it using a boundary-based strategy with spaces (using for instance `\s`) and punctuation
But punctuation signs like commas, dots, or dashes can also be parts of tokens
Possible improvements using micrograms
At some point, need of a dictionary:
Can't → can n't, *we'll* → we 'll
J'aime → j' aime but *aujourd'hui*
See: `https://github.com/karpathy/minGPT/blob/master/mingpt/bpe.py`

Sentence Segmentation

As for tokenization, segmenters use either rules (or regexes) or statistical methods.
Grefenstette and Tapanainen (1994) used the Brown corpus and experimented increasingly complex rules
Most simple rule: a period corresponds to a sentence boundary: 93.20% correctly segmented
Recognizing numbers:

`[0-9]+(\/[0-9]+)+`	Fractions, dates
`([+\-])?[0-9]+(\.)?[0-9]*%`	Percent
`([0-9]+,?)+(\.[0-9]+|[0-9]+)*`	Decimal numbers

93.78% correctly segmented

Abbreviations

Common patterns (Grefenstette and Tapanainen 1994):

- single capitals: *A.*, *B.*, *C.*,
- letters and periods: *U.S. i.e. m.p.h.*,
- capital letter followed by a sequence of consonants: *Mr. St. Assn.*

Regex	**Correct**	**Errors**	**Full stop**
`[A-Za-z]\.`	1,327	52	14
`[A-Za-z]\.([A-Za-z0-9]\.)+`	570	0	66
`[A-Z][bcdfghj-np-tvxz]+\.`	1,938	44	26
Totals	3,835	96	106

Correct segmentation increases to 97.66%
With an abbreviation dictionary to 99.07%

Counting Words With Unix Tools

1. `tr -cs 'A-Za-z' '\n' <input_file |`
   Tokenize the text in `input_file`, where `tr` behaves like Perl `tr`: We have one word per line and the output is passed to the next command.
2. `tr 'A-Z' 'a-z' |`
   Translate the uppercase characters into lowercase letters and pass the output to the next command.
3. `sort |`
   Sort the words. The identical words will be grouped in adjacent lines.
4. `uniq -c |`
   Remove repeated lines. The identical adjacent lines will be replaced with one single line. Each unique line in the output will be preceded by the count of its duplicates in the input file (`-c`).
5. `sort -rn |`
   Sort in the reverse (`-r`) numeric (`-n`) order: Most frequent words first.
6. `head -5`
   Print the five first lines of the file (the five most frequent words).

Counting Words in Python

```
def tokenize(text):
    words = re.findall(r'\p{L}+', text)
    return words

def count_unigrams(words):
    frequency = {}
    for word in words:
        if word in frequency:
            frequency[word] += 1
        else:
            frequency[word] = 1
    return frequency
```

Counting Words in Python (Cont'd)

```
if __name__ == '__main__':
    text = sys.stdin.read().lower()
    words = tokenize(text)
    frequency = count_unigrams(words)
    for word in sorted(frequency.keys()):
        print(word, '\t', frequency[word])
```

The Counter Class

Python has a build-in class to count the items in a list

```
from collections import Counter


words = tokenize(text).lower()
counter = Counter(words)
```

Counter is a dictionary:

```
>>> counter['hector']
480
```

and has a method to sort the frequencies

```
>>> counter.most_common(10)
[('the', 9948),
 ('and', 6624),
 ('of', 5606),
 ('to', 3329), ...
```

Demo

Tokenizing and counting:
https://github.com/pnugues/pnlp/tree/main/notebooks

Posting Lists

Many websites, such as Wikipedia, index text using an inverted index. Each word in the dictionary is linked to a posting list that gives all the documents where this word occurs and its positions in a document.

Collection

D1: Chrysler plans new investments in Latin America.

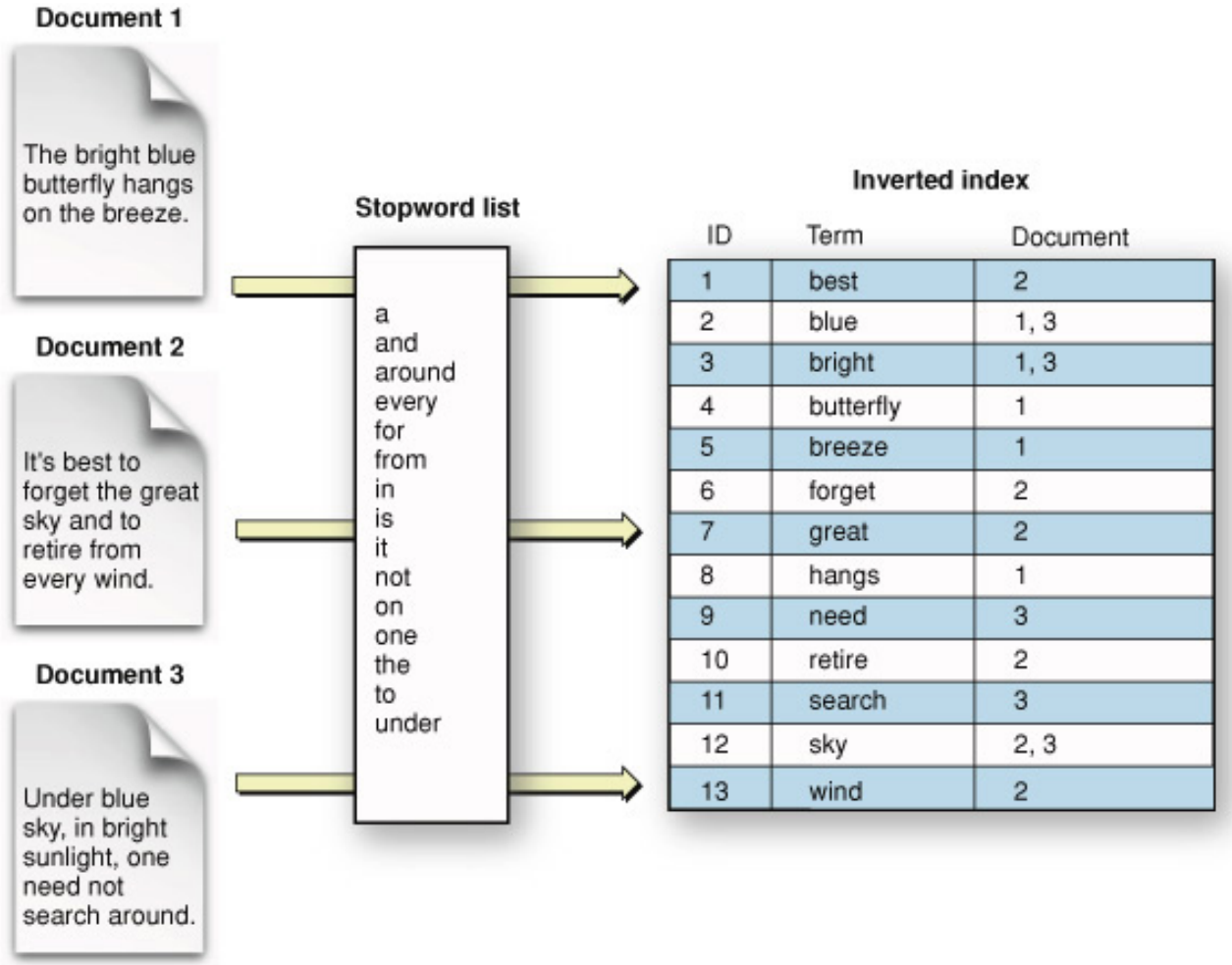
D2: Chrysler plans major investments in Mexico.

Index

Words	Posting lists
America	(D1, 7)
Chrysler	(D1, 1) → (D2, 1)
in	(D1, 5) → (D2, 5)
investments	(D1, 4) → (D2, 4)
Latin	(D1, 6)
major	(D2, 3)
Mexico	(D2, 6)
new	(D1, 3)
plans	(D1, 2) → (D2, 2)

Lucene is a high quality open-source indexer. (`http://lucene.apache.org/`)

Inverted Index (Source Apple)



https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/SearchKitConcepts/searchKit_basics/searchKit_basics.html

Information Retrieval: The Vector Space Model

The vector space model represents a document in a space of words.

Documents \Words	w_1	w_2	w_3	...	w_m
D_1	$C(w_1, D_1)$	$C(w_2, D_1)$	$C(w_3, D_1)$	...	$C(w_m, D_1)$
D_2	$C(w_1, D_2)$	$C(w_2, D_2)$	$C(w_3, D_2)$	...	$C(w_m, D_2)$
...					
D_n	$C(w_1, D_n)$	$C(w_2, D_n)$	$C(w_3, D_n)$	...	$C(w_m, D_n)$

It was created for information retrieval to compute the similarity of two documents or to match a document and a query.
We compute the similarity of two documents through their dot product.

The Vector Space Model: Example

A collection of two documents D1 and D2:

D1: Chrysler plans new investments in Latin America.

D2: Chrysler plans major investments in Mexico.

The vectors representing the two documents:

D.	america	chrysler	in	investments	latin	major	mexico	new	plans
1	1	1	1	1	1	0	0	1	1
2	0	1	1	1	0	1	1	0	1

The vector space model represents documents as bags of words (BOW) that do not take the word order into account.

The dot product is $\mathbf{D1} \cdot \mathbf{D2} = 0+1+1+1+0+0+0+0+1 = 4$

Their cosine is $\frac{\mathbf{D1} \cdot \mathbf{D2}}{||\mathbf{D1}||.||\mathbf{D2}||} = \frac{4}{\sqrt{7}.\sqrt{6}} = 0.62$

Giving a Weight

Word clouds give visual weights to words

Image: Courtesy of Jonas Wisbrant

$TF \times IDF$

The frequency alone might be misleading
Document coordinates are in fact $tf \times idf$: Term frequency by inverted document frequency.

- Term frequency $tf_{i,j}$: frequency of term j in document i
- Inverted document frequency:

$$idf_j = \log(\frac{N}{n_j}),$$

where N is the total number of documents in the collection and n_j is the number of documents, where term j occurs.

Note that tfidf has more than one possible definitions:
`https://en.wikipedia.org/wiki/Tf%E2%80%93idf`

Document Similarity

Documents are vectors where coordinates could be the count of each word: $\mathbf{d} = (C(w_1), C(w_2), C(w_3), ..., C(w_n))$
The similarity between two documents or a query and a document, $\mathbf{q}$ and $\mathbf{d}$, is given by their cosine:

$$\cos(\mathbf{q}, \mathbf{d}) = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q_i^2} \sqrt{\sum_{i=1}^{n} d_i^2}}.$$

ranging from 0 (orthogonal) to 1 (identical or very similar).

Categorizing Text

Text categorization (or classification) is a task where we assign one or more classes to a text.
The text size can range from a few words to entire books.
In sentiment analysis (or opinion mining), the goal is to classify judgments or emotions expressed, for instance, in product reviews collected from consumer forums, into three base categories:

- positive,
- negative,
- or neutral;

In spam detection, the categorizer classifies electronic messages into two classes:

- *spam* or
- *no spam*, often called *ham*.

The Reuters Corpus

The Reuters corpus consists of 800,000 economic newswires in English and about 500,000 in 13 other languages.
Each newswire is manually annotated with one or more topics selected from a set of 103 predefined categories, such as:

C11: STRATEGY/PLANS,

C12: LEGAL/JUDICIAL,

C13: REGULATION/POLICY,

C14: SHARE LISTINGS

etc.

Automatic Categorization

- With an annotated corpus, we can apply supervised machine-learning techniques to train classifiers.
- The training procedure uses a bag-of-word representation of the documents, either with Boolean features, term frequencies, or $tf \times idf$, and their classes as input.
- This process is called a vectorization
- Logistic regression is a simple, yet efficient, technique to carry out text classification.

Modeling the Texts

- Tfidf is an efficient vectorization technique
- We store one text (one vector) per row in an X matrix.
- We store the corresponding classes in an $\mathbf{y}$ vector

$$X = \begin{bmatrix} text1 \\ text2 \\ text3 \\ \dots \\ textn \end{bmatrix}; \mathbf{y} = \begin{bmatrix} class1 \\ class1 \\ class2 \\ \dots \\ class1 \end{bmatrix};$$

We then train a model to categorize the texts:

- sklearn uses the `fit()` function for all its classifiers
- We then predict the class of a text with `predict()`

Demo

Text categorization:
https://github.com/pnugues/pnlp/tree/main/notebooks