Université Jean Monnet, Saint-Étienne
Faculté des Sciences et Techniques de Saint-Étienne
Laboratoire Hubert Curien, UMR CNRS 5516

Recherche d'Information (RI)

Mathias Géry
Mathias.Gery@univ-st-etienne.fr

Information Retrieval: Outline

0) Introduction

1) IR Model & Boolean Retrieval

2) Pre-Processing & Dictionary

3) Ranked Retrieval

4) Experimental Evaluation of IR

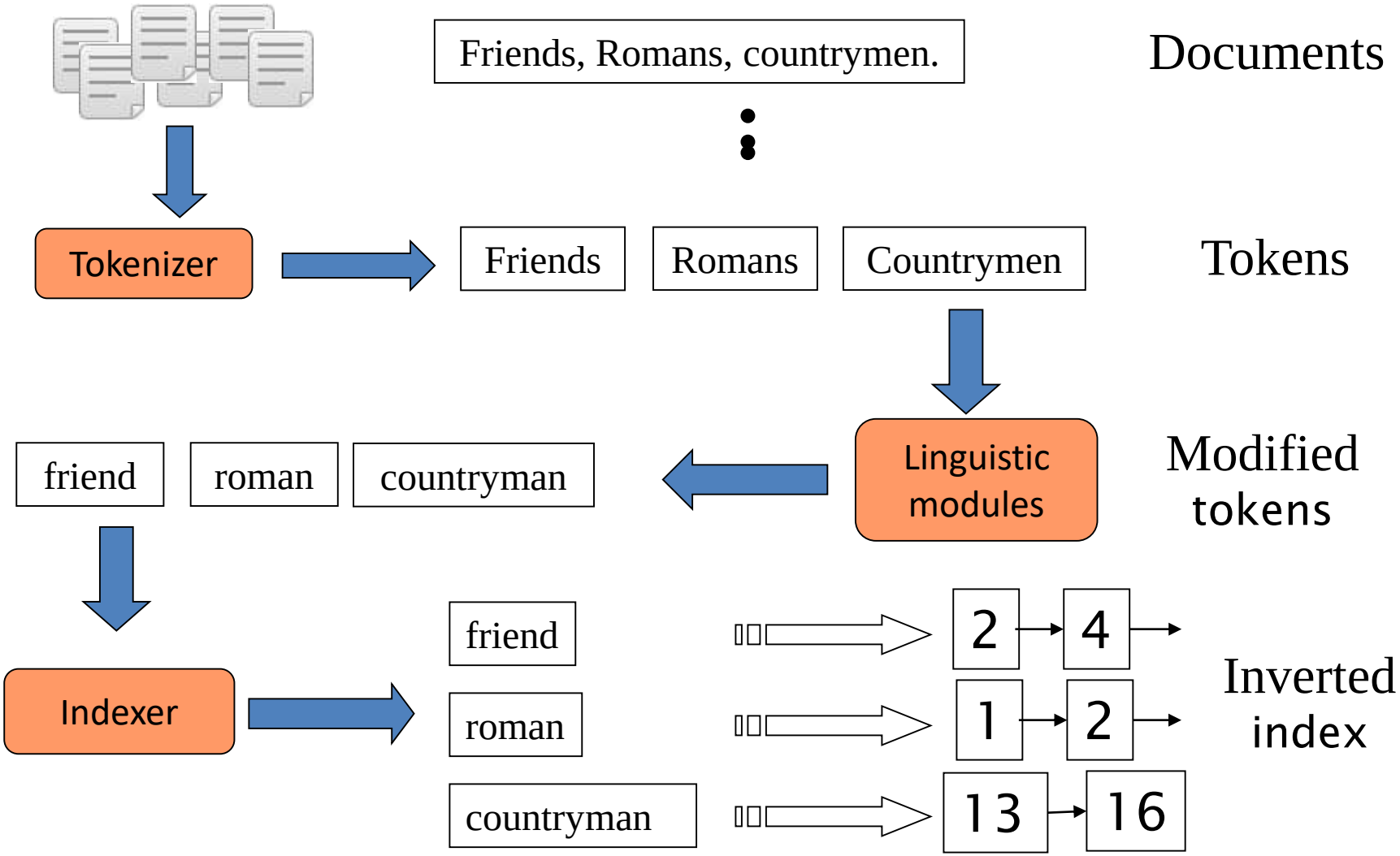
5) Structured Retrieval

6) Link Analysis for IR

Chapter 2
Pre-Processing & Dictionary

2.1: Indexing
2.2: Terms: tokenization
2.3: Stop-words
2.4: Normalization
2.5: Stemming
2.6: Statistics: Heaps' law, Zipf's law
2.7: Practice

(Recall) Inverted index construction



Choosing document unit

- What is a “document”?
  - A file?
  - An email? (Perhaps one of many in a single mbox file)
    - What about an email with 5 attachments?
  - A group of files: Powerpoint or LaTeX split over HTML pages?
  - What about **granularity**:
    - books / plays?
    - proceedings / article?
    - Passage retrieval, element retrieval?

Parsing a document

- What format is it in?
  - pdf/word/excel/html?
- What character set is in use?
  - (Latin, Unicode, UTF-8, …)
- Other decoding:
  - HTML: é
  - Latex: \'e
  - XML: &
- What language is it in?
  - Many things depends on the language…

Chapter 2
Pre-Processing & Dictionary

2.1: Indexing
2.2: Terms: tokenization
2.3: Stop-words
2.4: Normalization
2.5: Stemming
2.6: Statistics: Heaps' law, Zipf's law
2.7: Practice

Index Terms or « Features »

- The index is accessed by the atoms of a query language.
- The atoms are called “features” or “terms”.
- Most common feature types:
  - Words in text
  - N-grams (consecutive substrings) of a document
  - Manually assigned terms (controlled vocabulary)
  - Document structure (sentences & paragraphs)
  - Inter- or intra-document links (e.g., citations)

Tokenization: Token / Term?

- **Token:** an instance of a sequence of characters that are grouped together as a useful semantic unit for processing (hopefully).

- **Term:** normalized token included in the final dictionary.

Tokenization

- **Input:** "***Friends, Romans and Countrymen***"
- **Output:** Tokens
  - ***Friends***
  - ***Romans***
  - ***and***
  - ***Countrymen***
- A token is an instance of a sequence of characters.
- Each such token is now a candidate for an index entry, after further processing.

Tokenization: issues

- *Finland's capital* → *Finland* AND *s*? *Finlands*? *Finland's*?
- *Hewlett-Packard* → *Hewlett* and *Packard* as two tokens?
  - *state-of-the-art*: break up hyphenated sequence.
  - *co-education*
  - *lowercase*, *lower-case*, *lower case* ?
  - It can be effective to get the user to put in possible hyphens
- *San Francisco*: one token or two?
  - How do you decide it is one token?
  - More complex processing (e.g. Named Entities extraction…)

Numbers

- *3/20/91 Mar. 12, 1991 20/3/91*
- *55 B.C.*
- *B-52*
- *My PGP key is 324a3df234cb23e*
- *(800) 234-2333*
  - Often have embedded spaces
  - Older IR systems may not index numbers (but often very useful)
  - Will often index “meta-data” separately:
    - Creation date, format, etc.

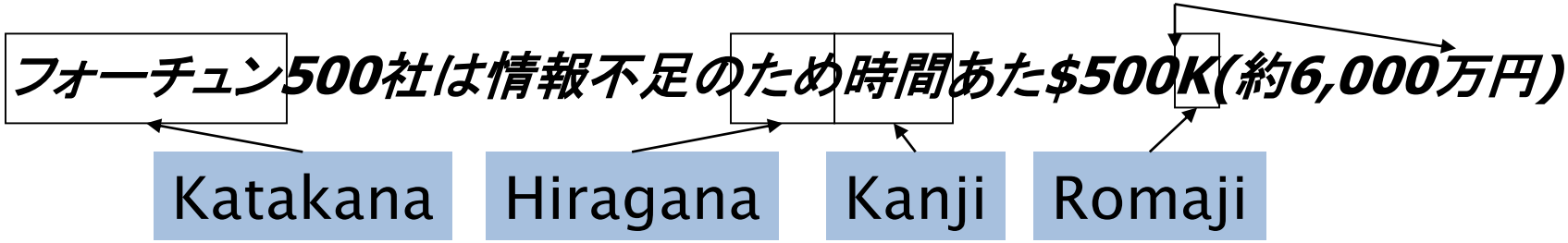
Tokenization: language issues

- French
  - ***L'école*** → one token or two?
    - ***L*** ? ***L'*** ? ***La*** ?
    - Want ***l'école*** to match with ***une école***
      - Until at least 2003, it didn't on Google
        - » Internationalization!

- German noun compounds are not segmented
  - ***Lebensversicherungsgesellschaftsangestellter***
  - 'life insurance company employee'
  - German retrieval systems benefit greatly from a **compound splitter** module (~15% performance boost).

Tokenization: language issues

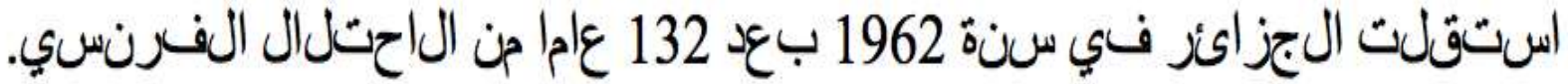
- Chinese and Japanese have no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - Not always guaranteed a unique tokenization
- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats



End-user can express query entirely in hiragana!

Tokenization: language issues

- Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right.
- Words are separated, but letter forms within a word form complex ligatures:



- ← → ← → ← start
- ‘Algeria achieved its independence in 1962 after 132 years of French occupation.’

Chapter 2
Pre-Processing & Dictionary

2.1: Indexing
2.2: Terms: tokenization
2.3: Stop-words
2.4: Normalization
2.5: Stemming
2.6: Statistics: Heaps' law, Zipf's law
2.7: Practice

Stop words removal

- Idea: exclude from the dictionary the commonest words.
- Based on a “stop list”.
- Intuition:
  - These words have little semantic content: *the*, *a*, *and*, *to*, *be*
  - There are a lot of them: ~30% of postings for top 30 words

- English stop list example:

  a an and are as at be by for from has he in is it its let not of on or that the to was were will with

- See also: a small stop list, a collection of stop lists in 50 languages.

Stop words removal

- The trend is away from using stop words:
  - Good compression techniques means the space for including stop words in a system is very small.
  - Good query optimization techniques mean you pay little at query time for including stop words.
  - You need them for:
    - Phrase queries: “King of Denmark”
    - Various song titles, etc.: “Let it be”, “To be or not to be”
    - “Relational” queries: “flights to London”

Chapter 2
Pre-Processing & Dictionary

2.1: Indexing
2.2: Terms: tokenization
2.3: Stop-words
2.4: Normalization
2.5: Stemming
2.6: Statistics: Heaps' law, Zipf's law
2.7: Practice

Normalization to terms

- We may need to “normalize” words in indexed text as well as query words into the same form.
  - We want to match ***U.S.A.*** and ***USA***

- We most commonly implicitly define equivalence classes of terms by, e.g.,
  - deleting periods to form a term:
    - ***U.S.A., USA → USA***
  - deleting hyphens to form a term:
    - ***anti-discriminatory, antidiscriminatory → antidiscriminatory***

Normalization: other languages

- Accents: e.g., French ***résumé*** vs. ***resume.***
- Umlauts: e.g., German: ***Tuebingen*** vs. ***Tübingen***
- Even in languages that standardly have accents, users often may not type them
  - Often best to normalize to a de-accented term
    - ***Tuebingen, Tübingen, Tubingen → Tubingen***

- **<u>Crucial:</u>** Need to "normalize" indexed text as well as query terms identically

Case folding

- Reduce all letters to lower case
  - exception: upper case in mid-sentence?
    - e.g., General Motors
    - Fed vs. fed
    - SAIL vs. sail
  - Often best to lower case everything, since users will use lowercase regardless of ‘correct’ capitalization…

- Longstanding Google example: [fixed in 2011…]
  - Query C.A.T.
  - #1 result is for “cats” (well, Lolcats) not Caterpillar Inc.

Normalization to terms

- An alternative to equivalence classing is to do expansion.
- An example of where this may be useful:
  - Enter: ***window*** Search: ***window, windows***
  - Enter: ***windows*** Search: ***Windows, windows, window***
  - Enter: ***Windows*** Search: ***Windows***
- Potentially more powerful, but less efficient.

Thesauri and soundex

- Do we handle synonyms and homonyms?
  - E.g., by hand-constructed equivalence classes
    - ***car*** = ***automobile*** ***color*** = ***colour***
  - We can rewrite to form equivalence-class terms
    - When the document contains ***automobile***, index it under ***car-automobile*** (and vice-versa)
  - Or we can expand a query
    - When the query contains ***automobile***, look under ***car*** as well

- What about spelling mistakes?
  - One approach is Soundex, which forms equivalence classes of words based on phonetic heuristics

Chapter 2
Pre-Processing & Dictionary

2.1: Indexing
2.2: Terms: tokenization
2.3: Stop-words
2.4: Normalization
2.5: Stemming
2.6: Statistics: Heaps' law, Zipf's law
2.7: Practice

Lemmatization

- Lemmatization: in linguistics, is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. [Collins English Dictionary]
- E.g.,
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors*

  → *the boy car be different color*
- Lies on linguistic analysis.

Stemming

- Stemming is the process for reducing inflected words to their "roots" before indexing.

- In IR, Stemming does not try to be linguistically correct.

- "Stemming" suggests crude affix chopping
  - language dependent
  - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

for example compressed and compression are both accepted as equivalent to compress.

for exampl compress and compress ar both accept as equival to compress

Stemming: Porter's algorithm

- Porter's Algorithm [1980] (cf. here).
- Commonest algorithm for stemming English
  - Results suggest it's at least as good as other stemming options
- Rule based.
- Five successive steps of reduction.
- Sample rules in one step.

Rule	Example
sses → ss	caresses → caress
ies → i	ponies → poni
ss → ss	caress → caress
s →	cats → cat

Size of word sensitive rules:

(m>1) EMENT →

e.g.:

replacement → replac

cement → cement

- Only the rule that allows to match the longest suffix is applied.
- See also: Online Porter's Stemmer tool, source code.

Stemming result examples

Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

- Lovins stemmer [1968]:

such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

- Porter stemmer [1980]:

such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

- Paice stemmer [1990]:

such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

Does stemming help?

- English: very mixed results.
  - Helps recall for some queries but harms precision on others.
  - E.g., operative (dentistry) ⇒ oper
- Definitely useful for Spanish, German, Finnish, …
  - 30% performance gains for Finnish!

Chapter 2
Pre-Processing & Dictionary

2.1: Indexing
2.2: Terms: tokenization
2.3: Stop-words
2.4: Normalization
2.5: Stemming
2.6: Statistics: Heaps' law, Zipf's law
2.7: Practice

Example 1: Reuters RCV1 Collection

- One year of Reuters newswire (part of 1995 and 1996).
- 800,000 documents, ~1GB of text.

REUTERS

You are here: Home > News > Science > Article

Go to a Section: U.S. International Business Markets Politics Entertainment Technology Sports Oddly Enoug

Extreme conditions create rare Antarctic clouds

Tue Aug 1, 2006 3:20am ET

Email This Article | Print This Article | Reprints

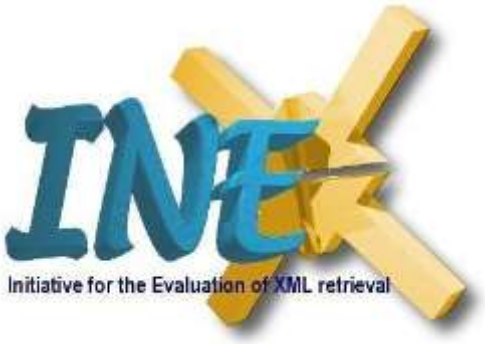
[-] Text [+]

SYDNEY (Reuters) - Rare, mother-of-pearl colored clouds caused by extreme weather conditions above Antarctica are a possible indication of global warming, Australian scientists said on Tuesday.

Known as nacreous clouds, the spectacular formations showing delicate wisps of colors were photographed in the sky over an Australian meteorological base at Mawson Station on July 25.

Example 2: INEX 2006 Wikipedia Collection

- **IN**itiative for the **E**valuation of **X**ML Retrieval.
- 2 666 000 documents, ~1.6Gb of text.



A INEX 2006 Wikipedia document

```
<article xmlns:xlink="http://www.w3.org/1999/xlink">
  <header>
    <title>Handel House Museum</title>
    <id>1707709</id>
  </header>
  <bdy>
    <image width="150px" src="London_Handel_House.jpg" type="thumb">
      <caption>Handel House. Note the <link xlink:href="310649.xml">blue plaque</link>
      </caption>
    </image>
    <p>
      The <b>Handel House Museum</b> at 25 <link xlink:href="2599649.xml">Brook
Street</link>, in the exclusive central <link xlink:href="17867.xml">London</link>
district of <link xlink:href="94167.xml">Mayfair</link> was the home of the <link
xlink:href="11867.xml">German</link> born <link xlink:href="4500.xml">baroque</link>
composer <link xlink:href="12775.xml">George Frideric Handel</link>
from 1723 until his death at the house in 1759. He composed works such as
<it><link xlink:href="149131.xml">The Messiah</link></it>, <it><link
xlink:href="811987.xml">Zadok the Priest</link></it> and the <it><link
xlink:href="1246814.xml">Fireworks Music</link></it> there.
    </p>
    <sec>
      <st>The museum</st>
      <p>
        The house has been restored to look as it did during Handel's occupancy. A typical
early 18th century London terrace house, it comprises a basement, three main storeys and an
attic, and Handel was the first occupant. The attic was later converted into a fourth full
floor. The ground floor is now a music and gift shop and the upper floors are leased to a
charity called the Handel House Trust, and have been open to the public since 8 November
2001. The interiors have been restored to the somewhat spartan style of Georgian era,
using mostly architectural elements from elsewhere, as other than the staircase, few of
the original interior features survived. The Handel House Collection Trust has assembled a
collection of Handel memorabilia, including the Byrne Collection of several hundred items,
which was acquired in 1998.
      </p>
    </sec>
  </bdy>
</article>
```

Document « The Haendel House »
(Collection « INEX 2006 Wikipédia »)

Statistics on two collections

		Reuters-RCV1	INEX/Wikipedia
N	number of documents	806 791	2 666 190
L_{ave}	mean number of tokens per doc.	222	528
M	number of uniq terms	391 523	17 661 801
	mean number of char. per token		
	including spaces and punctuation	6.04	x.x
	excluding spaces and punctuation	4.5	x.x
	mean number of char. per uniq term	7.5	x.x
T	tokens	179 158 204	1 407 654 767

Reuters-RCV1 about 1 Gb of text, 800 000 documents

	uniq terms			non zero *tf*			tokens		
	nb.	Δ%	%T	nb.	Δ%	%T	nb.	Δ%	%T
no filtering	484 494			109 971 179			197 879 290		
without numbers	473 723	−2	−2	100 680 242	−8	−8	179 158 204	−9	−9
with case folding	391 523	−17	−19	96 969 056	−3	−12	179 158 204	−0	−9
30 stop words	391 493	−0	−19	83 390 443	−14	−24	121 857 825	−31	−38
150 stop words	391 373	−0	−19	67 001 847	−30	−39	94 516 599	−47	−52
with stemming	322 383	−17	−33	63 812 300	−4	−42	94 516 599	−0	−52

Vocabulary vs. collection size

- How big is the term vocabulary?
  - That is, how many distinct words are there?
- Can we assume an upper bound?
  - Not really: At least $70^{20} = 10^{37}$ different words of length 20
- In practice, the vocabulary will keep growing with the collection size
  - Especially with Unicode ☺

Vocabulary vs. collection size: Heaps' law [1978]

- Heap's Law models the number of words in the vocabulary as a function of the corpus size:
  - What is the number of unique words appearing in a corpus of size N words?
  - This determines how the size of the inverted index will scale with the size of the corpus .

$$M = kT^b$$

- With:
  - M is the size of the vocabulary.
  - T is the number of tokens in the collection.
  - k, b depend on the collection type. Typical values:
    - $30 \leq k \leq 100$
    - $b \approx 0.5$

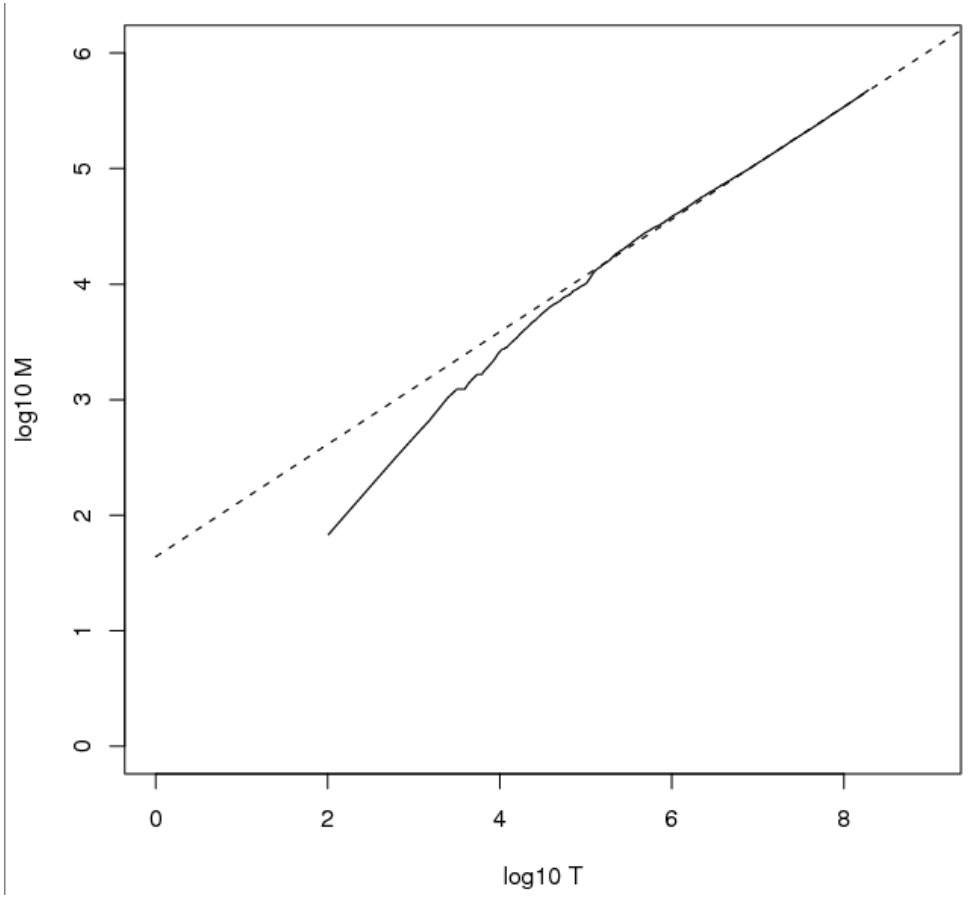
- In a log-log plot of vocabulary size M vs. T, Heaps' law predicts a line with slope about ½
  - It is the simplest possible relationship between the two in log-log space
  - An empirical finding ("empirical law")

Heaps' law for Reuters RCV1

- Good empirical fit!
- Dashed line is the best least squares fit:

  $\log_{10} M = 0.49 \log_{10} T + 1.64$.

- Thus:
  - $M = 10^{1.64} T^{0.49}$
  - $k = 10^{1.64} \approx 44$ and $b = 0.49$.
- For first 1,000,020 tokens ($\log_{10}$: 6), law predicts 38,323 terms; actually, 38,365 terms



Term distribution: Zipf's law [1949]

- Term distribution: in natural language, there are a few very frequent terms and very many very rare terms.

- Zipf's Law models the distribution of terms in a corpus:
  - How many times does the ith most frequent word appears in a corpus of size N words?
  - Important for determining index terms and properties of compression algorithms.

- Zipf's law:

The i^{th} most frequent term has frequency proportional to $\frac{1}{i}$

$$cf_i \approx \frac{K}{i}$$

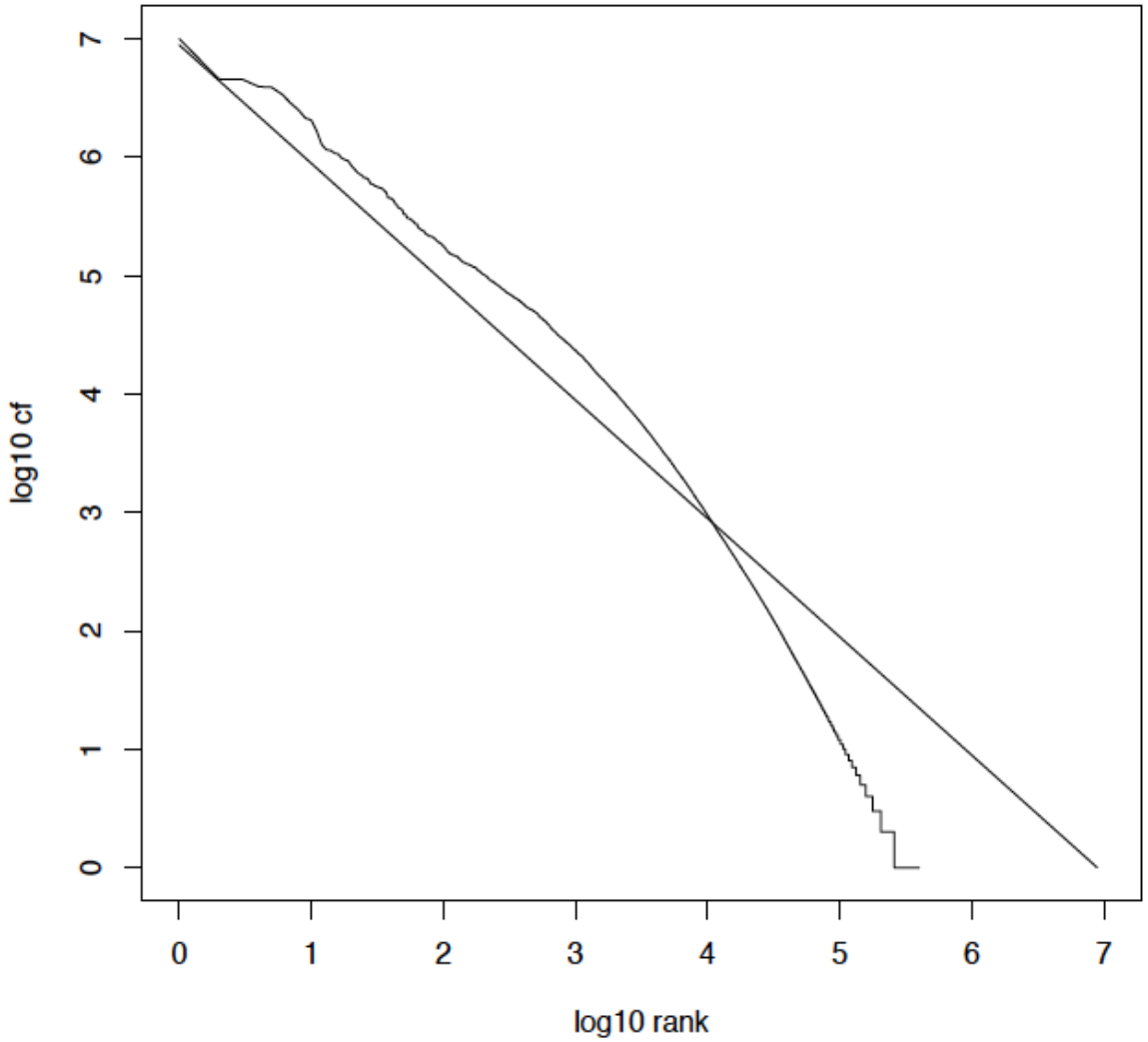
- With:
  - Collection frequency: $cf(t) = \sum_d tf_{t,d}$
    - *cf(t)*: the number of occurrences of the term *t* in the collection.
    - cf_i : the number of occurrences of the i^{th} most frequent term in the collection.
  - *K*: a normalizing constant.

Term distribution: Zipf's law [1949]

- Main Characteristics
  - a few elements occur very frequently
  - many elements occur very infrequently
  - frequency of words in the text falls very rapidly

- What kinds of data exhibit a Zipf distribution?
  - Words in a text collection
  - Library book checkout patterns
  - Document Size on Web (Cunha & Crovella)
  - Length of Web navigational sequences (Cooley, Mobasher, Srivastava)
  - Item popularity in E-Commerce
  - Number of movies/artists selected/rated in streaming services

Zipf's law for Reuters RCV1



Example of Frequent Words

Frequent Word	Number of Occurrences	Percentage of Total
the	7,398,934	5.9
of	3,893,790	3.1
to	3,364,653	2.7
and	3,320,687	2.6
in	2,311,785	1.8
is	1,559,147	1.2
for	1,313,561	1
The	1,144,860	0.9
that	1,066,503	0.8
said	1,027,713	0.8

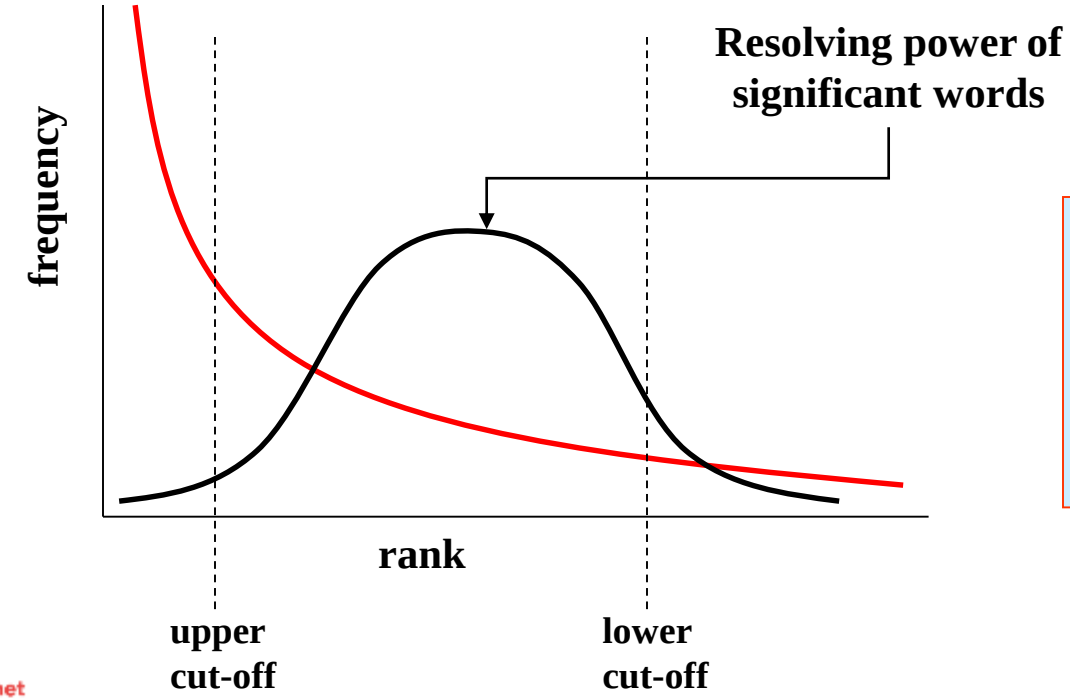
- Frequencies from 336,310 documents in the 1 GB TREC Volume 3 Corpus
  - 125,720,891 total word occurrences
  - 508,209 unique words

Zipf's Law and Indexing

- The most frequent words are poor index terms
  - they occur in almost every document
  - they usually have no relationship to the concepts and ideas represented in the document
- Extremely infrequent words are poor index terms
  - may be significant in representing the document
  - but, very few documents will be retrieved when indexed by terms with the frequency of one or two
- Index terms in between
  - a high and a low frequency threshold are set
  - only terms within the threshold limits are considered good candidates for index terms

Resolving Power

- Resolving Power: the ability of words to discriminate content.
- Zipf (and later H.P. Luhn) postulated that the resolving power of significant words reached a peak at a rank order position half way between the two cut-offs.



The actual cut-off are determined by trial and error, and often depend on the specific collection.

Chapter 2
Pre-Processing & Dictionary

2.1: Indexing
2.2: Terms: tokenization
2.3: Stop-words
2.4: Normalization
2.5: Stemming
2.6: Statistics: Heaps' law, Zipf's law
2.7: Practice

Lexical Analysis (Python Example)

```
import re

input_file = open('wordtest.txt', 'r')
word_count = {}
for line in input_file.readlines():
    words = line.strip()
    words = re.findall('\w+', line)
    for word in words:
        word = word.lower()
        # Special case if we're seeing this word for the first time
        if not word in word_count:
            word_count[word] = 1
        else:
            word_count[word] = word_count[word] + 1

word_index = sorted(word_count.keys())
for word in word_index:
    print word, word_count[word]
```

(source)

```
able 1
about 23
absurd 1
acceptance 1
accustomed 1
across 2
actually 1
ada 1
added 1
addressing 1
adjourn 1
adoption 1
advice 1
advisable 2
advise 1
afraid 4
```

Lexical Analysis (NLTK Example)

In [1]:
```
import nltk
```

In [2]:
```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
```

In [3]:
```
example = "Glimpse is an indexing and query system that allows for search through a file system or document collection." \
        + "The main processes in an retrieval system are document indexing, query processing, query evaluation."
```

In [4]:
```
words = word_tokenize(example)

print(words)
```

```
['Glimpse', 'is', 'an', 'indexing', 'and', 'query', 'system', 'that', 'allows', 'for', 'search', 'through', 'a', 'file', 'syste
m', 'or', 'document', 'collection.The', 'main', 'processes', 'in', 'an', 'retrieval', 'system', 'are', 'document', 'indexing',
',', 'query', 'processing', ',', 'query', 'evaluation', '.']
```

In [5]:
```
stop_words = nltk.corpus.stopwords.words('english')

print(stop_words)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'y
ourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those',
'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'a
n', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'b
etween', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
f', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'ar
en', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "have
n't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "should
n't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

In [6]:
```
filtered_words = [w for w in words if w not in stop_words]

print(filtered_words)
```

```
['Glimpse', 'indexing', 'query', 'system', 'allows', 'search', 'file', 'system', 'document', 'collection.The', 'main', 'process
es', 'retrieval', 'system', 'document', 'indexing', ',', 'query', 'processing', ',', 'query', 'evaluation', '.']
```

In [7]:
```
ps = PorterStemmer()
stemmed = [ps.stem(w) for w in filtered_words]
print(stemmed)
```

```
['glimps', 'index', 'queri', 'system', 'allow', 'search', 'file', 'system', 'document', 'collection.th', 'main', 'process', 're
triev', 'system', 'document', 'index', ',', 'queri', 'process', ',', 'queri', 'evalu', '.']
```

Lexical Analysis (Scikit-learn Example)

```
In [18]: docs = [u"Glimpse is an indexing and query system that allows for search through a file system or document collection. ", \
                 u"The main processes in an retrieval system are document indexing, query processing, query evaluation. ", \
                 u"Glimpse is the default search engine in a larger information retrieval system and a web search engines. "]

In [28]: from sklearn.feature_extraction.text import CountVectorizer

In [29]: tf = CountVectorizer(stop_words=stop_words)
         tf_matrix = tf.fit_transform(docs)

In [30]: tf_matrix.shape
Out[30]: (3, 20)

In [33]: print(tf_matrix)
  (0, 8)        1
  (0, 9)        1
  (0, 15)       1
  (0, 18)       2
  (0, 0)        1
  (0, 17)       1
  (0, 7)        1
  (0, 3)        1
  (0, 1)        1
  (1, 9)        1
  (1, 15)       2
  (1, 18)       1
  (1, 3)        1
  (1, 12)       1
  (1, 13)       1
  (1, 16)       1
  (1, 14)       1
  (1, 6)        1
  (2, 8)        1
  (2, 18)       1
  (2, 17)       2
  (2, 16)       1
  (2, 2)        1
  (2, 4)        1
  (2, 11)       1
  (2, 10)       1
  (2, 19)       1
  (2, 5)        1

In [36]: print(tf.get_feature_names())
         print(tf_matrix.A)
['allows', 'collection', 'default', 'document', 'engine', 'engines', 'evaluation', 'file', 'glimpse', 'indexing', 'informatio
n', 'larger', 'main', 'processes', 'processing', 'query', 'retrieval', 'search', 'system', 'web']
[[1 1 0 1 0 0 0 1 1 1 0 0 0 0 0 1 0 1 2 0]
 [0 0 0 1 0 0 1 0 0 1 0 0 1 1 1 2 1 0 1 0]
 [0 0 1 0 1 1 0 0 1 0 1 1 0 0 0 0 1 2 1 1]]
```

Information Retrieval: Outline

0) Introduction

1) IR Model & Boolean Retrieval

2) Pre-Processing & Dictionary

3) Ranked Retrieval

4) Experimental Evaluation of IR

5) Structured Retrieval

6) Link Analysis for IR