CS4051

Information Retrieval

Week 04

Muhammad Rafi

February 14, 2023

Agenda

- Why Compression?
- Heap’s Law
- Zipf’s Law
- Dictionary compression
- Postings compression
- Summary

Why Compression

- Use less disk space (saves money)
- Keep more stuff in memory (increases speed)
- Increase speed of transferring data from disk to memory (increases speed)
  - [read compressed data and decompress] is faster than [read uncompressed data]
- Premise: Decompression algorithms are fast
  - True of the decompression algorithms we use
- In most cases, retrieval system runs faster on compressed postings lists than on uncompressed postings lists.

Why Compression

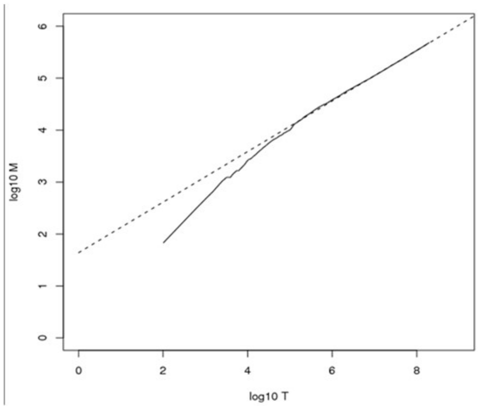
- First, we will consider space for dictionary
  - Make it small enough to keep in main memory
- Then the postings
  - Reduce disk space needed, decrease time to read from disk
  - Large search engines keep a significant part of postings in memory
- (Each postings entry is a docID)

Heap's Law

- HEAPS' LAW helps in estimates vocabulary size as a function of collection size

  M = kT^b

- Vocabulary size M as a function of collection size T
- For RCV1, the dashed line
  $\log_{10} M = 0.49 \log_{10} T + 1.64$
  is the best least squares fit.
  Thus, $M = 10^{1.64} T^{0.49}$
  so $k = 10^{1.64} \approx 44$
  and b = 0.49.



Heap's Law

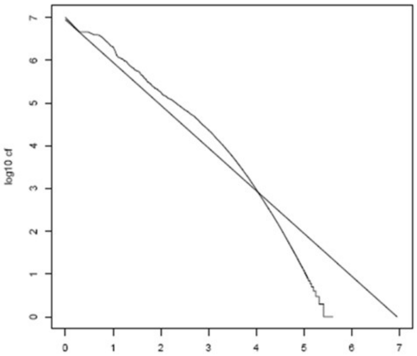
- The parameter k is quite variable because vocabulary growth depends a lot on the nature of the collection and how it is processed.
- Case-folding and stemming reduce the growth rate of the vocabulary, whereas including numbers and spelling errors increase it.
- Regardless of the values of the parameters
- for a particular collection, Heaps' law suggests that
  - The dictionary size continues to increase with more documents in the collection, rather than a maximum vocabulary size being reached,
  - The size of the dictionary is quite large for large collections.

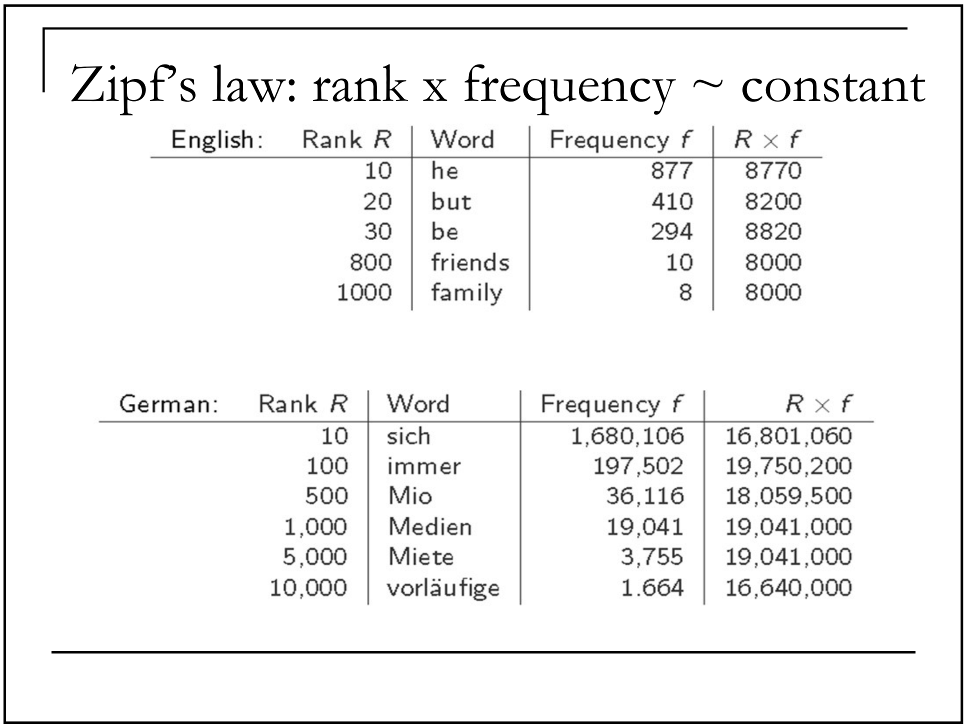
Zipf’ s law

- Heaps’ Law gives the vocabulary size in collections.
- We also study the relative frequencies of terms.
- In a natural language, there are very few very frequent terms and very many very rare terms.
- Zipf’ s law: The *i*th most frequent term has frequency proportional to 1/*i* .
- $\mathrm{cf}_i \propto 1/i = a/i$ where *a* is a normalizing constant
- cf_i is collection frequency: the number of occurrences of the term t_i in the collection.

Zipf consequences

- If the most frequent term (*the*) occurs cf_1 times, then
  - the second most frequent term (*of*) occurs $\mathrm{cf}_1/2$ times
  - the third most frequent term (*and*) occurs $\mathrm{cf}_1/3$ times …
- Equivalent: $\mathrm{cf}_i = a/i$, so
  - $\log \mathrm{cf}_i = \log a - \log i$
  - Linear relationship between $\log \mathrm{cf}_i$ and $\log i$

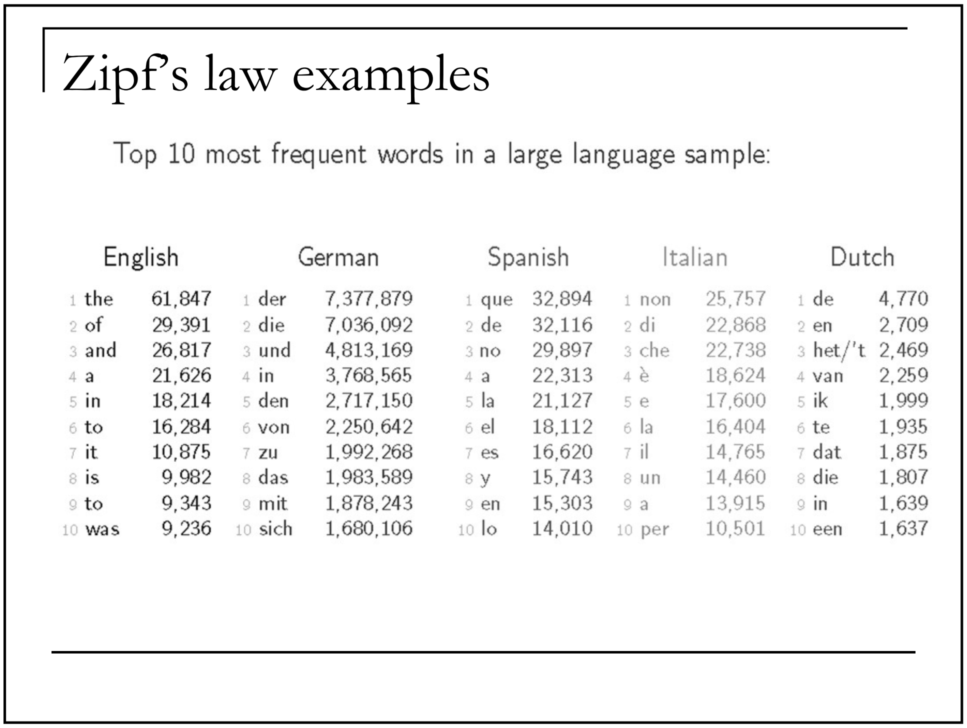




Zipf's law: rank x frequency ~ constant

English: Rank R	Word	Frequency f	$R \times f$
10	he	877	8770
20	but	410	8200
30	be	294	8820
800	friends	10	8000
1000	family	8	8000

German: Rank R	Word	Frequency f	$R \times f$
10	sich	1,680,106	16,801,060
100	immer	197,502	19,750,200
500	Mio	36,116	18,059,500
1,000	Medien	19,041	19,041,000
5,000	Miete	3,755	19,041,000
10,000	vorläufige	1.664	16,640,000

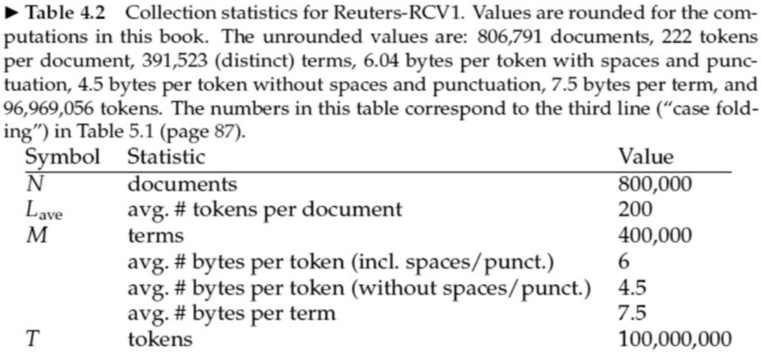


Zipf's law examples

Top 10 most frequent words in a large language sample:

English			German			Spanish			Italian			Dutch		
1	the	61,847	1	der	7,377,879	1	que	32,894	1	non	25,757	1	de	4,770
2	of	29,391	2	die	7,036,092	2	de	32,116	2	di	22,868	2	en	2,709
3	and	26,817	3	und	4,813,169	3	no	29,897	3	che	22,738	3	het/'t	2,469
4	a	21,626	4	in	3,768,565	4	a	22,313	4	è	18,624	4	van	2,259
5	in	18,214	5	den	2,717,150	5	la	21,127	5	e	17,600	5	ik	1,999
6	to	16,284	6	von	2,250,642	6	el	18,112	6	la	16,404	6	te	1,935
7	it	10,875	7	zu	1,992,268	7	es	16,620	7	il	14,765	7	dat	1,875
8	is	9,982	8	das	1,983,589	8	y	15,743	8	un	14,460	8	die	1,807
9	to	9,343	9	mit	1,878,243	9	en	15,303	9	a	13,915	9	in	1,639
10	was	9,236	10	sich	1,680,106	10	lo	14,010	10	per	10,501	10	een	1,637

RCV1 Collection



► **Table 4.2** Collection statistics for Reuters-RCV1. Values are rounded for the computations in this book. The unrounded values are: 806,791 documents, 222 tokens per document, 391,523 (distinct) terms, 6.04 bytes per token with spaces and punctuation, 4.5 bytes per token without spaces and punctuation, 7.5 bytes per term, and 96,969,056 tokens. The numbers in this table correspond to the third line ("case folding") in Table 5.1 (page 87).

Symbol	Statistic	Value
N	documents	800,000
L_{ave}	avg. # tokens per document	200
M	terms	400,000
	avg. # bytes per token (incl. spaces/punct.)	6
	avg. # bytes per token (without spaces/punct.)	4.5
	avg. # bytes per term	7.5
T	tokens	100,000,000

REUTERS

Dictionary Compression

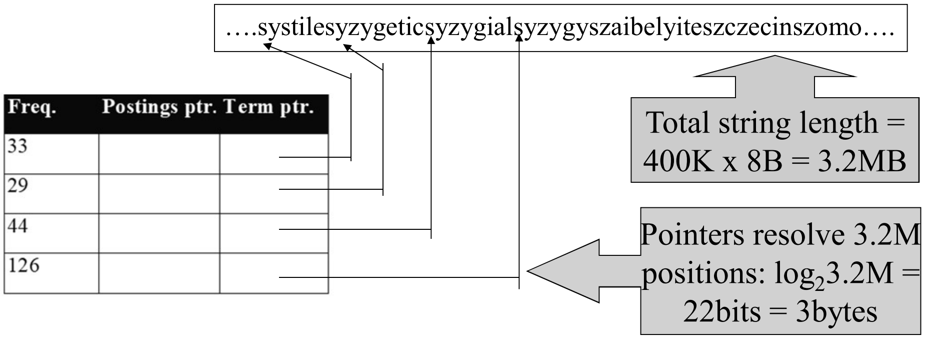
- Written English- 4.5 character / words
- Ave. dictionary word in English: ~8 characters
- Fixed length words for dictionary (20 bytes)
- Fixed length term fields with arrays are certainly wasteful.
- Short words dominate token counts but not type average.

Dictionary Compression –RVC 1

- 4 bytes per term for Freq.
- 4 bytes per term for pointer to Postings.
- 20 bytes for each term
- 400K terms x 28 ⇒ 11.2 MB

Dictionary Compression

- Store dictionary as a (long) string of characters:
  - Pointer to next word shows end of current word
  - Hope to save up to 60% of dictionary space.



….systilesyzygeticsyzygialsyzygyszaibelyiteszczecinszomo….

Freq.	Postings ptr.	Term ptr.
33		
29		
44		
126		

Total string length = 400K x 8B = 3.2MB

Pointers resolve 3.2M positions: $\log_2 3.2M$ = 22bits = 3bytes

Dictionary Compression –RVC 1

- 4 bytes per term for Freq.
- 4 bytes per term for pointer to Postings.
- 3 bytes per term pointer

Now avg. 11 bytes/term, not 20.

- Avg. 8 bytes per term in term string
- 400K terms x 19 ⇒ 7.6 MB (against 11.2MB for fixed width)

Posting Compression

- The postings file is much larger than the dictionary, factor of at least 10.
- Key desideratum: store each posting compactly.
- A posting for our purposes is a docID.
- For Reuters (800,000 documents), we would use 32 bits per docID when using 4-byte integers.
- Alternatively, we can use $\log_2 800{,}000 \approx 20$ bits per docID.
- Our goal: use far fewer than 20 bits per docID.

Posting Compression

- Variable length Encoding
  - Aim:
    - For ***arachnocentric***, we will use ~20 bits/gap entry.
    - For ***the***, we will use ~1 bit/gap entry.
  - If the average gap for a term is *G*, we want to use ~$\log_2 G$ bits/gap entry.
  - Key challenge: encode every integer (gap) with about as few bits as needed for that integer.
  - This requires a *variable length encoding*
  - Variable length codes achieve this by using short codes for small numbers

Index Compression

- We can now create an index for highly efficient Boolean retrieval that is very space efficient
- Only 4% of the total size of the collection
- Only 10-15% of the total size of the text in the collection
- However, we've ignored positional information
- Hence, space savings are less for indexes used in practice
  - But techniques substantially the same.

Summary

- Compression plays an important role for large scale IR systems.