Information Retrieval
Topic: Index Compression (Part-1)
Lecture-21

Prepared By

Dr. Rasmita Rautray & Dr. Rasmita Dash
Associate Professor
Dept. of CSE

Dictionary compression

- The dictionary is small compared to the postings file.
- But we want to keep it in memory.
- Also: competition with other applications, cell phones,
- onboard computers, fast startup time
- So compressing the dictionary is important.

Why compression in information retrieval?

- First, we will consider space for dictionary
  - Main motivation for dictionary compression: make it small enough to keep in main memory
- Then for the postings file
  - Motivation: reduce disk space needed, decrease time needed to read from disk

Lossy vs. lossless compression

- Lossy compression: Discard some information Several of the preprocessing steps we frequently use can be viewed as lossy compression:
- Lossless compression: All information is preserved.

Term Statistics

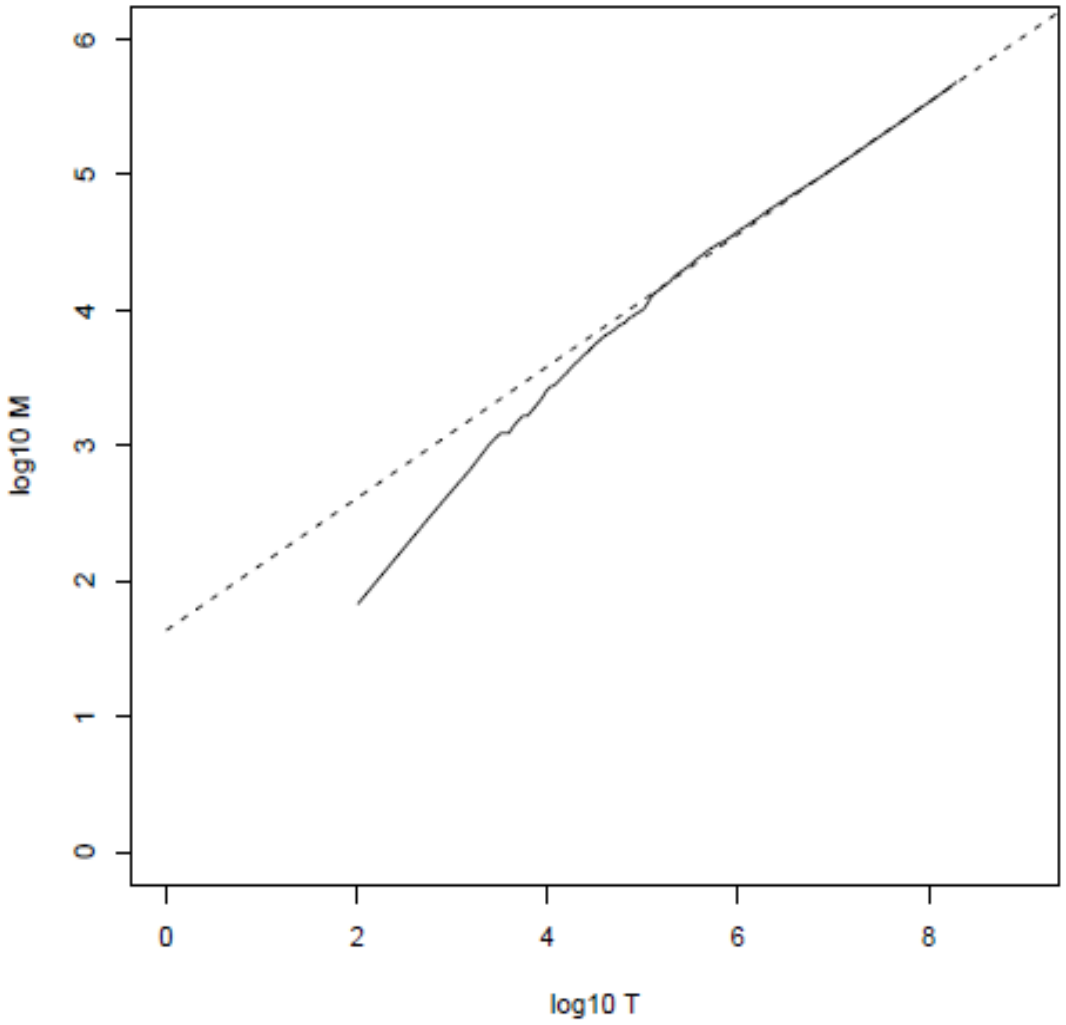
How big is the term vocabulary?

- That is, how many distinct words are there?
- Can we assume there is an upper bound?
- Not really: At least $7020 \approx 1037$ different words of length 20.
- The vocabulary will keep growing with collection size.

Heaps' law

- Heaps' law: $M = kT^b$
- M is the size of the vocabulary, T is the number of tokens in the collection.
- Typical values for the parameters k and b are: $30 \leq k \leq 100$ and $b \approx 0.5$.
- Heaps' law is linear in log-log space.
  - It is the simplest possible relationship between collection size and vocabulary size in log-log space.
  - Empirical law

Heaps' law for Reuters



Vocabulary size M as a function of collection size T (number of tokens) for Reuters-RCV1. For these data, the dashed line $\log_{10} M = 0.49 * \log_{10} T + 1.64$ is the best least squares fit. Thus, $M = 10^{1.64} T^{0.49}$ and $k = 10^{1.64} \approx 44$ and $b = 0.49$.

Empirical fit for Reuters

- Good, as we just saw in the graph.
- Example: for the first 1,000,020 tokens Heaps' law predicts 38,323 terms:

$$44 \times 1000020^{0.49} \approx 38{,}323$$

- The actual number is 38,365 terms, very close to the prediction.
- Empirical observation: fit is good in general.

Zipf's law

Zipf's law: The i^{th} most frequent term has frequency proportional to $1/i$.

$\text{cf}_i \propto \frac{1}{i}$

cf is collection frequency: the number of occurrences of the term in the collection.

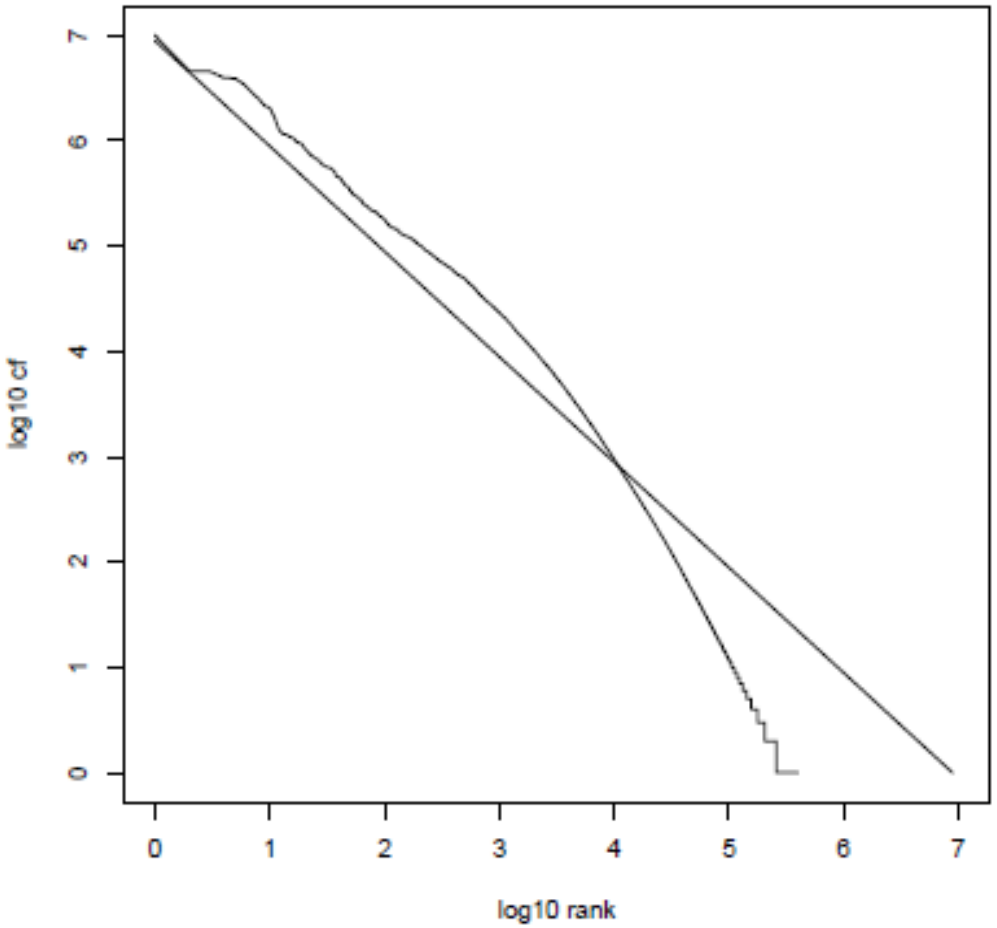
So if the most frequent term (*the*) occurs cf_1 times, then the second most frequent term (*of*) has half as many occurrences $\text{cf}_2 = \frac{1}{2}\text{cf}_1$ ...

...and the third most frequent term (*and*) has a third as many occurrences $\text{cf}_3 = \frac{1}{3}\text{cf}_1$ etc.

Equivalent: $\text{cf}_i = ci^k$ and $\log \text{cf}_i = \log c + k \log i$ (for $k = -1$)

Example of a power law

Zipf's law for Reuters



Fit is not great. What is important is the key insight: Few frequent terms, many rare terms.

Dictionary compression

- The dictionary is small compared to the postings file.
- But we want to keep it in memory.
- Also: competition with other applications, cell phones, onboard computers, fast startup time
- So compressing the dictionary is important.

Recall: Dictionary as array of fixed-width entries

term	document frequency	pointer to postings list
a	656,265	→
aachen	65	→
…	…	…
zulu	221	→
space needed: 20 bytes	4 bytes	4 bytes

Space

- for Reuters: (20+4+4)*400,000 = 11.2 MB

Limitation of Fixed-width entries

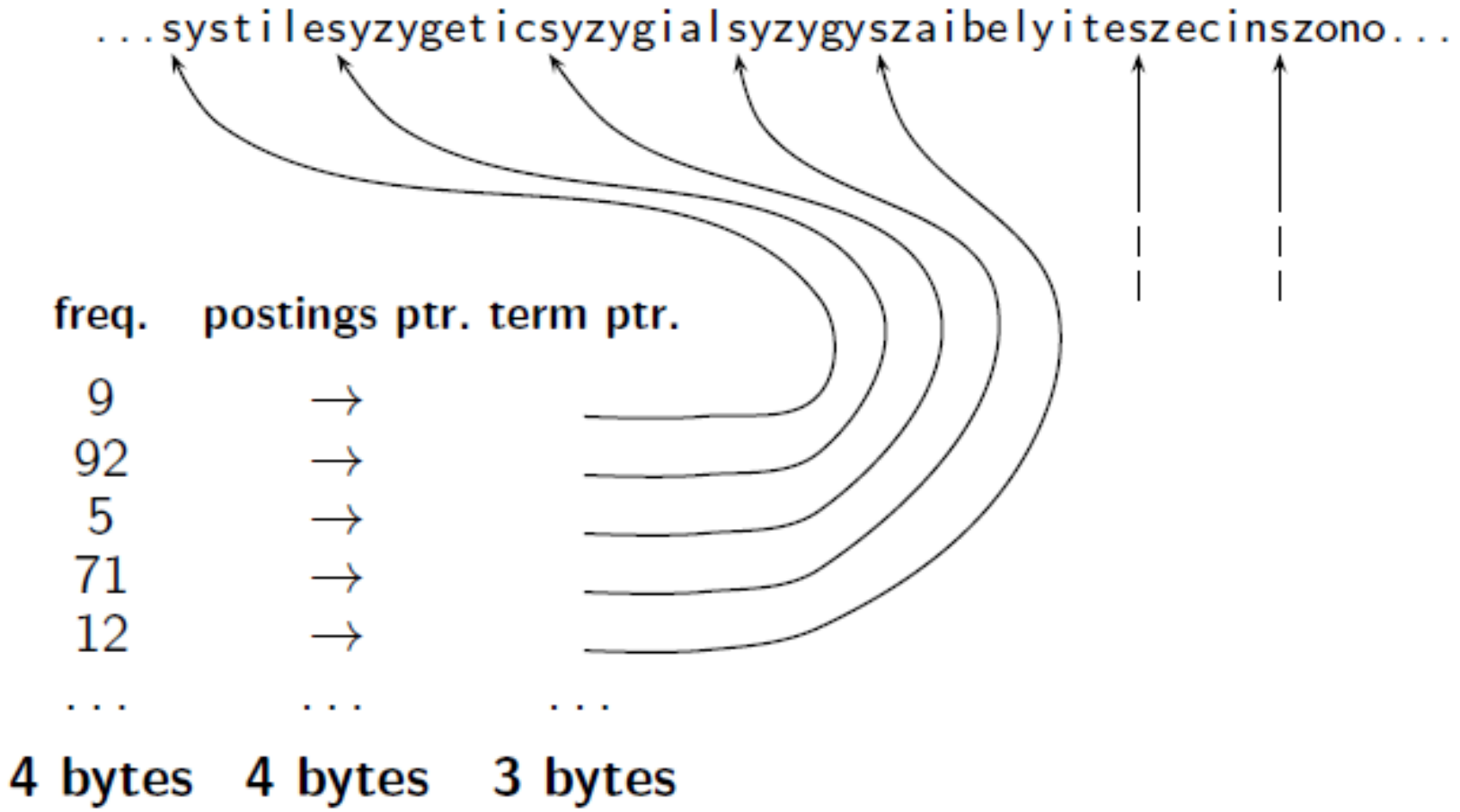
- Most of the bytes in the term column are wasted.
  - We allot 20 bytes for terms of length 1.
- We can't handle hydrochlorofluorocarbons and supercalifragilisticexpialidocious
- Average length of a term in English: 8 characters (or a little bit less)
- How can we use on average 8 characters per term?

Dictionary as a string

- It store the dictionary terms as one long string of characters.
- Term pointers mark the end of the preceding term and the beginning of the next.

  For example, the first three terms in this example are systile, syzygetic, and syzygial

Dictionary as a string



Space for dictionary as a string

- 4 bytes per term for frequency
- 4 bytes per term for pointer to postings list
- 8 bytes (on average) for term in string
- 3 bytes per pointer into string

(need $\log_2 8 \times 400{,}000 < 24$ bits to resolve $8 \cdot 400{,}000$ positions)

- Space: 400,000 × (4 +4 +3 + 8) = 7.6MB (compared to 11.2 MB for fixed-width array)

Thank You