Processamento e Recuperação de Informação

Querying

Departamento de Engenharia Informática
Instituto Superior Técnico

1º Semestre
2018/2019

Outline

Bibliography

Managing Gigabytes: Compressing and Indexing Documents and Images - 2nd edition Ian H. Witten, Alistair Moffat, Timothy C. Bell Morgan Kaufmann 2000

Outline

The Cosine Measure

Similarity function

$$sim(d_j, q) = \frac{1}{W_d \times W_q} \times \sum_{i=1}^{t} w_{i,j} \times w_{i,q}$$

- W_d is the document norm
- W_q is the query norm
  - irrelevant for ranking
- $w_{i,j} = f_{i,j} \times idf_i$

Implementation problems

- Processing the documents is unfeasible
- Processing the index is expensive
  - The index needs to store $f_{d,t}$
  - The inverted lists can be huge
- We need only the top r documents from a sorted list of N documents, where $r \ll N$

Outline

An Example Inverted File

Lexicon

Num.	Term	Add.
1	best	0000
2	expedient	0024
3	government	0032
4	governs	0064
5	inexpedient	0080
6	least	0088
7	machines	0096
8	manufactured	0104
9	mass	0108
10	men	0116
11	purpose	0132
12	serve	0140
13	state	0156
14	wooden	0164

Inverted File

Inverted list
$(1;1),(2;1),(5;1)$
$(5;1)$
$(1;1),(2;1),(5;1),(6;1)$
$(1;1),(2;1)$
$(6;1)$
$(1;1)$
$(3;1)$
$(4;1)$
$(3;1)$
$(3;2),(4;1)$
$(4;1)$
$(3;1),(4;1)$
$(3;1)$
$(4;1)$

Computing the Cosine Value

1. Set $A \leftarrow \{\}$
2. For each query term $t \in Q$
   1. Stem t
   2. Search the lexicon
   3. Get f_t and the address of I_t
   4. Set $idf_t \leftarrow \log(N/f_t)$
   5. Read the inverted list I_t
   6. For each $(d, f_{d,t})$ pair in I_t
      1. If $A_d \notin A$ then
         Set $A_d \leftarrow 0$
         Set $A \leftarrow A \cup \{A_d\}$
      2. Set $A_d \leftarrow A_d + f_{d,t} \times idf_t$
3. For each $A_d \in A$, set $A_d \leftarrow A_d / W_d$
4. For $1 \leq i \leq r$
   1. Select d such that $A_d = \max\{A\}$
   2. Look up the address of d
   3. Retrieve d
   4. Set $A \leftarrow A - \{A_d\}$

Main Problems

- How to efficiently obtain the document norms?
- How to efficiently process the inverted lists?
- How to efficiently select the top-k documents?

Outline

Storing the Document Norms

The problem

- All in memory: too expensive
  - E.g. 4×10^9 docs $\Rightarrow$ 484Gb
- All in disk: too slow
  - E.g. can take several seconds to read the norms

Storing the Document Norms

The problem

- All in memory: too expensive
  - E.g. 4×10^9 docs $\Rightarrow$ 484Gb
- All in disk: too slow
  - E.g. can take several seconds to read the norms

The solution

- Use approximations
- Read only selected document weights

Approximate Document Norms

- Document norms are real numbers
  - Need 4 to 8 bytes of storage
- Real numbers can approximated by b-bit values

Approximate Document Norms

- Document norms are real numbers
  - Need 4 to 8 bytes of storage
- Real numbers can approximated by b-bit values

Approximate Norm

Using b bits to approximate x, such that $L \leq x \leq U$:

$$c = \left\lfloor \frac{x - L}{U - L + \epsilon} 2^b \right\rfloor$$

where c is the code for x.

An Example

Consider $10 \leq x \leq 18$, $b = 2$, $\epsilon = 0.1$:

- For $x = 15.3$:

$$c = \left\lfloor \frac{15.3 - 10}{18 - 10 + 0.1} \times 2^2 \right\rfloor = \lfloor 2.617 \rfloor = 2 = 10$$

- For $x = 12.4$:

$$c = \left\lfloor \frac{12.4 - 10}{8.1} \times 4 \right\rfloor = \lfloor 1.185 \rfloor = 1 = 01$$

- For $x = 17.9$:

$$c = \left\lfloor \frac{17.9 - 10}{8.1} \times 4 \right\rfloor = \lfloor 3.901 \rfloor = 3 = 11$$

Approximation Error

- The previous approximation assumes we are distributing the norms over equal sized-buckets
- However, some norms occur more frequently than others
  - Short documents are more frequent than long documents
  - Small values introduce higher error
- Thus, we need more precision for short documents

Reducing the Relative Error

Geometric-sized buckets

Using b bits to approximate x, such that $L \leq x \leq U$:

$$B = \left(\frac{U+\epsilon}{L}\right)^{2^{-b}}$$
$$c = f(x) = \lfloor \log_B(x/L) \rfloor = \left\lfloor \frac{\log(x/L)}{\log B} \right\rfloor$$

Range of values for x:

$$g(c) \leq x < g(c+1)$$

where

$$g(c) = L \cdot B^c$$

An Example

Consider $10 \leq x \leq 18$, $b = 2$, $\epsilon = 0.1$:

$$B = \left(\frac{18.1}{10.0}\right)^{2^{-2}} = 1.160$$

If $x = 15.3$:

$$c = f(15.3) = \lfloor \log_{1.16}(15.3/10.0) \rfloor = 2 = 10$$

Range for $c = 2$:

$$13.456 \leq x < 15.610$$

Using the Approximate Weights

3. For $1 \leq d \leq N$
   Set $A_d \leftarrow A_d / g(c_d)$
4. Set $H \leftarrow$ top r values of A_d
5. For $d \in H$
   1. Read W_d from disk
   2. Get the address of document d
   3. Set $A_d \leftarrow A_d \cdot g(c_d) / W_d$
6. For $1 \leq d \leq N$
   1. If $A_d \notin H \wedge A_d > \min\{H\}$ then
      1. Read W_d from disk
      2. Set $A_d \leftarrow A_d \cdot g(c_d) / W_d$
      3. If $A_d > \min\{H\}$ then
         Set $H \leftarrow H - \{\min\{H\}\} + \{A_d\}$
         Get address of document d

Outline

Storing the Accumulators

- One accumulator per document may be too expensive
- Solution: use pruning strategies
- Example:
  - Process terms with higher weights first
  - Stop creating accumulators when weight is below a threshold

Frequency-Sorted Indexes

- Order the inverted lists by $f_{d,t}$, followed by d

$$\langle 5; (1,2), (2,2), (3,5), (4,1), (5,2) \rangle$$
$$\Downarrow$$
$$\langle 5; (3,5), (1,2), (2,2), (5,2), (4,1) \rangle$$

- Advantage: allows easy access to the most important terms

Storing Frequency-Sorted Lists

Documents are grouped by frequencies

$$\langle 5; (3,5), (1,2), (2,2), (5,2), (4,1) \rangle$$
$$\Downarrow$$
$$\langle (5, 1 : 3), (2, 3 : 1, 2, 5), (1, 1 : 4) \rangle$$

- d-gap coding can be used within each block
- frequency gaps can also be coded

Storing Frequency-Sorted Lists

Documents are grouped by frequencies

$$\langle 5; (3,5), (1,2), (2,2), (5,2), (4,1) \rangle$$

$$\Downarrow$$

$$\langle (5, 1 : 3), (2, 3 : 1, 2, 5), (1, 1 : 4) \rangle$$

- *d*-gap coding can be used within each block
- frequency gaps can also be coded

Processing the Lists

1. Lists are processed in parallel, one block at a time
2. The block with the highest $TF \times IDF$ value is always processed first

Processing Frequency-Sorted Indexes

Advantages

- More accuracy: if a cut threshold is used, the lost documents will have small importance
- Less processing: the larger blocks (which are those with low frequencies) have a smaller chance of being processed
- Less disk transfer: lists can be read one block at a time
- No loss in retrieval effectiveness: experiments show that it does not loose and sometimes improves the results

Drawback

May not appropriate for Boolean queries (why?)

Outline

Sorting the Ranked Documents

The problem

- Sorting all documents requires $N \log N$ operations
  - For large collections this implies several seconds
- However: we are only interested in the k top documents, where $k \ll N$

Sorting the Ranked Documents

The problem

- Sorting all documents requires $N \log N$ operations
  - For large collections this implies several seconds
- However: we are only interested in the k top documents, where $k \ll N$

The solution

Use a heap data structure

Selecting the Top r documents

1. Set $H \leftarrow \{\}$
2. For $1 \leq d \leq r$
    1. Set $A_d \leftarrow A_d / W_d$
    2. Get the address of document d
    3. Set $H \leftarrow H + \{A_d\}$
3. Build H into a heap
4. For $r + 1 \leq d \leq N$
    1. Set $A_d \leftarrow A_d / W_d$
    2. If $A_d > \min\{H\}$ then
        1. Set $H \leftarrow H - \min\{H\} + \{A_d\}$
        2. Sift H
        3. Get the address of document d
5. For $1 \leq i \leq r$
    1. Select d such that $A_d = \max\{H\}$
    2. Retrieve d
    3. Set $H \leftarrow H - \{A_d\}$

Algorithm Complexity

Worst case

$$2r + (N - r) + 2(N - r)\log r + r\log r$$

Expected value

$$2r + N + 1.4r\log r\log(N/r) + r\log r$$

Algorithm Complexity

Worst case

$$2r + (N - r) + 2(N - r)\log r + r\log r$$

Expected value

$$2r + N + 1.4r\log r\log(N/r) + r\log r$$

Example

For $N = 1\,000\,000$ and $r = 100$:

- Full sort: 20 000 000 comparisons
- Heap: 1 013 000 comparisons

Questions?