Table of Contents

Inverted Files for Text Search Engines

JUSTIN ZOBEL And ALISTAIR MOFFAT

by M. Levent KOC

Sample Collection for examples:

1. The cleaner job is clean
2. The cleaner cleans the big old house in the town
3. The old cleaners like sleeping
4. It is only big old house that is clean
5. The cleaner cleans houses that are not clean
6. The clean operations are performed at only night

Collection with only case-folding

are at big clean cleaner cleaners cleans house houses in is it job like night not old only operations performed sleeping that the town

Collection with case-folding and stemming

are at big clean house in is it job like night not old only operations performed sleeping that the town

Collection with case-folding, stemming and stopping

big clean house job like night old operations performed sleeping town

Some terms used in the paper

—fd,t , the frequency of term t in document d ;
—fq,t , the frequency of term t in the query;
—ft , the number of documents containing one or more
—Ft , the number of occurrences of term t in the collection;
—N , the number of documents in the collection; and
—n , the number of indexed terms in the collection.

Some conventions

(1) Less weight is given to terms that appear in many documents;
(2) More weight is given to terms that appear many times in a document; and
(3) Less weight is given to documents that contain many terms.

Cosine similarity

$$w_{q,t} = \ln\left(1 + \frac{N}{f_t}\right) \qquad w_{d,t} = 1 + \ln f_{d,t}$$

$$W_d = \sqrt{\sum_t w_{d,t}^2} \qquad W_q = \sqrt{\sum_t w_{q,t}^2}$$

$$S_{q,d} = \frac{\sum_t w_{d,t} \cdot w_{q,t}}{W_d \cdot W_q}.$$

Equation 1: Cosine similarity

To rank a document collection with regard to a query q and identify the top r matching documents:

(1) Calculate $w_{q,t}$ for each query term t in q.
(2) For each document d in the collection,
  (a) Set $S_d \leftarrow 0$.
  (b) For each query term t,
    Calculate or read $w_{d,t}$, and
    Set $S_d \leftarrow S_d + w_{q,t} \times w_{d,t}$.
  (c) Calculate or read W_d.
  (d) Set $S_d \leftarrow S_d / W_d$.
(3) Identify the r greatest S_d values and return the corresponding documents.

Figure 1: Exhaustive computation of cosine similarity between a query q and every document in a text collection. This approach is suitable only when the collection is small or is highly dynamic relative to the query rate.

However, this exhaustive approach is not efficient, because we have to process all documents even if some documents do not contain that term. Therefore, inverted files are used. An example inverted file for given sample collection is like in Figure 2.

term t	f_t	Inverted list for t
are	2	<5,1><6,1>
at	1	<6,1>
big	2	<2,1><4,1>
clean	3	<1,1><5,1><6,1>
cleaner	3	<1,1><2,1><5,1>
cleaners	1	<3,1>
cleans	2	<2,1><5,1>
house	2	<2,1><4,1>
houses	1	<5,1>

in	1	<2,1>
is	3	<2,1><4,2>
it	1	<4,1>
job	1	<1,1>
like	1	<3,1>
night	1	<6,1>
not	1	<5,1>
old	3	<2,1><3,1><4,1>
only	2	<4,1><6,1>
operations	1	<6,1>
performed	1	<6,1>
sleeping	1	<3,1>
that	2	<4,1><5,1>
the	7	<1,1><2,3><3,1><5,1><6,1>
town	1	<2,1>

d	1	2	3	4	5	6
W_d	2	3.38	2.24	2.45	2.83	2.83

Figure 2: Complete document-level inverted file for the sample database. The entry for each term t is composed of the frequency f_t and a list of pairs, each consisting of a document identifier d and a document frequency $f_{d,t}$. Also shown are the W_d values as computed for the cosine measure shown in Equation 1.

An algorithm for computing similarity using inverted files

To use an inverted index to rank a document collection with regard to a query q and identify the top r matching documents:

(1) Allocate an accumulator A_d for each document d and set $A_d \leftarrow 0$.
(2) For each query term t in q,
  (a) Calculate $w_{q,t}$, and fetch the inverted list for t.
  (b) For each pair $\langle d, f_{d,t} \rangle$ in the inverted list
    Calculate $w_{d,t}$, and
    Set $A_d \leftarrow A_d + w_{q,t} \times w_{d,t}$.
(3) Read the array of W_d values.
(4) For each $A_d > 0$, set $S_d \leftarrow A_d / W_d$.
(5) Identify the r greatest S_d values and return the corresponding documents.

Figure 3: Indexed computation of cosine similarity between a query q and a text collection.

Index Construction

1. In-Memory Inversion

To build an inverted index using the in-memory technique:

(1) Make an initial pass over the collection.
For each term t count its document frequency f_t, and determine an upper bound u_t on the length of the inverted list for t.

(2) Allocate an in-memory array of $\sum_t u_t$ bytes, and, for each term t, create a pointer c_t to the start of a corresponding block of u_t bytes.

(3) Process the collection a second time.
For each document d, and for each term t in d, append a code representing d and $f_{d,t}$ at c_t, and update c_t.

(4) Make a sequential pass over the in-memory index that has been constructed.
For each t, copy the f_t representations of the $\langle d, f_{d,t} \rangle$ pointers from the allocated u_t bytes to the inverted file, compressing if desired.

Figure 4: In-memory inversion

2. Sort-Based Inversion

3. Merge-Based Inversion

To build an inverted index using a merge-based technique:

(1) Until all documents have been processed,
  (a) Initialize an in-memory index, using a dynamic structure for the vocabularies and a static coding scheme for inverted lists; store lists either in dynamically resized arrays or in linked blocks.
  (b) Read documents and insert $\langle d, f_{d,t} \rangle$ pointers into the in-memory index, continuing until all allocated memory is consumed.
  (c) Flush this temporary index to disk, including its vocabulary.

(2) Merge the set of partial indexes to form a single index, compressing the inverted lists if required.

Figure 5: Merge-based inversion

CODING THEORIES

Efficient Index Representations

Parameterless Codes

Those are Unary, Elias' Gamma (γ), and Elias' Delta (δ)

Unary

If number is x, there will be 1 as (x – 1), then it will terminate with 0.

Example: unary code of 4 is: 1110 (There are 3 (4 – 1) 1, then 0)

Some more examples on Unary Code:

Number	Unary Code
1	0
2	10
3	110
4	1110
5	11110
6	111110
7	1111110
8	11111110
9	111111110
10	1111111110

Figure 6: Examples for Unary Coding

Elias' Gamma

If number is like $x = 2^N + t$, we will write (N + 1) as unary code, than we will state t in binary as N bits.

Example: If number is 5, $5 = 2^2 + 1$. Representation of 5 in unary code is 11110. Representation of 1 in binary is 1. So we will write, 11110:01. Here, we wrote 1 as 01, because N = 2. Therefore, we have to write 1 in 2 bytes.

If number is 1, $1 = 2^0 + 1$. (0 + 1) in unary code is 0. We have to write 1 in binary as 0 bytes. So it's 0: . We cannot write anything after :, because representation of 1 in binary with 0 bytes is nothing.

Some more examples on Elias' Gamma Code:

Number	Elias' Gamma Code

$1 = 2^0 + 1$	0:
$2 = 2^1 + 0$	10:0
$3 = 2^1 + 1$	10:1
$4 = 2^2 + 0$	110:00
$5 = 2^2 + 1$	110:01
$6 = 2^2 + 2$	110:10
$9 = 2^3 + 1$	1110:001
$15 = 2^3 + 7$	1110:111
$17 = 2^4 + 1$	11110:0001
$35 = 2^5 + 3$	111110:00011

Figure 7: Examples for Elias' Gamma Coding

Elias' Delta

If number is $x = 2^{N'} + t$, we say that $N = N' + 1$. We write N in Gamma, then write t in binary as N' bits.

Example: If number is 7, $7 = 2^2 + 3$. Here N' is 2, so $N = 2 + 1 = 3$. We write N = 3 in Elias' Gamma which is 10:1. Then we write 3 as 2 bytes, that is 11.

If number is $1 = 2^0 + 1$. Here $N' = 0$, so $N = 0 + 1 = 1$. We write N = 1 in Elias' Gamma which is 0: , then we write t = 1 as 0 bytes, that is nothing. So 1 representation in Elias' Delta is 0::

Some more examples on Elias' Delta Code:

Number	Elias' Delta Code
$1 = 2^0 + 1$	0::
$2 = 2^1 + 0$	10:0:0
$3 = 2^1 + 1$	10:0:1
$4 = 2^2 + 0$	10:1:00
$15 = 2^3 + 7$	110:00:111
$45 = 2^5 + 13$	110:10:01101
$324 = 2^8 + 68$	1110:111:01000100
$381 = 2^8 + 125$	1110:111:01111101
$24412 = 2^{14} + 8028$	1110:111:01111101011100
$66291 = 2^{16} + 755$	11110:0001:0000001011110011

Figure 8: Examples for Elias' Delta Coding

Another algorithm for coding

This algorithm gives advantages for variable-length coding. For example, numbers less than 128 are represented with one byte, whereas numbers greater than 127 are represented at least two bytes.

To encode integer $x \geq 1$:

(1) Set $x \leftarrow x - 1$.
(2) While $x \geq 128$,
  (a) *write_byte*$(128 + x \bmod 128)$.
  (b) Set $x \leftarrow (x \text{ div } 128) - 1$.
(3) *write_byte*(x).

To decode an integer x:

(1) Set $b \leftarrow$ *read_byte*(), $x \leftarrow 0$, and $p \leftarrow 1$.
(2) While $b \geq 128$,
  (a) Set $x \leftarrow x + (b - 127) \times p$ and $p \leftarrow p \times 128$.
  (b) Set $b \leftarrow$ *read_byte*().
(3) Set $x \leftarrow x + (b + 1) \times p$.

Figure 9: Encoding and decoding variable-length byte-aligned codes. Input values to the encoder must satisfy x ≥ 1.

Decoding examples for this algorithm:

1) If encoded number is 3, represented as 0000 0011,
We write,
b = 3, x = 0, p = 1
Since b ≥ 128 is false, x = x + (b + 1) * p = 0 + 4 * 1 = 4

So encoded 3 represents 4.

2) If encoded number is 138:5, represented as 1000 1010: 0000 0101
We write,
b = 138, x = 0, p = 1
since b ≥ 128,
x = x + (b – 127) * p = 0 + 11 * 1 = 11
p = p * 128 = 128
b = 5
since b ≥ 128 is false, it exits loop
x = x + (b + 1) * p = 11 + (5 + 1) * 128 = 11 + 6 * 128 = 779

Encoding examples for this algorithm:
1) If number is 4,
x = x – 1 = 3

x ≥ 128 is false,
So, we write 3

2) If number is 1045,

x = x – 1 = 1045 – 1 = 1044

x ≥ 128 is true

write 128 + 1044 mod 128 = 128 + 20 = 148

x = x / 128 – 1 = 8 – 1 = 7

7 ≥ 128 is false

write 7

So encoded form of 1045 is: 148:7

3) If number is 779,

x = x – 1 = 779 – 1 = 778

x ≥ 128 is true

write 128 + 778 mod 128 = 128 + 10 = 138

x = x / 128 – 1 = 6 – 1 = 5

5 ≥ 128 is false

write 5

So encoded form of 779 is 138:5

Golomb Rice Codes

To encode integer $x \geq 1$ using parameter b:

(1) Factor $x \geq 1$ into $q \cdot b + r + 1$ where $0 \leq r < b$.
(2) Code $q + 1$ in unary.
(3) Set $e \leftarrow \lceil \log_2 b \rceil$ and $g \leftarrow 2^e - b$.
(4) If $0 \leq r < g$ then code r in binary using $e - 1$ bits; otherwise, if $g \leq r < b$, then code $r + g$ in binary using e bits.

Fig. 10: Encoding using a Golomb code with parameter b. Input values must satisfy x ≥ 1.

Golomb and Rice Codes are not so different. Rice Code is special form of Golomb code. Golomb code with parameter b, is called as Rice Code if $b = 2^k$.

Examples:

1) If value is 3 and b = 5,

3 = 0*b + 2 + 1, where r = 2 and q = 0

Unary code of (0 + 1) = 1 is 0

e is 3, since $\log_2 5$ is approximately 2.322, $g = 2^3 - 5 = 3$

Since, $0 \leq r < g$ is true, because $0 \leq 2 < 3$, we code r = 2 in binary which is 10 using e – 1 = 2 bits.

So it's 0:101

2) If value is 15 and b = 3,

15 = 4 * 3 + 2 + 1, where r = 2 and q = 4

Unary code of (4 + 1) = 5 is 11110

e is 2, since $\log_2 3$ is approximately 1.585, $g = 2^2 - 3 = 1$

Since, $g \leq r < b$ is true, because $1 \leq 2 < 3$, we code r + g = 3 in binary which is 11 using e = 2 bits.

So it's 11110:11

3) If value is 38 and b = 8,

38 = 4 * 8 + 5 + 1, where r = 5 and q = 4

Unary code of (4 + 1) = 5 is 11110

e is 3 since $\log_2 8$ is 3. $g = 2^3 - 8 = 0$

Since, $g \leq r < b$ is true, because $0 \leq 5 < 8$, we code r + g = 5 in binary which is 101 using e = 3 bits.

So it's 11110:101

Limiting Memory Requirements

1. Accumulator Limiting

To use an inverted index and an accumulator limit L to rank a document collection with regard to a query q and identify the top r matching documents:

(1) Create an empty set A of accumulators.
(2) For each query term t in q, ordered by decreasing $w_{q,t}$,
  (a) Fetch the inverted list for t.
  (b) For each pair $\langle d, f_{d,t} \rangle$ in the inverted list
    i. If A_d does not exist and $|A| < L$, create accumulator A_d.
    ii. If A_d exists, calculate $w_{d,t}$ and set $A_d \leftarrow A_d + w_{q,t} \times w_{d,t}$.
(3) Read the array of W_d values.
(4) For each accumulator $A_d \in A$, set $S_d \leftarrow A_d / W_d$.
(5) Identify the r greatest S_d values and return the corresponding documents.

Figure 11: The limiting method for restricting the number of accumulators during ranked query evaluation. The accumulator limit L must be set in advance. The thresholding method involves a similar computation, but with a different test at step 2(b)i.

2. Accumulator Thresholding

Here, a threshold value is chosen and this value is used to create accumulators. If similarity is not less than threshold, an accumulator is created.

Reducing Retrieval Costs

1. Skipping
2. Frequency-Ordered Inverted Lists
3. Impact-Ordered Inverted Lists

To use an impact-sorted inverted index and an accumulator limit L to rank a document collection with regard to a query q and identify the top r matching documents:

(1) Create an empty set A of accumulators.
(2) Fetch the first block of each term t's inverted list. Let s_t be the stored impact score for that block.
(3) While processing time is not exhausted and while inverted list blocks remain
  (a) Identify the inverted list block of highest $w_{q,t} \times s_t$, breaking ties arbitrarily. Let C be the integer contribution derived from $w_{q,t} \times s_t$.
  (b) For each document d referenced in that block,
    i. If A_d does not exist and $|A| < L$, create an accumulator A_d.
    ii. If A_d exists, set $A_d \leftarrow A_d + C$.
  (c) Ensure that the next equi-impact block for term t is available, and update s_t.
(4) Identify the r greatest A_d values and return the corresponding documents.

Figure 12: Impact-ordered query evaluation.