CS 276 / LING 286 Information Retrieval and Web Search

Problem Set 1 Solutions

May, 2017

Problem 1 - Positional Indexes

Consider the following documents:

Doc 1: I am a student, and I currently take CS276.

Doc 2: I was a student. I have taken CS276.

1. We have seen that positional indexes are very useful to run queries and search against documents. Let's build positional indexes on top of these documents using the format DocID: <(Position$_{abc}$, Position$_{xyz}$), ...>: For example, the positional index for the word "CS276" is as follows:

   CS276: 1: <9>; 2: <8>

   Show the positional indexes for the words "I" and "student".

   Solution.

   I: 1: <1, 6>; 2: <1, 5>
   student: 1: <4>; 2: <4>

2. A phrase query "word1 word2" retrieves the occurrences where word1 is immediately followed by word2. A /k query "word1 /k word2" (k is a positive integer) retrieves the occurrences of word1 within k words of word2 on either side. For example, $k = 1$ demands that word1 be adjacent to word2, but word1 may come either before or after word2.

   For the following queries, return all the docs and corresponding positions for which the query conditions are met. If none of the documents meet the criteria, return None.

   (a) "I student"
   (b) "student I"
   (c) "I /2 student"
   (d) "student /2 I"

   Solution.

   (a) "I student":
   None

(b) "student I":
2: <(4, 5)>

(c) "I /2 student" :
1: <(4, 6)>; 2: <(4, 5)>

(d) "student /2 I" Same as c

3. Let's say we want to find documents in which "I" and "student" are at most 2 words apart, but in the same sentence. This condition only applies to document 1.

   How would you modify the positional index to support queries that demand the terms to be in the same sentence? You can assume that the parsing step is able to identify the sentences in a document. Write down an example of the modified postings list for the words "I" and "student".

   Solution. For each term in the document, create a tuple containing both the position and the sentence ID. The postings list will look like:

   Word: DocId: <(Position, SentenceId), (Position, SentenceId), ...>; DocId: <...>; ...

   The ordering is preserved since $\text{SentenceId}_1 < \text{SentenceId}_2$ implies $\text{Position}_1 < \text{Position}_2$, so the streaming algorithms still work.

   I: 1: <(1,1), (6,1)>; 2: <(1,1), (5,2)>
   student: 1: <(4,1)>; 2: <(4,1)>

Problem 2 - Index Compression

1. Suppose the vocabulary for your inverted index consists of the following 6 terms:

   elite
   elope
   ellipse
   eloquent
   eligible
   elongate

   Assume that the dictionary data structure used for this index stores the sorted list of terms using dictionary-as-a-string storage with front coding and a block size of 3. Show the resulting storage of the above vocabulary of 6 terms. Use the special symbols * and ♢ as used in the discussion on front coding in Chapter 5.

   Solution.

   8el*igible3♢ite5♢lipse
   8elo*ngate2♢pe5♢quent

2. Consider a postings list:

<4, 15, 62, 63, 265, 268, 270, 500>

with the corresponding list of document gaps:

<4, 11, 47, 1, 202, 3, 2, 230>

Assume that the length of the postings list is stored separately, so the system knows when a postings list is complete. We are going to contrast the size and speed tradeoffs of using variable byte encoding with gamma encoding.

(a) Encode the document gaps using variable byte encoding.

Solution.

```
1000 0100   1000 1011   1010 1111   1000 0001
0000 0001   1100 1010   1000 0011   1000 0010
0000 0001   1110 0110
```

(b) Encode the document gaps using gamma encoding.

Solution.

```
110 00   1110 011   111110 01111   0
11111110 1001010   10 1   10 0
11111110 1100110
```

(c) Assuming a slow ethernet connection (1 million bytes / second) to network attached storage, how long would it take to transfer just the given postings list stored in variable byte encoding (of document gaps) to memory? How about the gamma encoded list? (Answer in μs = 10^{-6} seconds.) Note: Do not pad the transmitted pad to the byte boundary.

Solution. VBC: 10 μs; gamma: 7.5 μs (8 μs if you rounded up to the nearest number of bytes)

(d) Now, assume that to decode a gamma encoded postings list into usable numbers takes a constant time of $n/2$ μs where n is the number of elements in the postings list. This adds a processing time to storing a postings list in gamma encoded format. For the given postings list, what is the upper limit on how much processing time (in μs per list element) variable byte encoding can take, in order to be faster than (or equally fast as) gamma encoding?

Solution. Processing a gamma encoded list now takes 11.5 μs. VBC would have to take 0.1875 μs per list element to beat gamma encoding. (If you rounded up to bytes, the answer is 0.25 μs.)

Problem 3 - Tolerant Retrieval

Consider the following text:

peter piper picks papers

1. How many character bigram dictionary entries and character bigram posting entries are generated by indexing the bigrams in the terms in the text above?

Solution. The frequencies of character bigrams are

4: $p
3: er, pe
2: pi, r$, s$
1: ap, ck, et, ic, ip, ks, pa, rs, te

So the number of bigram dictionary entries is $1 + 2 + 3 + 9 = 15$, and the number of character bigram posting entries is $1 \times 4 + 2 \times 3 + 3 \times 2 + 9 \times 1 = 25$.

2. How would the wild-card query **p*er** be best expressed as an AND query using the bigram index over the text above?

   Solution. ((r$ AND er) AND $p) [(($p AND r$) AND er) also works]

3. For the most efficient query in part 2, how many postings must be traversed? Note: the merge algorithm can only take 2 lists at a time and has to traverse all postings in both lists.

   Solution. 11 (2 + 3 for the first AND, 2 + 4 for the second AND). (The answer can be $8 = (2 + 2) + (2 + 2)$ if the merges terminate early once one of the postings lists ends.)

4. How many permuterm dictionary entries are generated by indexing the words in the text above?

   Solution. 25. (peter: 6, piper: 6, picks: 6, papers: 7)

5. How would the wild-card query **p*er** be expressed for lookup in the permuterm index?

   Solution. er$p*

Problem 4 - Vector Space Model

Consider the following documents:

A: to be not to be not
B: it was not to be

1. Write down the term frequency of each term in each document.

   Solution.

   A: {to: 2, be: 2, not: 2}
   B: {it: 1, was: 1, not: 1, to: 1, be: 1}

2. Write down the number of occurrences of each word bigram in each document (ignoring the begin-of-document and end-of-document tokens).

   Solution.

   A: {to be: 2, be not: 2, not to: 1}
   B: {it was: 1, was not: 1, not to: 1, to be: 1}

3. Consider a vector space where each dimension is a word *bigram*. For each document, write down the *normalized* vector of raw bigram counts (no logarithm or idf) using the format {bigram: value, bigram: value, ...}. Use Euclidean normalization.

   Solution.

   A: {to be: 2/3, be not: 2/3, not to: 1/3}
   B: {it was: 1/2, was not: 1/2, not to: 1/2, to be: 1/2}

4. Now consider the tf-idf weighting:

$$w_{t,d} = \mathrm{tf}_{t,d} \times \log_{10}(N/\mathrm{df}_t)$$

   For each document, write down the *normalized* vector of tf-idf weights where each dimension is a word *bigram*.

   Solution.

   A: {to be: 0, be not: 1, not to: 0}
   B: {it was: $1/\sqrt{2}$, was not: $1/\sqrt{2}$, not to: 0, to be: 0}

5. For each of the weighting schemes in parts 3 and 4, compute the cosine similarity between documents A and B.

   Solution. For part 3, the similarity is $\frac{2}{3} \cdot \frac{1}{2} + \frac{1}{3} \cdot \frac{1}{2} = \frac{1}{2}$. For part 4, the similarity is 0. This might seem odd considering that the two documents are pretty similar. The reason is that idf downweights items (bigrams in our case) that are common in many documents. Since we have only two documents, the similarities between the documents are completely washed out by the idf terms. The result will be more meaningful if we compute idf on a larger corpus.