CS 276 / LING 286 Information Retrieval and Web Search

Problem Set 1

Due: Tuesday, April 30, 2019 at 16:00 PDT

General Instructions

This problem set has 5 problems. For some questions, you only need to write your answer as a number or a few characters; but you need to write a brief justification for your answer if the question requests you to do so. Please make sure that your answer is **clear and succinct**; we may not give full credits to answers that are overly long. After you finish this problem set, upload the PDF file with your answers to *GradeScope*.

It is your responsibility to ensure that you finish this problem set **individually**. You can have general knowledge-level discussions with other students, but sharing answers is not allowed under the Honor Code. If you have questions or need clarifications, please post on Piazza or come to office hours.

Problem 1 - Positional Indexes (5 points)

Consider the following documents:

Doc 1: I am a student, and I currently take CS276. I was a student in CS224N last quarter.

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

a. 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_a: $<\text{Position}_\text{abc}$, $\text{Position}_\text{xyz}>$; DocID_b: $<\text{Position}_\text{def}$, $\text{Position}_\text{ghi}>$; ...
The position of the first word in the document is 1, by convention.

   For example, the positional index for the word "am" is as follows:

   am: 1: $<2>$

   Show the positional indexes for the words "I" and "CS276".

b. 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. To take a hypothetical example: if a two-word query matches words 3 and 4, as well as words 7 and 8 in document 2, you should return the following:
   2: $<(3,4), (7,8)>$

   If none of the documents meet the criteria, return None.

i. “I student”
ii. “student I”
iii. “I /2 student”
iv. “student /2 I”

c. Let’s say we want to find documents in which "I" and "CS276" are at most 3 words apart, but in the same sentence.

   i. 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.
   ii. Write down an example of the modified postings list for the words "I" and "CS276".

Problem 2 - Dictionary Storage Compression (10 points)

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

elite
elope
ellipse
eloquent
eligible
elongate

a. 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. Show the resulting storage of the above vocabulary of 6 terms under two different block sizes:

   i. block size of 3
   ii. block size of 6

   Use the special symbols * and ◇ as used in the discussion on front coding in Chapter 5.

b. Calculate the total dictionary storage (in bytes) required to store the vocabulary using:

   i. Fixed width entries of 20 bytes
   ii. Blocked storage with front coding and a block size of 3 (i.e., Problem 2 (a) (i) above)

   By what proportion do we reduce storage by using blocked storage with front coding and a block size of 3?

   Assume that:
   - The dictionary stores the term, the document frequency, and a pointer to the postings list
   - The postings list pointer and the document frequency each occupy 4 bytes
   - Each character can be stored in 1 byte
   - For blocked storage, the term pointer occupies 3 bytes

c. Suppose the above vocabulary was only a snippet of a larger vocabulary. What is the maximum size of the vocabulary that can be resolved using 3 byte term pointers? Assume that each vocabulary entry is 8 characters long on average, and that blocked storage with front coding compression method compressing the dictionary string by 20%. Show your calculations.

Problem 3 - Postings Compression (8 points)

Consider a postings list:

$$<4, 15, 62, 63, 265, 267, 270, 501>$$

with the corresponding list of document gaps:

$$<4, 11, 47, 1, 202, 2, 3, 231>$$

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.

b. Encode the document gaps using gamma encoding.

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 $\mu s = 10^{-6}$ seconds.) Note: Do not pad the transmitted pad to the byte boundary.

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 an additional processing time to decode the posting list stored in gamma encoded format to documents gap integers. 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?

Problem 4 - Tolerant Retrieval (7 points)

Consider the following text:

peter piper picks papers

a. How many character bigram dictionary entries and character bigram posting entries are generated by indexing the bigrams in the terms in the text above? Use the special character $ to denote the beginning and end of terms.

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

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

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

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

Problem 5 - Vector Space Model (10 points)

Consider the following documents:

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

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

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

c. Consider a vector space where each dimension is a word *bigram*. For each document, write down the vector of *normalized* bigram term frequencies (without log scaling) using the format {bigram: value, bigram: value, ...}. Use Euclidean normalization.

d. 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 vector of *normalized* tf-idf weights where each dimension is a word *bigram*. Perform Euclidean normalization after calculating the tf-idf weight vector $w_{t,d}$. As in c, use the format {bigram: value, bigram: value, ...}. Assume that the corpus only has the two documents listed above.

e. For each of the weighting schemes in parts c and d, compute the cosine similarity between documents A and B. Briefly explain any insights you draw from your answers.

f. Let's say you now have the opportunity to add one document to the corpus. Design a document J' to add to the corpus such that the cosine similarity of A and B under part d remains unchanged.