Problem Set 2

Available: Apr.16, 2015
Due: 23:59PM PDT, Apr.23, 2015

General Instructions

This problem set consists of 3 problems. For some of the 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 over-long. After you finish this problem set, you need to upload an electronic document with your answers to GradeScope.

Note that it is your responsibility to make sure 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 - Tolerant Retrieval (15 points)

In this problem, we will explore tolerant retrieval techniques for handling wild-card queries and spelling corrections, by going through a (simple) example.

For Question 1 and 2 in this problem, please consider the following text:

humpty dumpty billy sat in the hall

1. Bigram Indexes (9 points)
   a. How many character bigram dictionary entries are generated by indexing the bigrams in the terms in the text above as discussed in Section 3.2.2 in the book? Your answer should include (1) the total number of bigram dictionary entries, (2) a list of all the dictionary entries.
   b. How would the wild-card query hu*ty be best expressed as an AND query using the bigram index you have constructed? Think about the most efficient query in terms of the number of posting entries traversed. For simplicity, answer by giving a list of bigrams ordered based on their postings list sizes (one line, space separated). For example, you should write ab cd ef to refer to the Boolean query ((ab AND cd) AND ef).
   c. How many posting entries are traversed for the most efficient query in 1.b?

2. Permuterm Indexes (6 points)
   a. Now you turn to use permuterm index to handle the wild-card query. How many permuterm dictionary entries are generated by indexing the same text given above? Your answer should include (1) the total number of permuterm dictionary entries, (2) a list of all the dictionary entries.
b. Again, how would the wild-card query `hu*ty` be expressed for lookup in the permuterm index you constructed? Your answer should be in the similar form as Question 1.b.

Problem 2 - Index Compression (17 points)

For this problem, assume that you are compressing a plain document-level (non-positional) postings list, encoded using gap encoding.

1. (5 points) Dictionary Compression: In the “dictionary-as-a-string” approach for compressing a term list, assume that we use 3 bytes per term pointer to refer to the position of a word in the long string representing our dictionary. Now, using a block size of $k$ terms in which 1 byte is needed to store each term length, how many bytes do we save per block (See IIR book 5.2.2)?

2. (6 points) Posting Compression:
   a. Write down the Gamma code of the integer 1891.
   b. Gamma codes are relatively inefficient for large numbers as they encode the length of the offset in inefficient Unary code. Delta codes differ from Gamma codes in that they encode the first part of the code (length) in Gamma code instead of Unary code. The encoding of the second part of the code (offset) is the same. For example, the Delta code of 7 is 10,0,11 (the commas are added for readability) in which 10,0 is the Gamma code for length of 2 and the encoding of offset (11) is unchanged. Now write down the Delta code for the number 1891.
   c. For the sequence of Gamma codes gaps `11100011101010101111101101111011`, please reconstruct the posting list sequence. Assume the first gap encodes offset from 0th document. You should write your posting list as space separated numbers, e.g. `1 2 15 30 121 343`.

3. (6 points) Comparison of Different Encoding Techniques
   a. Suppose a word (maybe `the`) occurs in every document. In terms of compression (space needed for storage), how many times more efficient is use of a gamma code versus variable-byte encoding? Briefly explain your answer.
   b. What range of values can be encoded in 1 byte in a variable-byte code, but consume more than 1 byte to store in a gamma code? Briefly explain your answer.
   c. Conversely, for what range(s) of gap values is a gamma code shorter than a variable-byte code? Briefly explain your answer.

Problem 3 - TF-IDF and Vector Space Model (18 points)

In this problem, we will explore the use of different scoring and weighting schemes with vector space model on information retrieval tasks.

For this problem, consider a simple collection with the following two documents:
Consider the query:

**Query:** school closed rain

Also, consider the following **stop word** list for Question 1 and 2 only:

[the, to, is, and, in, has, not]

1. (3 points) What are the similarity scores of the query with each document given above using Jaccard coefficient if there are no stop words? What are the similarity scores if we use the stop words?

2. (5 points) The query is converted to a unit vector using tf-idf weighting

\[ w_{t,d} = tf_{t,d} \times \log_{10}(N/d_f) \]

and Euclidean normalization. The documents are converted to unit vectors using raw tf weighting and Euclidean normalization. Now 1) compute and report the cosine similarity of the query with each document if there are no stop words; 2) Report the result when we use the stop words shown above. For this question, briefly show how you arrive at the answer. (Hint: Think about whether you can calculate the answer without computing all the tf-idf values.)

3. (5 points) We will now compute the similarity scores for each document with the query using a type of overlap score measure defined as follows: Each document is represented as a term frequency (tf) vector which is normalized using the maximum tf formula:

\[ 0.25 + [0.75 \times \frac{tf_{t,d}}{\max_t(tf_{t,d})}] \]

The score of a document with respect to a query is then computed as: the sum of the normalized tf values of the query terms in the document vector. Now compute and report the similarity of the query with each document using the stop word list:

[the, to, is, and, in, not]

4. (5 points) Remember that given a query \( q \) and documents \( d_1, d_2, \ldots \), in addition to ranking the documents in order of decreasing cosine similarities, we may also rank the documents \( d_i \) in order of increasing Euclidean distance from \( q \). Euclidean distance between two vectors \( \vec{x} \) and \( \vec{y} \) can be calculated as:

\[ |\vec{x} - \vec{y}| = \sqrt{\sum_i (x_i - y_i)^2} \]

Show that if \( q \) and the \( d_i \) are all normalized to unit vectors, then the rank ordering produced by Euclidean distance is identical to that produced by cosine similarities. (Hint: Write out \( |\vec{q} - \vec{d}|^2 \).)