605.744: Information Retrieval
Final Exam

Sabbir Ahmed

December 4, 2022

1. (a) Suppose the document collection is very large, and that the index will not fit in the available RAM. Describe an indexing algorithm that works when memory is small compared to the size of the index.

   Answer: One of the methods that were used in my implementation of the document ranking assignment to reduce memory overhead was to split up the pipeline into chunks so that the RAM would not get exhausted. The words needed to be extracted and sorted lexicographically, which can easily explode in memory requirements. One method employed was to gather as much of the term frequency of each of the terms in the documents and then write them out to disk. Once all of the term-docID-term frequency records were extracted across several files, they can be merged in a separate execution when the RAM usage is low. This final sorted file of the term-docID-term frequency records can later be used to traverse through and build the index file.

   (b) Once an inverted file has been created, it is possible to calculate document vector lengths for a TF/IDF cosine model. This pre-calculation makes query-time performance much more efficient. Explain how right after creating the inverted file document vector lengths can be efficiency computed for all docids in parallel using one traversal (i.e., one single pass) over the inverted file.

   Answer: Pre-calculating the document vector lengths was also utilized in my implementation of the document ranking assignment. Computing the cosine similarity of documents against a query requires computing dot products with sums of squares of the TF-IDF values of each terms in a dictionary, which can be very computationally expensive. However, instead of computing the sums of squares of all the terms every time a query is inputted, they can be pre-calculated independently. Once an inverted file is created, the system can loop through each of the words in the dictionary and retrieve its information from the index file, which contains the corresponding term frequencies required to compute the square of the TF-IDF values. A separate hash table that maps the docIDs to their partial length is created. The squared TF-IDF values of each of the terms in a document get added to the corresponding docID in the hash table until the end of the dictionary is reached. This hash table can finally be written to disk and later retrieved to compute the dot products against the queries.

2. For this problem consider the following collection of 8 documents. No other documents are present in the collection besides these eight. When analyzing these documents and the query below you are to ignore all punctuation and any word with four or fewer letters. All short

words with four or fewer letters are considered stopwords that are completely ignored for this problem. Use base 2 logs if any logarithm is required.

Answer:

The documents were normalized to the following:

D1: france sends ukraine

D2: ukraine economy

D3: france germany ukraine

D4: economy tanks inflation

D5: france economy

D6: france raises cheese prices

D7: economy inflation makes prices

D8: germany sends tanks ukraine

Query Q for both parts (a) and (b): "france ukraine".

(a) Compute cosine values and rank documents D1 and D2 using query Q using the vector cosine model with TF/IDF term weighting. The query Q consists of the words: "france aids ukraine". Show your work. Report scores to three decimal places (e.g., 0.123)

Answer: Compiling the term frequencies of all the relevant vocabulary: The TF and

Table 1: Term frequencies of *france sends ukraine economy*

Word	**D1**	**D2**	Q
france	1	0	1
sends	1	0	0
ukraine	1	1	1
economy	0	1	0

IDF of the vocabulary are computed using the following values:

$$
\begin{aligned}
TF(t) &= \text{term frequency in the corpus} \\
IDF(t) &= \text{inverse document frequency} \\
&= log_2\left(\frac{N}{df(t)}\right), \\
&\text{where:} \\
&N = \text{length of corpus} = 8, \\
&df(t) = \text{document frequency}
\end{aligned}
$$

Finally, computing the cosine similarities.

Table 2: TF-IDF of *france sends ukraine economy*

term	**DF**	**IDF**
france	4	$log_2\left(\frac{8}{4}\right) = 1$
sends	2	$log_2\left(\frac{8}{2}\right) = 2$
ukraine	4	$log_2\left(\frac{8}{4}\right) = 1$
economy	4	$log_2\left(\frac{8}{4}\right) = 1$

Table 3: Squares of TF-IDF

Word	**D1**	**D2**	Q
france	1	0	1
sends	4	0	0
ukraine	1	1	1
economy	0	1	0

Table 4: Similarities

value	**D1**	**D2**	**Q**
sum of squares	6	2	2
length, $\|\|D\|\| = \sqrt{\sum_{i=1}^{t} w_i^2}$	2.4494	1.4142	1.4142
dot product	2	1	2
cosine similarity, $\frac{d \cdot q}{\|\|d\|\| * \|\|q\|\|}$	$\frac{2}{2.4494 * 1.4142} = 0.577$	$\frac{1}{1.4142 * 1.4142} = 0.500$	1

(b) Now rank the same documents (D1 and D2) but this time by probability of relevance to the query using a unigram statistical language model. For smoothing purposes you should use a mixture model with a parameter $\lambda = 0.2$. You should assume that the prior probability of relevance is the same for each document. Report scores using three digits of precision (e.g., 1.23×10^{-4})

Answer: Computing the $tf_{t,d}$ (the (raw) term frequency of term t in document d) and L_d (the number of tokens in document d):

Table 5

	tf_{D1}	L_{D1}	tf_{D2}	L_{D2}	cf_t	cs
france	1	3	0	2	4	25
ukraine	1	3	1	2	4	25

Using $\lambda = 0.2$ and Q=*france ukraine*, and expanding the probability:

$$\begin{aligned} P(d|q) &\propto P(d) \prod_{t \in q} ((1-\lambda)P(t|M_c) + \lambda P(t|M_d)) \\ &\propto \prod_{t \in q} \left((1-\lambda)\frac{cf_t}{cs} + \lambda \frac{tf_{t,d}}{L_d} \right) \end{aligned}$$

Computing the probability for each documents:

$$\begin{aligned} P(Q|D1) &= \prod \left((1-\lambda)\frac{cf_t}{cs} + \lambda \frac{tf_{t,D1}}{L_{D1}} \right) \\ &= \prod \left(0.8 \times \frac{cf_t}{25} + 0.2 \times \frac{tf_{t,D1}}{3} \right) \\ &= \left(0.8 \times \frac{cf_{france}}{25} + 0.2 \times \frac{tf_{france,D1}}{3} \right) \times \left(0.8 \times \frac{cf_{ukraine}}{25} + 0.2 \times \frac{tf_{ukraine,D1}}{3} \right) \\ &= \left(0.8 \times \frac{4}{25} + 0.2 \times \frac{1}{3} \right) \times \left(0.8 \times \frac{4}{25} + 0.2 \times \frac{1}{3} \right) \\ &= (0.194)(0.194) \\ &= 3.79 \times 10^{-2} \end{aligned}$$

$$\begin{aligned} P(Q|D2) &= \prod \left((1-\lambda)\frac{cf_t}{cs} + \lambda \frac{tf_{t,D2}}{L_{D2}} \right) \\ &= \prod \left(0.8 \times \frac{cf_t}{25} + 0.2 \times \frac{tf_{t,D2}}{3} \right) \\ &= \left(0.8 \times \frac{cf_{france}}{25} + 0.2 \times \frac{tf_{france,D2}}{3} \right) \times \left(0.8 \times \frac{cf_{ukraine}}{25} + 0.2 \times \frac{tf_{ukraine,D2}}{3} \right) \\ &= \left(0.8 \times \frac{4}{25} + 0.2 \times \frac{0}{2} \right) \times \left(0.8 \times \frac{4}{25} + 0.2 \times \frac{1}{2} \right) \\ &= (0.128)(0.228) \\ &= 2.92 \times 10^{-2} \end{aligned}$$

3. Two IR systems (called A and B below) return top ranked lists of 20 documents for a query in an IR test collection. The query is known to have 10 relevant documents in the test collection.

The test collection contains exactly 1,000 documents. System A finds relevant docs at ranks: 2, 4, 5, 12, 15, 20. The other top 20 documents are not relevant. System B finds relevant docs at ranks: 1, 2, 8, 10, 15, 18, 20. The other top 20 documents are not relevant.

(a) Which system has higher Recall at rank 20?

Answer: System B has the higher recall at rank 20 with 7 retrieved relevant documents.

(b) What is Precision at 10 documents (P@10) for both systems?

Answer: For System A, the relevant documents at rank 10 retrieved are {2,3,5}. Therefore, the P@10 is 3/10=0.3.

For System B, the relevant documents at rank 10 retrieved are {1,2,8,10}. Therefore, the P@10 is 4/10=0.4.

(c) What is the average precision (AP) of System A?

Answer: $(1/2 + 2/4 + 3/5 + 4/12 + 5/15 + 6/20)/10 = 0.257$

(d) In general, what is the difference between mean average precision and average precision?

Answer: The average precision computes the averages of the precision of a retrieval system over a range of recall values. The mean average precision refers to averaging the average precision values over multiple queries.

(e) Suppose System A is used to create a ranking of all 1,000 documents and the first 20 documents retrieved as the same as those listed above. What is the highest possible average precision score for System A computed over the entire ranking?

Answer: The precision may increase after the first 20 documents with every one of its later retrievals being relevant.

4. Short answer questions / calculations about index compression. Show your work and justify any responses. Note in working on this problem you should use the method for calculating gamma/delta codes and variable byte coding presented in the lecture (or textbook) and not any other variation.

(a) Represent the integer 79 using Gamma coding.

Answer: Converting to gamma coding:

$$\begin{aligned} unary(floor(log_2(79))) &= unary(6) \\ &= 1111110 \\ binary(x - 2^{(floor(log_2(x)))}) &= binary(79 - 2^6) \\ &= binary(79 - 64) \\ &= 001111 \\ gamma(79) &= 1\ 1111\ 1000\ 1111 \end{aligned}$$

(b) Why do we encode docid gaps instead of raw docids in postings lists?

Answer: DocID gaps take up a significantly less storage than entire DocIDs. For example, encoding the DocIDs {1,7,1000,1001} can be compressed with their DocID gaps as {1,6,994,1}.

(c) The following Gamma-encoded bit string represents a gap list of 4 docids. What are the docids?

11111010101110001001110010

Answer: Regrouping the bit sequence using the following algorithm:

- Read the 1 bits until the first zero and store it as *magnitude* of the unary code
- Read the *magnitude* number of bits as $b =$ binary code
- Compute $u = 2^{magnitude}$
- Add u to the decimal value of the binary code, $u + dec(b)$

Therefore,

$$\begin{aligned}&\{11111010101\ 11000\ 100\ 1110010\}\\ &= \{2^5 + dec(10101),\ 2^2 + dec(00),\ 2^1 + dec(0),\ 2^3 + dec(010)\}\\ &= \{2^5 + 21,\ 2^2 + 0,\ 2^1 + 0,\ 2^3 + 2\}\\ &= \{53, 4, 2, 10\}\end{aligned}$$

The docIDs are:

$$\begin{aligned}&= \{53, 53 + 4, 53 + 4 + 2, 53 + 4 + 2 + 10\}\\ &= \{53, 57, 59, 69\}\end{aligned}$$

(d) Term “JHU” occurs in four documents with docids: 64, 100, 2000, and 2032. Calculate a compressed representation of this four docid posting list using variable byte coding. Hint: you should encode gaps, not raw docids.

Answer: The gap list is {64,36,1900,32}.

For (e-g) indicate True or False – and Justify Your Response.

(e) Claim: Gamma codes are always an odd number of bits in length.

Answer: True, since gamma codes are concatenations of the unary and binary representations of a number, the binary code will be even if the unary code is odd and vice versa.

(f) Claim: A Gamma code always starts with a one bit

Answer: False, since the unary code of 1 would be 0.

(g) Claim: The Gamma code for the integer $2^{51} - 1000$ (read: two raised to the fifty-first power minus one thousand) is shorter than 85 bits.

Answer: False, since the unary code of the integer would be 51 bits in length, the binary portion will be close to that length and add up to more than 85 bits.

5. The three major problems in text retrieval are: (a) polysemy; (b) synonymy; and, (c) morphology. Briefly explain each issue and how it can lower performance. Give an example of each phenomena.

Answer:

(a) Polysemy refers to words that can have multiple meanings depending on the context. For example, *space* can refer to its noun version of unoccupied area. The unoccupied area can be physical or abstract, i.e. “the space between planets” or “a teenager needing their own personal space”. The word can also be used as a verb to refer to a person physically or emotionally distancing themselves from a situation, i.e. “spacing out during lectures”. Polysemy introduces ambiguity to a retrieval if a query is not given enough context and the system retrieves the undesired version of the word.

(b) Synonymy refers to different words addressing the same meaning. For example, *colossal*, *giant* and *huge* all describe the size of an object to be very big. Numerous other words also act as synonyms for *big*. Synonymy can lower performance of a retrieval system if it is not aware of the numerous synonyms a query word may have. If a user queries for “big company” but the system only contains documents with the numerous synonyms of *big*, the ranked documents may not be what the user implied.

(c) Morphology refers to the various conjugations of a word. In English, adding suffixes such as “s” or “es” transforms a noun into its plural form. Adding suffixes such as “d”, “ed”, and “ing” transforms a present tense verb to a different tense. Morphology is not only limited to suffixes or prefixes, since there are special cases of words needing a replacement in a character, i.e. *sang* is the past tense form of *swim*, while *sung* is its past participle form. Morphology can introduce issues in a retrieval system if the different variations of the words in a query are not accounted for. These systems often employ some levels of stemming in their dictionary and the query processing to normalize the words to their base forms. However, stemming can introduce additional ambiguity when different words get stemmed to a common word. i.e. “transparent” and “transparency” get stemmed to “transpar” using a Porter stemmer.

6. Short answers about text classification.

   (a) What is negative evidence in Binomial (also called Bernoulli) Naïve Bayes text classification?

   Answer: That the word does not occur in the classes.

   (b) For a class that attains precision of 0.5 and recall of 0.6, what is the corresponding F1 score?

   Answer: The F1 score is computed as $2 \times \frac{P \times R}{P+R}$. Therefore,

$$\begin{aligned} F1 &= 2 \times \frac{P \times R}{P + R} \\ &= 2 \times \frac{0.5 \times 0.6}{0.5 + 0.6} \\ &= 0.54 \end{aligned}$$

   (c) For the three classes below (business, local, and sports) with the indicated system predictions, calculate precision for the 12 news articles in two ways: using micro averaging and macro averaging.

   Answer: Precision is the ratio of the true positives over the total number of classes labeled positive (both true and false positive classes). In the table, there are a total of

8 true positives, with 2, 3, and 3 true positives and 2, 0, and 2 false positives in the 3 respective classes.

The micro average, P_μ, can be computed as:

$$\begin{aligned}
P_\mu &= \frac{\sum_{i=0}^{n}(TP_i)}{\sum_{i=0}^{n}(TP_i + FP_i)} \\
&= \frac{2+3+3}{(2+2)+(3+0)+(3+2)} \\
&= \frac{8}{12} \\
&= 0.67
\end{aligned}$$

The macro average, P_M, is computed by taking the expected value of the individual precision scores of the 3 classes.

$$\begin{aligned}
P_b &= \frac{TP_i}{TP_i + FP_i} \\
&= \frac{2}{4} \\
&= 0.5 \\
P_l &= \frac{TP_i}{TP_i + FP_i} \\
&= \frac{3}{3} \\
&= 1 \\
P_s &= \frac{TP_i}{TP_i + FP_i} \\
&= \frac{3}{5} \\
&= 0.6 \\
P_M &= \frac{P_b + P_l + P_s}{3} \\
&= \frac{0.5 + 1 + 0.6}{3} \\
&= 0.70
\end{aligned}$$

7. Give brief responses to the following:

   (a) Give one example when converting all upper-case letters to lower-case could cause a retrieval error.

   Answer: Acronyms are almost always consisting of all capital letters. They are used as proper nouns referring to organizations or businesses. If

   (b) True or False: stemming can improve recall at rank 100 in a TF/IDF vector cosine system?

   Answer: True, because the less frequent words would become stemmed to more common words.

(c) What is a permuterm index useful for?

Answer: Permuterms are used in wildcard queries to indicate the end of a term. This is needed because when creating rotation permutations of the vocabulary, the original word may become lost. The permuterm index can be used to rotate the word back to its original form.

(d) What modification to an inverted file data structure makes it possible to intersect two postings lists for a Boolean AND query in less than linear time in the sum of the lengths of the two postings lists?

Answer: By adding the file offsets to the index file.

(e) What is front-coding and what is it used for in information retrieval?

Answer: Front-coding is a method of text compression that utilizes prefixes from the previous word in the dictionary to avoid duplicating portions of the words. For example, if a dictionary contains the words *trova*, *trovano* and *trovare*, then the compression string would use *trova* as the base prefix, and append the remnants of the individual strings to represent them, i.e. 5trova*no2◊re instead of trova/trovano/trovare. This method works best if the dictionary is lexicographically sorted.

(f) In a large collection of English documents, which word would you expect to have a lower inverse document frequency (IDF), apple or volcanic? Why?

Answer: The more frequent a word appears across documents, the lower its IDF value tends to be. I would expect *apple* to appear a lot more than *volcanic* in documents in a general corpus. However, if the documents are in a corpus related to topics on geology or seismic activities, I'd expect *volcanic* to appear more frequently across the documents and thus having a lower IDF value.

(g) What are the two main principles behind cover density ranking?

Answer: The covers for the query contain the shortest interval between the terms and the documents have to be concatenated together.

(h) Explain Broder's taxonomy of information needs.

Answer: Broder's taxonomy refers to the classification of search queries by users into 3 categories: informational, navigational, and transactional. The system can provide higher quality documents if it can classify the users' query.

(i) How are estimates of p(word|class) calculated in Multinomial Naïve Bayes text classification?

Answer: With the product of the number of occurrences of the word in each of the classes.

(j) What is the kernel trick?

Answer: The kernel trick refers to converting data points into vectors.