CS 276 / LING 286 Information Retrieval and Web Search

Problem Set 2 Solutions

Apr. 27, 2015

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.

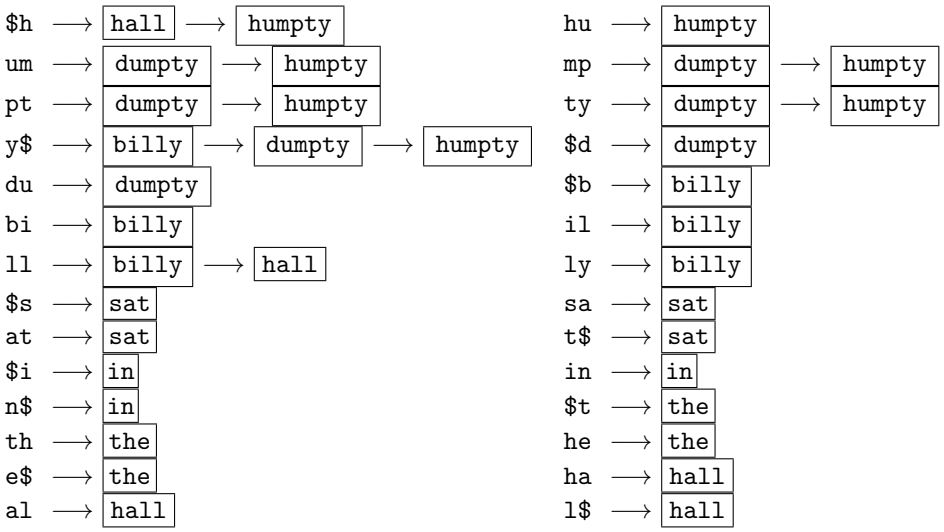
For Question 1 and 2 in this problem, please consider the following text:

`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.

Solution: There are **28** dictionary entries in total (shown lexicographically sorted):



---

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)*.

Solution: `hu $h ty y$`.
We get this by ordering by the posting list sizes `hu`:1, `$h`:2, `ty`:2, `y$`:3

---

c. How many posting entries are traversed for the most efficient query in 1.b?

Solution: **10** posting entries are traversed.
Assuming the query is `(((hu AND $h) AND ty) AND y$)`, we have the following:

	count	traversal	result
first `AND`	3	`humpty`, `hall` and `humpty`	`humpty`
second `AND`	3	`humpty` and `dumpty`, `humpty`	`humpty`
third `AND`	4	`humpty` and `billy`, `dumpty`, `humpty`	`humpty`

Summing up the counts, we get a total of 10 posting entries traversed.

---

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.

Solution: There are 36 terms in total:

`humpty$`	`dumpty$`	`billy$`	`sat$`	`in$`	`the$`	`hall$`
`$humpty`	`$dumpty`	`$billy`	`$sat`	`$in`	`$the`	`$hall`
`y$humpt`	`y$dumpt`	`y$bill`	`t$sa`	`n$i`	`e$th`	`l$hal`
`yt$hump`	`yt$dump`	`ly$bil`	`at$s`		`he$t`	`ll$ha`
`pty$hum`	`pty$dum`	`lly$bi`				`all$h`
`mpty$hu`	`mpty$du`	`illy$b`				
`umpty$h`	`umpty$d`					

---

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.

Solution: We rotate the query **hu*ty** until the wildcard is at the end to get: **ty$hu***.

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)?

Solution: Saving per block of $2k - 3$ bytes.
Initially, k terms require $3k$ bytes. After blocking, they require $3 + k$ bytes (3 bytes for the initial block pointers, and $1 \times k$ bytes for the length of each term). So the savings per block is $(3k) - (3 + k) = 2k - 3$ bytes.

---

2. (6 points) Posting Compression:
a. Write down the Gamma code of the integer 1891.

Solution: The Gamma code of 1891 is **111111111101101100011**
We first express 1891 in binary: 11101100011. Dropping the first 1, we have an *offset* of 1101100011 and a *length* of 10. Expressing the length in unary, we get 1111111111 + 0 + 1101100011 → 111111111101101100011

---

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.

Solution: The delta code of 1891 is **11100101101100011**.
From part a), we have 1891 in binary as 11101100011, with an *offset* of 1101100011 and a *length* of 10. We now encode the length 10 using Gamma encoding to get 1110010. Putting the length and the offset together, we have 1110010 + 1101100011 → 11100101101100011.

For the Gamma of 10: 10 has binary form 1010, which gives offset 010 and length 3. So the Gamma of 10 is 1110010.

---

c. For the sequence of Gamma codes gaps **1110001110101010111101101111011**, please reconstruct the posting list sequence. Assume the first gap encodes the offset from 0th document. You should write your posting list as space separated numbers, e.g. 1 2 15 30 121 343.

Solution: Postings List: 9 15 18 19 78 85
Breaking apart 1110001110101010111101101111011, we get:

Gamma	1110001	11010	101	0	11111011011	11011
Binary	1001	110	11	1	111011	111
Gaps	9	6	3	1	59	7
Postings List	9	15	18	19	78	85

---

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.

Solution: Gamma codes are **8 times** more efficient than VB codes.
For words that appear in every document, the gap to be encoded is 1, which has a Gamma code of 0 (taking up 1 bit) versus requiring 1 byte (8 bits) using variable-byte encoding.

---

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.

Solution: The range **16 - 127** can be encoded in one byte for VB encoding, but takes multiple bytes for Gamma encoding.

Consider any number $n \in [2^x, 2^{x+1} - 1]$. In normal binary, n requires $x+1$ bits. In variable-byte encoding, $8 \times \lceil (x+1)/7 \rceil$ is required, whereas in Gamma encoding, $2x+1$ bits are required (x bits for offset, and $x+1$ bits for length).

This means any values that can be encoded in 7 bits in binary can be encoded in 1 byte in VB code. In contrast, for a number to fit into 8 bits in Gamma code, it can only be up to 4 bits (or $x + 1 \leq 4$).

So values that require greater than 4 bits and lesser than 8 bits in binary can be encoded in VB by a byte, but not in Gamma by a byte ($x \geq 4$ and $x < 8$).

---

c. Conversely, for what range(s) of gap values is a gamma code shorter than a variable-byte code? Briefly explain your answer.

Solution: The ranges **1 - 15** and **128 - 255**.
1 - 15 take 7 or less bits in Gamma code but 8 bits in VB encoding. **128 - 255** take 15 bits in Gamma code but 16 bits in VB encoding.
From part b), we know that for a number expressable in $x+1$ bits in binary, it can be encoded with $2x+1$ bits using Gamma encoding and $8 \times \lceil (x+1)/7 \rceil$ bits using VBC. So we want to find values of x such that $2x + 1 < 8 \times \lceil (x+1)/7 \rceil$. This is true when $x \leq 3$ and $x \geq 8$.

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:

Document 1: `the way to the school is long and hard when walking in the rain`
Document 2: `the rain has not stopped in days and the school has closed`

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?

Solution: The Jaccard Coefficient without and with the stopword list for the two documents is:

	Doc1	Doc2
without the stopword list	$2/13 = 0.1538$	$3/10 = 0.3$
with the stopword list	$2/8 = 0.25$	$3/5 = 0.6$

We consider the number of distinct words for the two cases, and use that the Jaccard Coefficient is given by $\frac{|A \cap B|}{|A \cup B|}$ to get:

a.) Without the stopword list (we consider all words in our calculations):

Doc	$\lvert\text{Doc} \cap \text{Query}\rvert$	$\lvert\text{Doc} \cup \text{Query}\rvert$	Jaccard
Doc1	2	13	$2/13 = 0.1538$
Doc2	3	10	$3/10 = 0.3$

b.) With the stopword list (we ignore stopwords in our calculations):

Doc	$\lvert\text{Doc} \cap \text{Query}\rvert$	$\lvert\text{Doc} \cup \text{Query}\rvert$	Jaccard
Doc1	2	8	$2/8 = 0.25$
Doc2	3	5	$3/5 = 0.6$

---

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

$$w_{t,d} = tf_{t,d} \times \log_{10}(N/df_t)$$

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.)

Solution: The cosine similarity of the query with the documents without and with the stopword list is as follows:

	Doc1	Doc2
without the stopword list	0	0.25
with the stopword list	0	0.4472

Let us consider the tf-idf weighted vectors for the query and the documents. Note that for the query vector, all terms not appearing in the query will have a weight of 0. Notice also, that for this formulation of tf-idf, words that appear in every document in the corpus have a weight of 0.

For those terms, when we take the cosine similarity of the document vector with the query vector, they will contribute 0 to the cosine similarity, so we do not need to compute the tf-idf values

for them. For normalization of the document vectors, we will need to compute the Euclidean length of the vector $\sqrt{\sum_t tf_{t,d}^2}$

a.) Without the stopword list

For Doc1, we have 11 terms that occur once, and one term that occurs 3 times, so we get a normalization factor of $\sqrt{11 \times 1 + 1 \times 9} = \sqrt{20}$. For Doc2, we have 8 terms that occur once, and 2 terms that occur twice, so we get a normalization factor of $\sqrt{8 \times 1 + 2 \times 4} = \sqrt{16} = 4$.

Term	Query	Doc1	Doc2
school	$1 * \log(2/2) = 0$	$1/\sqrt{20}$	$1/4$
closed	$1 * \log(2/1) = 1$	$0/\sqrt{20}$	$1/4$
rain	$1 * \log(2/2) = 0$	$1/\sqrt{20}$	$1/4$
cosine similarity		0	$1/4 = 0.25$

b.) With the stopword list

Ignoring the stopwords, for Doc1, we have 7 terms that occur once for a normalization factor of $\sqrt{7}$. For Doc2, we have 5 terms that occur once for a normalization factor of $\sqrt{5}$.

Term	Query	Doc1	Doc2
school	$1 * \log(2/2) = 0$	$1/\sqrt{7}$	$1/\sqrt{5}$
closed	$1 * \log(2/1) = 1$	$0/\sqrt{7}$	$1/\sqrt{5}$
rain	$1 * \log(2/2) = 0$	$1/\sqrt{7}$	$1/\sqrt{5}$
cosine similarity		0	$1/\sqrt{5}= 0.4472$

---

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:

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

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]

Solution: The similarity of the query with the documents, without and with the stopword list is:

	Doc1	Doc2
without the stopword list	1	1.875
with the stopword list	2.25	1.875

a.) Without the stopword list

For Doc1, the maximum term frequency is 3 for the word *the*. For Doc2, the maximum term frequency is 2 for the words *the* and *has*.

Term	Doc1	Doc2
school	$0.25 + (0.75 * 1/3)$	$0.25 + (0.75 * 1/2)$
closed	$0.25 + (0.75 * 0/3)$	$0.25 + (0.75 * 1/2)$
rain	$0.25 + (0.75 * 1/3)$	$0.25 + (0.75 * 1/2)$
sum	$0.75 + 2 \times (0.75 * 1/3) = 1$	$3 \times (0.25 + (0.75 * 1/2)) = 1.875$

b.) With the stopword list
Ignoring the stopwords, for Doc1, the maximum term frequency is 1. For Doc2, the maximum term frequency is 2 for the word *has*.

Term	Doc1	Doc2
school	$0.25 + (0.75 * 1/1)$	$0.25 + (0.75 * 1/2)$
closed	$0.25 + (0.75 * 0/1)$	$0.25 + (0.75 * 1/2)$
rain	$0.25 + (0.75 * 1/1)$	$0.25 + (0.75 * 1/2)$
sum	$0.75 + 2 \times (0.75 * 1/1) = 2.25$	$3 \times (0.25 + (0.75 * 1/2)) = 1.875$

---

4. (5 points) Remember that given a query q and documents $d_1, d_2, ...$, 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 $|\vec{q} - \vec{d}|^2$.)

Solution: Note that when q and d are normalized, $q * q = d * d = 1$ and $\text{sim}_{\text{cosine}}(q, d) = q \cdot d$. We can then rewrite $|q - d|^2$ as:

$$\begin{aligned} |q - d|^2 &= (q - d) \cdot (q - d) = q \cdot q + d \cdot d - 2q \cdot d \\ &= 1 + 1 - 2(\text{sim}_{\text{cosine}}(q, d)) \\ &= 2 - 2(\text{sim}_{\text{cosine}}(q, d)) \end{aligned}$$

Notice that when $p = d$, the Euclidean distance is at the minimum of 0, and the cosine similarity is at the maximum of 1, and that as the cosine similarity decreases, the Euclidean distance increases. Therefore ranking based on decreasing cosine similarity is identical to ranking based on increasing Euclidian distance when q and d are normalized.
