CSC 785: Information storage & retrieval

Srijana Raut(101134199)

Assignment 06 (Test 06)

1. Why is the phrase query important? In what situation we think of phrase query is required. Explain it with examples.

A **phrase query** is a sort of search query that lets the user find documents that contain a group of words that come together in the exact sequence that the query specifies. This idea is crucial to information retrieval **because** it makes it possible to find documents that contain the exact phrase rather than just a few, possibly unrelated terms. **For example,** if we search for "heart attack prevention", the phrase query treats all words together as a phrase and delivers results that are likely to be more closely related to what the user is interested in. Simply, Phrase queries are required in circumstances where the relationship between the words is just as significant as the words themselves. Additionally, when a user searches for papers with a given word sequence in a particular order. Some of the real scenario of the phrase query importance with example are given below;

- Phrase queries make sure that the IR engine provides documents that include the exact phrase when the user is searching for a precise match of a phrase, such as a well-known quote, the title of a book, etc. **For instance**, a student looking for a study on Martin Luther King Jr.'s speech would use the search term "I have a Faith" to locate articles that address this speech particularly.

- In academic research or when searching for specific information, the context in which words appear is crucial. Phrase query helps in finding documents where the terms are used in the correct context. **For example,** "cell division" in biology has a different meaning from "division of cells" in a mathematical context.

- Some words have multiple meanings, and when used together in a phrase they provide clear meaning. A phrase query reduces ambiguity and results in relevant documents. **For example,** for a search query "apple pie", if searched separately, may introduce ambiguity as "apple" could be fruit, or a tech company and "pie" could be food or mathematical constant. But when searched together, results are specifically about the sweet pastry which is what we searched for.

2. Shown below is a portion of a positional index in the format:

term: doc1: ⟨position1, position2, ...⟩ ...

angels: 2: ⟨36,174,252,651⟩; 4: ⟨12,22,102,432⟩; 7: ⟨17⟩;
fools: 2: ⟨1,17,74,222⟩; 4: ⟨8,78,108,458⟩; 7: ⟨3,13,23,193⟩;
fear: 2: ⟨87,704,722,901⟩; 4: ⟨13,43,113,433⟩; 7: ⟨18,328,528⟩;

in: 2: ⟨3,37,76,444,851⟩; 4: ⟨10,20,110,470,500⟩; 7: ⟨5,15,25,195⟩;
rush: 2: ⟨2,66,194,321,702⟩; 4: ⟨9,69,149,429,569⟩; 7: ⟨4,14,404⟩;
to: 2: ⟨47,86,234,999⟩; 4: ⟨14,24,774,944⟩; 7: ⟨199,319,599,709⟩;
tread: 2: ⟨57,94,333⟩; 4: ⟨15,35,155⟩; 7: ⟨20,320⟩;
where: 2: ⟨67,124,393,1001⟩; 4: ⟨11,41,101,421,431⟩; 7: ⟨16,36,736⟩;

Which document(s) if any match each of the following queries, where each expression within quotes is a phrase query: 'fools rush in'?

Solution-

Analyzing the index for these terms:
fools: 2: ⟨1,17,74,222⟩;
4: ⟨8,78,108,458⟩;
7: ⟨3,13,23,193⟩;

rush: 2: ⟨2,66,194,321,702⟩;
4: ⟨9,69,149,429,569⟩;
7: ⟨4,14,404⟩;

in: 2: ⟨3,37,76,444,851⟩;
4: ⟨10,20,110,470,500⟩;
7: ⟨5,15,25,195⟩;

Occurs in Doc2, 4 and 7 i.e., Doc2(1,2,3) Doc4(8,9,10) AND Doc7(3,4,5 and 13,14,15)

Therefore, the phrase 'fools rush in' has matches in doc2, doc4 and doc7.

3. Consider an information need for which there are 4 relevant documents in the collection. Contrast two systems running on this collection. Their top 10 results are judged for relevance as follows (the leftmost item is the top ranked search result):

System 1 R N R N N N N N R R
System 2 N R N N R R R N N N

(a) Which one (either system 1 or system 2) is having high precision, and recall?

Both systems have the same precision and recall.
System 1

Precision for system 1 = 4 / 10 = 0.4 AND Recall for system 1 = 4 / 4

System 2

Precision for system 2 = 4 / 10 = 0.4 AND Recall for system 2 = 4 / 4 = 1

(b) Compute precision and recall at top-5 (ranked results) for both systems.

System 1- Precision = 2 / 5 = 0.4 AND Recall = 2 / 4 = 0.5

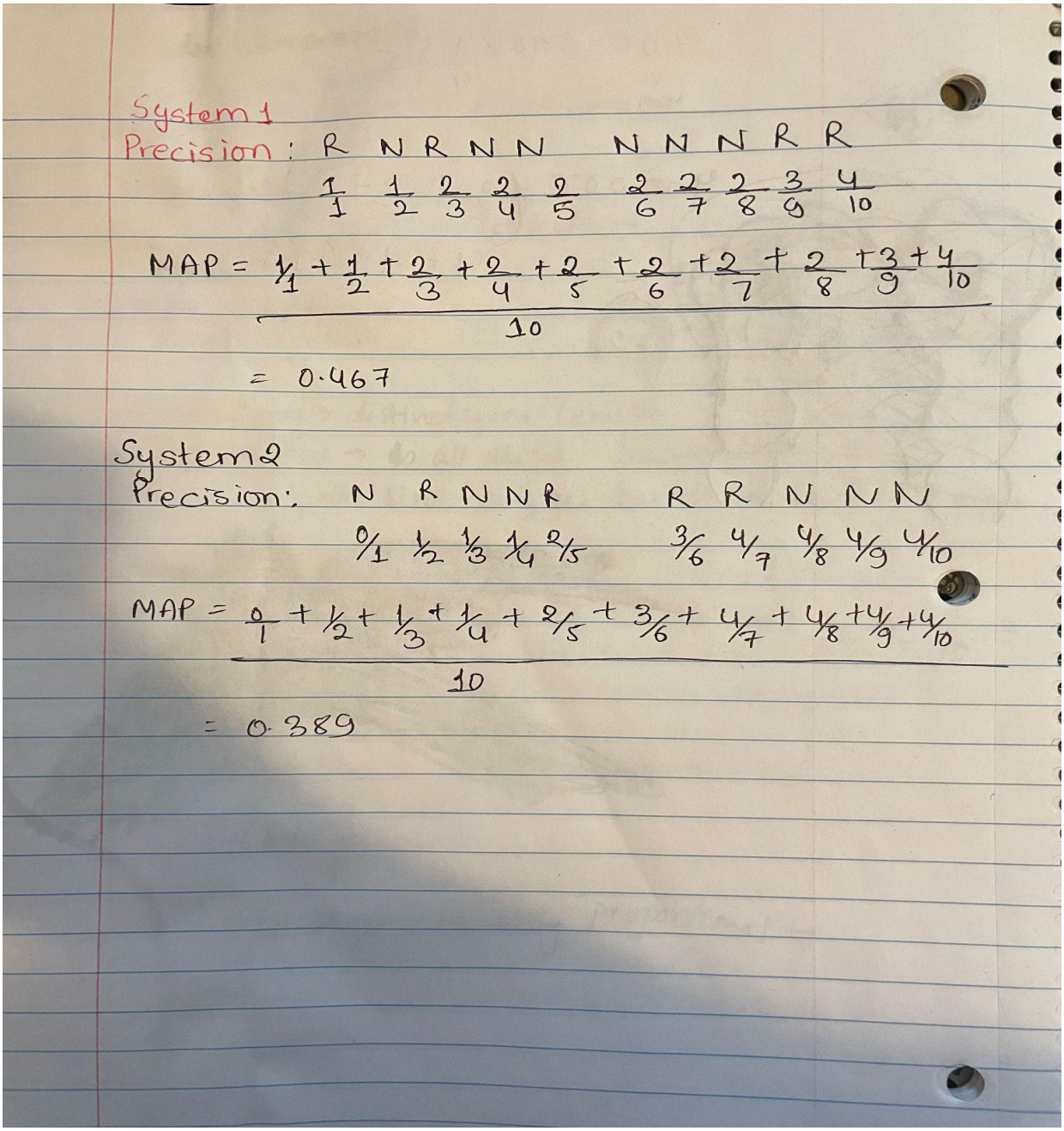
System 2- Precision = 2 / 5 = 0.4 AND Recall = 2 / 4 = 0.5

(c) Compute F_1-score for both systems.

$$F1\ Score\ = 2 * \frac{Precision * Recall}{Precision\ +\ Recall}$$

According to the Above formula, **F1 score for both System1 and System2 is 0.571.**

(d) What is the MAP of each system? What has a higher MAP?
System 1 has the higher mean average precision(MAP) than System 2.



4. Compute *kappa* value based on the following two judgements, for any particular IR system.

		Judge 2 relevance	
		YES	NO
Judge 1 relevance	YES	300	50
	NO	10	50

Kappa value is used to measure inter-rater or inter-judge reliability. It is calculated as:

$$kappa = \frac{P(A) - P(E)}{1 - P(E)}$$

where,
P(A) is the proportion of the times the judge agreed, and
P(E) is the proportion of the times they would be expected to agree by chance

Observed proportion of the times both the judges agreed:
P(A) = (300 + 50) / 410 = 0.853

Pooled marginals:
P(nonrelevant) = (60 + 100) / (410 + 410) = 0.195
P(relevant) = (350 + 310) / (410 + 410) = 0.805

Probability that the two judges agreed by chance:
$P(E) = P(\text{nonrelevant})^2 + P(\text{relevant})^2 = (0.195)^2 + (0.805)^2 = 0.686$

Kappa Statistic = 0.532

5. Explain details of Map-reduce architecture in distributed indexing.

Large datasets can be processed and generated using a parallel, distributed method on a cluster or across distributed computers using the map-reduce architecture concept. This architecture is made to make sure that no one point of failure can take down the entire system and to gracefully handle failures. A cluster of computers or the nodes of distributed systems is used by map-reduce to operate. The dataset has been divided into digestible portions that can be handled separately. This facilitates effective processing in parallel and also makes it simpler to redistribute processing in the event that a node fails. Though they are large enough to minimize the burden of maintaining too many splits, the data is divided into manageable portions. In a distributed system, one or more splits are processed by each node. The parser turns documents into key-value pairs for indexing, usually in the format (termID, docID). A term's ID within a document is represented by its termID, while the document in which the term appears is indicated by its docID. These key-value pairs are processed by the Map function, which then writes them to intermediate files that are typically kept locally on the same node. The intermediate files are divided into segments according to term partitions, where a range of terms are indicated by symbols like [a-f], [g-p], and [q-z]. These segments

are defined by the system operator and match the ranges of the keys. The values connected to a certain key are gathered from the intermediate files by the inverter (reduction phase) and combined into a single list.For each termID, the inverter compiles a list of all docIDs where the term appears, creating the postings list. The process is coordinated by a master node that assigns each term partition to an inverter and handles the reassignment of tasks if an inverter fails. Map-reduce architecture is fault-tolerant and ensures that even with frequent failures, the indexing process can continue without significant interruption.

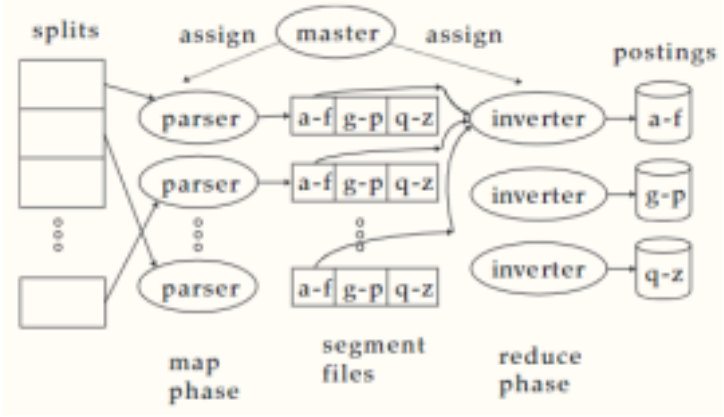


Fig: Distributed indexing with MapReduce

6. Use of Heap's law (empirical law):

a) Consider the fit is excellent, and T > 10^5(= 100,000), for the parameter values b = 0.49 and k = 44. What is the size of vocabulary for the first 1,000,020 tokens?

Solution-

Given,

$T > 10^5$(= 100,000)
$b = 0.49$
$k = 44$

where,
M is the vocabulary size,
T is the total number of tokens,
k and b are parameters that depend on the dataset,

The size of vocabulary for the first 1,000,020 tokens:

$$M = k \cdot T^b$$

$= 44 * 1{,}000{,}020^{0.49}$
= 38,323 (= vocabulary size)

7. For Reuters-RCV1, consider a naive compression approach, we have a collection of size 400,000, and 20 bytes for term, 4 bytes for document frequency and 4 bytes for pointer

posting lists.

a) Compute the total space required for the same? How much space we can reduce if we use blocked storage compression and the value k = 4.

Given,

Collection size = 400,000
20 bytes for terms
4 bytes for document frequency
4 bytes for pointer posting lists

Space required = M * (20 + 4 + 4) = 400,000 * 28 = 11,200,200 Bytes = 11.2 MB

Using blocked storage compression,

For k = 4,
we save (k – 1) * 3 = 9 bytes for term pointers, but need additional k = 4 bytes for term lengths. So, total space requirements are reduced by 5 bytes per four-term block.

Space saved = 400,000 * 1 / 4 * 5 = 500,000 bytes = 0.5 MB
(on top of dictionary-as-a-string compression strategy)

From below, dictionary-as-a-string compression space required = 7.6 MB
Blocked storage compression space required = 7.6 MB – 0.5 MB = 7.1 MB
In comparison to the naïve compression approach, blocked storage compression reduces space from 11.2 MB to 7.1 MB saving 4.1 MB of space.

b) Do you think allocating 20 bytes per term is a good idea? Explain.

Allocating 20 bytes per term is not the most efficient strategy, especially considering the average length of a term in English language is about eight characters. By allocating 20 bytes per term, the fixed-width entry scheme wastes on average 12 bytes per term. This inefficiency becomes significant when multiplied across hundreds of thousands or even millions of terms. The wastage of storage would be substantial, leading to inefficient use of storage. The fixed-width scheme of allocating 20 bytes per term would not be able to accommodate terms that exceed the allocated space. In English, there are terms longer than 20 characters, such as "antidisestablishmentarianism" and "pneumonoultramicroscopicsilicovolcanoconiosis". The inability of storing these words limits the comprehensiveness of the indexing system.

c) Consider a compression strategy: 'dictionary as a string' and keep 4 bytes each for frequency and postings pointer, 3 bytes for the term pointer and 8 bytes on average for the term. Can the space be reduced?

Given,

4 bytes each for frequency and postings pointer,
3 bytes for the term pointer, and
8 bytes on average for the term

Then, space required = M * (8 + 4 + 4 + 3) = 400,000 * 19 = 7,600,000 Bytes = 7.6 MB

In comparison to the naïve compression approach, 'dictionary-as-a-string' compression strategy reduces the space required to 7.6 MB saving 3.6 MB of space.

8. For the same data in question 7, compute the space required when k=4 in case of blocked storage compression. Compute and illustrate the idea with figures.

For blocked storage compression:
Given,
k = 4

We save (k – 1) * 3 = 9 bytes for term pointers, but need additional k = 4 bytes for term lengths. So, total space requirements are reduced by 5 bytes per four-term block.

Space saved = 400,000 * 1 / 4 * 5 = 500,000 bytes = 0.5 MB
This is the calculation of space saved on top of document-as-a-string compression. So, calculating space required for document-as-a-string compression:

M * (8 + 4 + 4 + 3)
= 400,000 * 19
= 7,600,000 Bytes
= 7.6 MB

Thus, the space required for blocked storage compression 7.6 – 0.5 = 7.1

Fig: Blocked storage (k=4)

9. Use the Jaccard coefficient (JC) as shown below,

$$JC(A,B) = \frac{|A \cap B|}{|A \cup B|} \in [0,1]$$

And figure out the similarity between query and documents.

query (q) = ides of March
doc1 (d1) = caesr died in March
doc2 (d2) = the long March, not easy in Winter

Which document is similar to the query?

Given,
Set of words in the query, Q = {ides, of, March}
Set of words in doc1, D1 = {caesr, died, in, March}
Set of words in doc2, D2 = {the, long, March, not, easy, in, Winter}

For doc1:
$|Q \cap D1| = |\{March\}| = 1$
$|Q \cup D1| = |\{ides, of, March, caesr, died, in\}| = 6$

$JC(Q, D1) = |Q \cup D1|/|Q \cap D1| = 1/6 = 0.167$

For doc2:
$|Q \cap D2| = |\{March\}| = 1$
$|Q \cup D2| = |\{ides, of, March, the, long, not, easy, in, Winter\}| = 9$

$JC(Q, D2) = |Q \cup D2|/|Q \cap D2| = 1/9 = 0.11$

Since the Jaccard coefficient (JC) for query and doc1 is higher, document doc1 is similar to the query.

10. Consider the table of term frequencies for 3 documents denoted Doc1, Doc2, Doc3 in Figure A. Compute the tf-idf weights for the terms car, auto, insurance, best, for each document, using the idf values from Figure B.

	Doc	Doc	Doc

	1	2	3
car auto insura nce best	27 3 0 14	4 33 33 0	24 0 29 17

Fig. A: Table of tf values

term	dft	idft
car auto insura nce best	18,16 5 6723 19,24 1 25,23 5	1.6 5 2.0 8 1.6 2 1.5

Fig. B: Example of idf values. Here we give the idf's of terms with various frequencies in the Reuters collection of 806,791 documents. The tf-idf weights can be calculated as:

TF-IDF = TF * IDF

	Doc1	Doc2	Doc3
car	27 * 1.65 = **44.55**	4 * 1.65 = **6.6**	24 * 1.65 = **39.6**
auto	3 * 2.08 = **6.24**	33 * 2.08 = **68.64**	0 * 2.08 = **0**
insurance	0 * 1.62 = **0**	33 * 1.62 = **53.46**	29 * 1.62 = **46.98**
best	14 * 1.5 = **21**	0 * 1.5 = **0**	17 * 1.5 = **25.5**

11. Figure 6.12 shows the number of occurrences of three terms (affection, jealous and gossip) in each of the following three novels: Jane Austen's *Sense and Sensibility* (SaS) and *Pride and Prejudice* (PaP) and *Emily Brontë's Wuthering Heights* (WH).

term	SaS	PaP	WH

affection jealous gossip	115 10 2	58 7 0	20 11 6

• Compute normalized *tf* s.
[Hint. In this example we represent each of these novels as a unit vector in three dimensions, corresponding to these three terms (only); we use raw term frequencies here, with no *idf* multiplier.]

• Using cosine similarities, find out which novel is more similar to *SaS*.
[Hint. Use of dot products.]

Cosine similarity of vector representation of documents:

$$\text{Sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}$$

Where, the numerator represents the dot product of the vectors V(d1) and V(d2) while the denominator is the product of their Euclidean lengths.

The dot product x.y of two vectors is defined as

$\sum_{i=1}^{n} x_i y_i$.

The Euclidean length of *d* is defined by:

$$\sqrt{\sum_{i=1}^{M} \overrightarrow{V_i^2}(d)}$$

Hence, Similarity

$$\text{Sim}(d_1, d_2) = \tilde{v}(d_1) \cdot \tilde{v}(d_2)$$

For normalized *tf*s:
Calculating the Euclidean length for each document's vector:

For SaS = 115.45

For PaP = 58.42

For WH = 23.6

Using the resultant Euclidean lengths for each novel, the normalized *tf* values calculated as:

Term - Affection

SAS = 115 / 115.45 = **0.996**

PAP = 58 / 58.42 = **0.993**

WH = 20 / 23.6 = **0.847**

Term - Jealous

SAS = 10 / 115.45 = **0.087**

PAP = 7 / 58.42 = **0.120**

WH = 11 / 23.6 = **0.466**

Term - Gossip

SAS = 2 / 115.45 = **0.017**

PAP = 0 / 58.42 = **0**

WH = 6 / 23.6 = **0.254**

For Cosine similarity:
Calculating dot product of the normalized vectors to get cosine similarity:

For SaS and PaP: $(0.996 * 0.993) + (0.087 * 0.120) + (0.017 * 0) = 0.999$

For SaS and WP:$(0.996 * 0.847) + (0.087 * 0.466) + (0.017 * 0.254) = 0.888$

$$sim(\vec{v}(SAS), \vec{v}(PAP)) > sim(\vec{v}(SAS), \vec{v}(WH))$$

Thus, the novel *Pride and Prejudice* (PaP) is considerably similar to the novel *Sense and Sensibility* (SaS).