National University of Computer & Emerging Sciences, Karachi
Spring-2023 School of Computing
Midterm Exam-I (Sol)
March 01, 2023, 08:30 AM – 09:30 AM

Course Code: CS4051	**Course Name:** Information Retrieval
Instructor Name / Names: Muhammad Rafi	
Student Roll No:	**Section:**

Instructions:

- Return the question paper.
- Read each question completely before answering it. There are **3 questions** on **2 pages**.
- In case of any ambiguity, you may make assumption. But your assumption should not contradict any statement in the question paper.

Time: 60 minutes **Max: 40 Marks**

Question No. 1	**<CLO # 1>**	**[Time: 25 Min] [Marks: 2X10]**

Answer the following questions briefly using 4-5 lines of the answer book. Be precise, accurate and to the point, only answer genuine query in the question. Each question is of 2 marks.

a. Outline any three differences between stemming and lemmatization.

Stemming	**Lemmatization**
It is a heuristic- rule based approach, generally fast and use a single term. It generates unreadable tokens. Stemming algorithms err on the side of being too aggressive, sacrificing precision in order to increase recall.	It is a rigor process that uses a dictionary and uses context to determine the lemma, considered a slow approach. It generates readable lexeme from the dictionary. Lemmatization offers better precision than stemming, but at the expense of recall.

b. Why inverted index is considered as best data structures for IR? Explain.

An inverted index is an index data structure storing a mapping from content, such as words or numbers, to its locations in a document or a set of documents. In simple words, it is a hashmap like data structure that directs you from a word to a document or a web page. When we compare it with term-document matrix it has less sparsity and quick access, hence it is one of the best option for IR.

c. Why choosing a document unit is important for document processing?

An IR system should be designed to offer choices of granularity as there are two broad categorization of documents with respect to moderen text application (i) Long Documents (Reports and Articles) and (ii) Short Documents (Comments, Reviews, Tweets. etc.). The text retrieval systems are very sensitive to the choice of document or grain szie (index) unit. Hence it is desirable to have a document unit decied at the time of indexing particularly suited to the application vertical.

d. What is a champion list in the context of IR? On what parameter it is based on?

The idea of champion lists (sometimes also called fancy lists or top docs) is to precompute, for each term t in the dictionary, the set of the r documents with the highest weights of tf values for t. There is no reason to have the same value of r for all terms in the dictionary; it could for instance be set to be higher for rarer terms.

e. Explain the syntax and semantic of proximity query in IR.

The proximity query of the form " x y /k". Where x and y are terms from the collection and k is an integer. The result-set contains documents where the two separately matching terms x and y occurrences are within a specified distance say k, where distance is the number of intermediate words.

f. Why Skip Pointers in inverted index are only helpful in executing t1 AND t2 type of query efficiently?

The use of skip pointers in posting part of an invested index is very helpful in quickly getting the result-set of a query of type t1 AND t2, as it need to be a common document that contains both the terms. So, the intersect part quickly move to the higher index to get a match. This is not very beneficial for the query of the type t1 OR t2.

g. How can a bi-word index and a positional index can be combined in a modern information retrieval system?

In modern information retrieval systems – 12% - 15 % queries are phrase queries and mostly about the named entities. Both bi-word index and positional index can be used wisely to scale the performance. The most frequent phrases like celebrity's names "Will Smiths" can be access via bi-word index and the less frequent phrases can be accessed by positional index. In this way a lot of saving can also be done.

h. What do we mean by a query post-processing in IR? What is its complexity?

The post processing in IR is a specialized phase of result-set processing. In some model of IR there is a chance of getting false positive documents against a given query due to the processing approach. The post processing tries to filter all the false positive documents. The complexity of this phase is directly proportional to number of documents in the result-set.

i. When isolated term spelling correction is failed? Give an example.

Consider the query “I’d like to eat dessert", where dessert was typed when desert was intended. Isolated spelling correction approach does not work as all individual terms are correct. When we see the complete semantic of the query we know that dessert was typed when desert was intended. It is a perfect example where isolation correction failed.

j. Why do we want to compress the dictionary in the context of IR?

One of the primary factors in determining the response time of an IR system is the number of disk seeks necessary to process a query. If parts of the dictionary are on disk, then many more disk seeks are necessary in query evaluation. Thus, the main goal of compressing the dictionary is to fit it in main memory, or at least a large portion of it, to support high query throughput.

Question No. 2	<CLO # 2>	[Time: 15 Min] [Marks: 10]

a. Given a set of strings S = {opera, operate, operator}. Build a 2-gram index over S. [2.5]

bi-gram	Posting
$o	opera, operate, operator
e$	operate
er	opera, operate, operator
op	opera, operate, operator
or	operator
pe	opera, operate, operator
r$	operator
ra	opera, operate, operator
te	operate
to	operator
$o	opera, operate, operator

b. Give a list of permuterm vocabulary for the term "operator" [2.5]

$operator
o$perator
op$erator
ope$rator
oper$ator
opera$tor
operat$or
operato$r
operator$

c. Explain how a Single-Pass In-Memory Indexing (SPIMI) improves over Block-sort based indexing approach. [2.5]

SPIMI adds a posting directly to its postings list. Instead of first collecting all termID-docID pairs and then sorting them like BSBI, each postings list is dynamic (i.e., its size is adjusted as it grows) and it is immediately available to collect postings. This has two advantages: It is faster because there is no sorting required, and it saves memory because we keep track of the term a postings list belongs to, so the termIDs of postings need not be stored. As a result, the blocks that individual calls of SPIMI-INVERT can process are much larger and the index construction process as a whole is more efficient hence these are some of the advantages over BSBI.

d. How Zipf's Law helps in building inverted index for IR systems? [2.5]

In information retrieval system, we often need to estimate the distribution of terms in a collection, we only know the size of collection as a whole. Heaps Law help us in determining how many terms will be there in a collection. It states that the vocabulary size can be estimated as a function of collection size $M = kT^b$ where k is a constant and b=0.4 as an exponent used to estimate the number of terms in the collection. It eventually helps in building inverted index.

Question No. 3	**<CLO # 2>**	**[Time: 15 Min] [Marks: 10]**

Consider the following two documents and a query.

Doc 1: w7 w2 w2 w3 w1 w6
Doc 2: w2 w3 w5 w3 w1
Query: w1 w2 w6

Assume the vector space comprises of **V= <w1, w2, w3, w4, w5, w6, w7>** each vector in this model belongs to $\mathbf{R^7}$ and **idf = log(N/DF$_t$).** Assume N=20 is the number of documents;

a. Using the following document frequency of terms DF(V) = {20, 10, 5, 4, 2, 2, 1} and weights using **tf*idf** build the document vectors for Doc 1, and Doc 2 – you can use the simple TF as the term frequency in a document. [2.5]

words	tf(d1)	tf(d2)	tf(q)	df	idf	tf*idf (d1)	tf*idf (d2)	tf*idf (q)
w1	1	1	1	20	0	0	0	0
w2	2	1	1	10	0.30103	0.60206	0.30103	0.30103
w3	1	2	0	5	0.60206	0.60206	1.20412	0
w4	0	0	0	4	0.69897	0	0	0
w5	0	1	0	2	1	0	1	0
w6	1	0	1	2	1	1	0	1
w7	1	0	0	1	1.30103	1.30103	0	0

d1 = < 0, 0.602, 0.602, 0, 0, 1, 1.301>
d2 = < 0, 0.301, 1.20, 0, 1, 0, 0>

b. Using the same document frequency of terms in part (a), build a query vector for the given query using **tf*idf**. as weights. [2.5]

q = < 0, 0.301, 0, 0, 0, 1, 0>

c. Compute the **cosine** (Doc 1, Query) using the formula developed for efficient computation of an angle between the document vector and query vector. [2.5]

cosine (Doc 1, Query) = **< 0, 0.602, 0.602, 0, 0, 1, 1.301> . < 0, 0.301, 0, 0, 0, 1, 0>**

Θ = <0 x0 , 0.602 x 0.301, 0.602 x 0, 0 x 0, 0 x0, 1x0, 1x 1,1.301x 0>

Θ = <0 , 0.181, 0, 0, 0, 0, 1, 0>

Θ = 1.181

d. Compute the **cosine** (Doc 1, Doc 2) using the formula developed for efficient computation of an angle between the document vector and query vector. [2.5]

cosine (Doc 1, Doc 2) = **< 0, 0.602, 0.602, 0, 0, 1, 1.301> . < 0, 0.301, 1.20, 0, 1, 0, 0>**

Θ = <**0, 0.181, 0.7224, 0, 0, 0, 0**>

Θ = 0.9034

BEST OF LUCK