Information Retrieval Mid Sem-1 Exam

Name- Sayam Kumar
Roll No- S20180010158

Q-1 Descriptive Questions

Q-1 State the tasks involved in preprocessing text data with suitable examples -

Ans - After we have gathered good amount of data, we need to do preprocessing before inverted index construction. Following steps can be used -

(1) Tokenization → We need to convert documents into stream of tokens and then proceed with index creations. These tokens can then used in query processing.

(2) Normalization → Remove punctuations can help save a lot of memory and vocab list can be made efficient.

(3) ~~Remo~~ Lowercasing / Uppercasing → ~~for~~ efficient query processing, it is always better to convert terms to a particular case (either upper or lower)

(4) Removing Stop Words - The most frequent occuring words (example - a, the, in, of, at) are called stop words. They can be removed before creating index being less significant

(5) Stemming - Transforming the word into its root word (example - mostly to most), removing s/es or any stemmer can help make better index

Name - Sayan Kumar
Roll No - 12018001015 8

(6) Lemmitization → We can also create mappings from completely different words to their root form.
Eg → ate → eat, went → go

Neccesity → We always look for building a scalable Information Retrieval System. Therefore in order to efficiently store and access indices, its neccesary to complete these steps. We cannot afford user waiting considerable long time for their queries.

Q-2 Assuming 1-based indexing, no stemming, no removal of stop words

Shipment → d1: 1
d3: 1

of → d1: 2 | d2: 2 | d3: 2
platinum → d1: 3 | d4: 1
damaged → d1: 4 |
in → d1: 5 | d2: 8 | d3: 7
Delhi → d1: 6 | d2: 6 | d4: 10
India → d1: 7 |
delivery → d2: 1 |
a → d2: 3,9 | d3: 8
parcel → d2: 4 | d3: 5
for → d2: 5

arrived → d2: 7 | d3: 6
new → d2: 10 | d3: 4 | d4: 9
truck → d2: 11 | d3: 9 |
my → d3: 3
and → d4: 2
parcels → d4: 3
are → d4: 4
the → d4: 5
Chipments → d4: 6
today → d4: 7
from → d4: 8

Name - Sayam Kumar
Roll No - S20180010158

Q-3 Ans → Wild card queries contains * to match words

eg → fun* to match words starting with fun

We can create B trees containing backwards addition words. for queries with abc*, B trees are efficiently faster and yield results b/w abc ≤ w < abd.

(*) for queries ending with *, (eg * abc)
we can look upon B tree and find cba ≤ w < cbb.

(*) It is very expensive operation to find queries with (*) in the middle.

Permuterm Index

① Create * ~~and~~ at the end and store words in backwards fashion.

for example word tiger, maintain dollar as ending
→ tiger$, regi$t, regti, retig, r$tige, $tiger

Lookups

X*	$X*
*X	X$*
X*X	X*

Comparsion

Permuterm index is very efficient in retrieving queries by requires a lot more space (quadruple in size). Instant time comparison take place in permuterm index

Name - Sayam Kumar
Roll No - 520180010158
Page No - 4

Q-4 Biwords

In biwords, we index every consecutive pair of terms in text.
eg: 'hello world people' can be indexed as →
- hello world
- world people

(*) We keep biwords as a dictionary term

(d) Advantages

(1) Two word phase query processing is fast

(2) Even longer queries can be broken down to 2phase (biword) queries

(3) Speed of retrieval is increased

(4) More contextual information and relevance can be formed

Disadvantages

(1) Space is increased

(2) False +ves can be there

(3) Not implemented as standard solution

(4) ~~or~~ 'hello world' and 'world hello' are stored differently.

So, seeing ups and downs, these are are not part of standard solution.

Sayam Kumar
Roll No - S20180010158

Q-5

Term	freq	x|w	P(x|w)	P(w)	P(x|w) P(w)
address	321	c|ct	0.000117	0.012	1.4×10^{-6}
cress	80	a| #	0.000144	0.003	4.3×10^{-7}
caress	121	ac|ca	0.000164	0.004	6.5×10^{-7}
access	527	r|c	0.000209	0.021	$4.3 \times 10 \times 10^{-6}$
across	803	e|o	~~0.00321~~ 0.00093	0.031	2.8×10^{-5}
acres	412	es|e	0.00321	0.005	1.6×10^{-6}
acres	412	ss|s	0.00342	0.005	1.7×10^{-6}

Best word = across (2.8×10^{-5})

Name - Sayam Kumar
Roll No - S20180010158

Q6 5 term weighting examples -

① It is important to have term weighting to measure relevance of a term in a document. This also leads to good results if done correctly.

① Raw TF factor → Just keep the no of occurences.
Ex- if Standford occurs in all documents, then it is ranked high.

② Logarithmic Scale - computed as

$$1 + \log_e(tf)$$

This helps in normalizing out high freq. terms with lower ones

③ Binary TF factor - Whether or not a term is there in a document. Either 0/1, Count and relevance is not stored. Results in poor performance.

④ Augmented TF factor - Reduces the range of frequencies to (0.5, 1) providing a normalized view of all terms. Calculated as

$$0.5 + 0.5 \frac{tf}{\max tf \text{ in document}}$$

⑤ Okapi's TF factor → Approximations of 2 Poisson Model

$$\frac{tf}{2+tf}$$

In all of the above, log scale is better.