CS3245 cheatsheet

Indexing

500, 000 bytes = 0.5 MB

Sort-based indexing

Blocked Sort-Based Indexing
Single-Pass In-Memory Indexing

Zipf's law

$\frac{\mathrm{c_f}}{\text{rank of } \mathrm{c_f}}$

Dynamic indexing

logarithmic merge

IR

Inverted index

Term	docID
ambitious	2
be	2

term	doc. freq.
ambitious	1
be	1
brutus	2

Document frequency is number of times the term appear in different documents

Boolean retrieval

AND results in few results, OR results in many results, information overload!

Ranked retrieval

query written in human language
Rank each document with a score in [0, 1] which measures how well the document and query match

Bag of words

doesn't consider ordering of words in document

Add one smoothing

does not favor training data
word with first letter capitalised is different from the word with first letter in lowercase
double count add-one-smoothing maintains the counting separately, still add one but store the second count in another list

$$\frac{\text{count of term in current corpus} + \text{smoothed count of observed term}}{\text{total counts for terms in doc for words in corpora} + \text{total smoothed counts added for terms in corpora}}$$

This models rank documents that contains both terms higher than documents that contain less query terms

Mixture model

- $P(t|d) = \lambda P(t|M_d) + (1-\lambda)P(t|M_c)$
- Mixes the probability from the document with the general collection frequency of the word.
- High value of λ: "conjunctive-like" search – tends to retrieve documents containing all query words.
- Low value of λ: more disjunctive, suitable for long queries
- Correctly setting λ is very important for good performance

Notation: M_c: the collection model; cf_t: the number of occurrences of t in the collection; $T = \sum_t cf_t$: the total number token in the collection.

$$\boxed{\hat{P}(t|M_c) = \frac{cf_t}{T}}$$

$$\hat{P}(t|M_d) = \frac{\mathrm{tf}_{t,d}}{|d|}$$

($|d|$: length of d; $tf_{t,d}$: # occurrences of t in d)

Term count matrices

Each document is a count (column) vector

Language model

Language Models for IR

- Give a query q, rank documents based on $P(d|q)$, which is the probability of d being relevant given q.

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)}$$

- $P(q|d)$ is the probability of q being generated by the language model of d.
- $P(d)$ is the prior – often treated as the same for all d
  - But we can give a prior to "high-quality" documents, e.g., those with high static quality score g(d) (cf. Section 7.14).
- $P(q)$ is the same for all documents, so ignore

Information Retrieval 32

tfxidf

tf weight (log frequency) $1 + log_{10}tf$ if tf > 0

inverse document frequency Meant to lower the weight of more common terms and increase weight for rare terms $\log_{10}(\text{collection size}/\mathrm{df_t})$

score will contain the weights for all terms in the q ∩ d

cosine similarity

$\mathrm{sum}(\mathrm{q}_i * d_i)$

Document: *car insurance auto insurance*
Query: *best car insurance*

wt is just tf * idf

Term	Document				Query						Prod
	tf-raw	tf-wt	wt	n'lize	tf-raw	tf-wt	df	idf	wt	n'lize	
auto	1	1	1	0.52	0	0	5000	2.3	0	0	0
best	0	0	0	0	1	1	50000	1.3	1.3	0.34	0
car	1	1	1	0.52	1	1	10000	2.0	2.0	0.52	0.27
insurance	2	1.3	1.3	0.68	1	1	1000	3.0	3.0	0.78	0.53

Quick Question: what is N, the number of docs? 1,000,000

Doc length $= \sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92$

Score = 0+0+0.27+0.53 = 0.8

Biword index

quiet phone call
2-gram
quiet phone, phone call
False positives
but quiet phone can phone call may exist in different positions

in the document False negatives cannot occur as it appears in posting list of "quiet phone" and "phone call" and will be part of the intersection when merged and returned

Positional index

to, 993427:
⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; . . . ⟩

be, 178239:
⟨ 1, 2: ⟨17, 25⟩;
4, 5: ⟨17, 191, 291, 430, 434⟩;
5, 3: ⟨14, 19, 101⟩; . . . ⟩

▶ **Figure 2.1** Positional index example. The word to has a document frequency 993,477, and occurs 6 times in document 1 at positions 7, 18, 33, etc.

narrow down the potential documents quickly
this is especially helpful when two juxtaposed query terms exist in many documents but whose intersection is small

Biword + Positional index

A hybrid algorithm might first use the biword index to quickly determine the set of documents that could contain the query.
get $d_{to} \cap d_{be}$
Since doc 1 contains be 2 times, process doc 1

Next, the algorithm would verify which candidate documents actually contain the phrase by checking through the positional postings. In both phases, we can use the strategy to process smaller document frequency items first. For efficiency, the biword index could also maintain the document frequency and postings pointer for its individual words, to save the cost of the additional dictionary lookups that would be incurred otherwise.

Skip pointer

skip pointer to point to end of list
10
$1 \to 2 \to 3 \to 4 \to 5$

Dense cluster chance of skipping is high, it would be more effective to have a single skip pointer $1 \to 15$

Overlap measure- Jaccard coefficient

assign number between 0 and 1

$\frac{\|X \cap Y\|}{\|X \cup Y\|}$

for trigram Tue,ues,esd,sda,day
If both words are same, jaccard is 1

4.3 Jaccard with k-Grams

So how do we put this together. Consider the $(k = 2)$-grams for each D_1, D_2, D_3, and D_4:

D_1: `[I am], [am Sam]`
D_2: `[Sam I], [I am]`
D_3: `[I do], [do not], [not like], [like green], [green eggs], [eggs and], [and ham]`
D_4: `[I do], [do not], [not like], [like them], [them Sam], [Sam I], [I am]`

Now the Jaccard similarity is as follows:

$$\begin{aligned}
\mathsf{JS}(D_1, D_2) &= \ 1/3 \ \approx 0.333 \\
\mathsf{JS}(D_1, D_3) &= \ 0 \ = 0.0 \\
\mathsf{JS}(D_1, D_4) &= \ 1/8 \ = 0.125 \\
\mathsf{JS}(D_2, D_3) &= \ 0 \ = 0.0 \\
\mathsf{JS}(D_3, D_4) &= \ 2/7 \ \approx 0.286 \\
\mathsf{JS}(D_3, D_4) &= \ 3/11 \ \approx 0.273
\end{aligned}$$

Soundex

only suitable for context of English

Soundex – typical algorithm

1. Retain the first letter of the word.
2. Change all occurrences of the following letters to '0' (zero):
   'A', E', 'I', 'O', 'U', 'H', 'W', 'Y'.
3. Change letters to digits as follows:
   - B, F, P, V → 1
   - C, G, J, K, Q, S, X, Z → 2
   - D,T → 3
   - L → 4
   - M, N → 5
   - R → 6

Soundex continued

4. Repeatedly remove one out of each pair of consecutive identical digits
5. Remove all zeros from the resulting string.
6. Pad the resulting string with trailing zeros and return the first four positions, which will be of the form <uppercase letter> <digit> <digit> <digit>.

E.g., ***Herman*** becomes H655.

Stop words removal

cause some phrase search to be less precise

Permuterm index

n = n*

Lookup n*
ney$mo = money
n$moo = moon

Lookup m*y
m*y → y$m*
y$mone = money
y$ma =may

Since n* intersected with m*y gives money, appears in both lookups, hence money is the output term.

For "moon"
moon$
oon$m
on$mo
n$moo
$moon

Heuristics

avoid unnecessary / time consuming computations
Index elimination
Champion list - Tiered lists
Impact-ordered postings - Early termination
sort by weight of the term frequency in the document
because not sorted by document id, cannot find the intersection
terminate when a fixed number or below a wf threshold (e.g. threshold 0.5, anything lower will not be included)
take the union as there's no concurrent traversal

Cluster pruning

leaders are chosen at random, and followers are precomputed

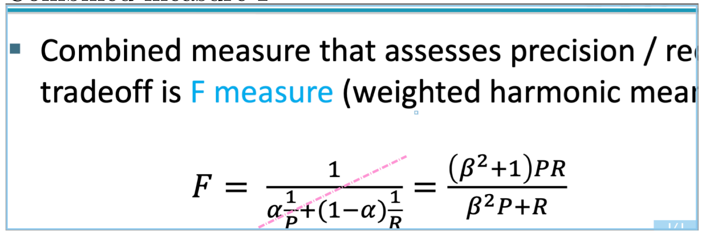
a document will be the follower of the leader whose cosine similarity with the document is the highest

Note: When computing cos similarity between docs, all the terms have to be considered not just the query

Only need to rank the documents in one cluster

Measure of search engine
Accuracy = (tp + tn) / (tp + fp + tn + fn)

Combined measure F



Harmonic mean, balanced
$\frac{2PR}{P+R}$

P = $\frac{\text{relevant documents retrieved}}{\text{retrieved documents}}$

R = $\frac{\text{relevant documents retrieved}}{\text{relevant documents}}$

Interpolated precision
Low point (ignore the drastic drop) does not reflect actual performance because it is going to go up later
Take the highest possible precision from the right

How to know if the doc is relevant?
Boolean search engine, if document contains most words in the query
Evaluate ranked results
For all the relevant documents add (P, R)

Alternatively:
- P = tp / (tp + fp)
- R = tp / (tp + fn)

	Relevant	Non-relevant
Retrieved	true positive (tp)	false positive (fp)
Not Retrieved	false negative (fp)	true negative (tn)

Kappa measure
Eliminate the factor agreement of chance
P(A) = (relevant docs | agree + nr docs | agree) / #docs = 0.925
P(non-relevant) = $\frac{\text{judge\#1 says non-relevant + judge\#2 says non-relevant}}{\text{\#docs} * 2}$
P(E) = $P(\text{non relevant})^2 + P(\text{relevant})^2$
Kappa (K) = [P(A)-P(E)]/[1-P(E)]
Kappa > 0.8 Good agreement

0.67 < Kappa < 0.8 Tentative conclusions

Average pairwise CK	Pairwise CK cols 1 & 4	Pairwise CK cols 1 & 3	Pairwise CK cols 1 & 2	Pairwise CK cols 2 & 4	Pairwise CK cols 2 & 3	Pairwise CK cols 3 & 4
0.629	0.65	**0.757**	0.59	0.558	0.518	0.699

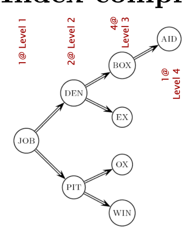
Average Pairwise Cohen's Kappa.

Query refinement
Document level
Standard RF Explicit feedback
Pseudo RF Explicit RF no feedback- Rocchio
RF does not work if there are misspellings/ mismatch
Blind feedback assumes that top k is actually relevant
Term level
Manual thesaurus

To cut down index size
Remove stop words 200MB savings
VB encoding during SPIMI block splitting stage 400MB savings
remove CSS/JS/HTML/Chinese characters, terms with only punctuations posting compression (gap encoding)

Posting compression -Variable byte (VB) coding
Last byte continuation bit is 1
To get the original number, remove the continuation bit and combine
Posting stored as, start number byte + gap byte...
0A and 1B, then the original bit represenation is AB

Index compression without blocking



level multiplied by #nodes at the level
1 * 1 + 2 * 2 + 3 * 4 + 4 * 1

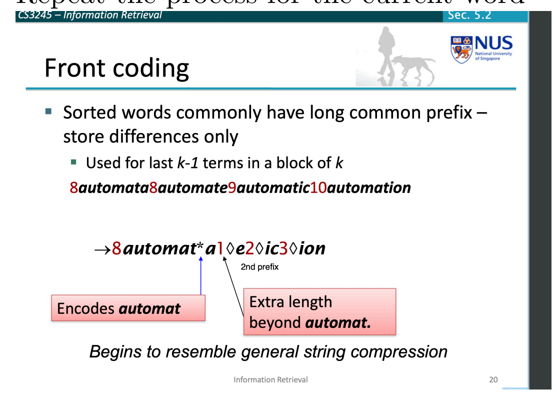
Takes 3 comparison to find the word EX

Dictionary as a string terms 20 bytes
freq 4 bytes → store as pointer instead (1 byte)
postings ptr 4 bytes

Index compression with blocking
Linear search in k term block
if k = 4, construct block from Level 4 number of comparisons is going to increase
terms / block size = comparisons on each level

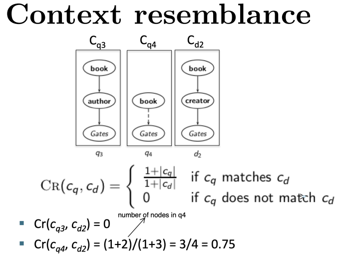
Front coding
Put the ∗ before the first letter that changes for all the cases
Number counted is for the entire word without ∗

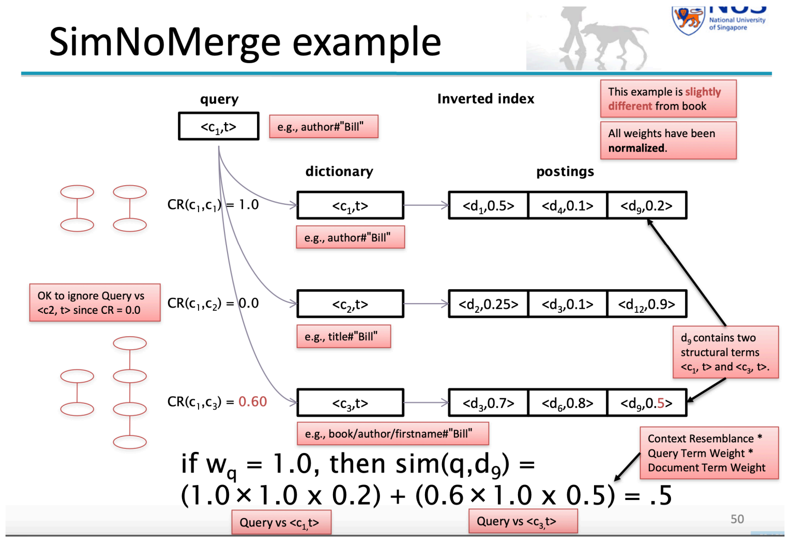
1 represents 1 character that follow after the diamond
Repeat the process for the current word



XML retrieval

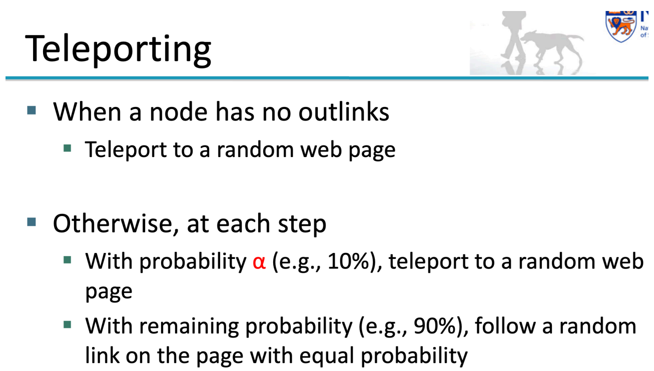
Pseudo xpath /person/nameBob
Context resemblance





Cr(c1, c3) = 1 + 2 / 1 + 4 = 0.60
Check if there are duplicates in structural terms

Link analysis - Pagerank
Damping factor 0.9 (10% teleportation rate)



To get nth iteration, multiply [1/n ... 1/n] with the power of the matrix