IRS

→ Use of IR

- more than 1000TB data produced every year
- websites ↑
- users ↑
- 1000 billion webpages
- document indexed
- 10-20TB text on web
- Recommender systems

- more information of websites
- more choices call for relevant results
- we want personalized results
  eg- content based recommender (movies - ~~genere~~ genre)- we check content of item & suggestions based on content similarity
- To deal with choices, we need IRS

→ HADOOP- distributed database.
clusters
(HDFS)
eg- facebook has billions of data stored in HADOOP.
- also billions of data added every day.

→ Key words in IRS → depend on system

- A (avege repo of docs stored on computers (corpus) are distributed across the web).
  eg- static- file retrieved from same machin
  - google- web based distribution
  ↳ dynamic ↳ called corpus

- Topic for which we need information (information need)
- Relevance - Some docs may contain info. which satisfy need
- How to retrieve? - use IRS
- communicate information need using query
  ↳ what we want

→ Structures of data

- IRS may extract implicit information (preferences)
  eg - if we buy a product, website knows what we like - so implicit.
- explicit preference may / may not be provided.

- structured data (like DBMS)
  - clear, overt semantic structure

ID	name		

↳ we can provide queries to it for retrieval - also complex queries (eg - complex DBMS query)

- But usually, we give simple query to unstructured data.
  eg - google query.

- unstruc data doesn't have clear, overt semantic structure.
- It allows less expensive queries
  - eg - text on webpage etc
    - give data with a keyword.
- It allows less complex queries.

⇒ unstructured data req. IRS, as structured has DBMS queries.

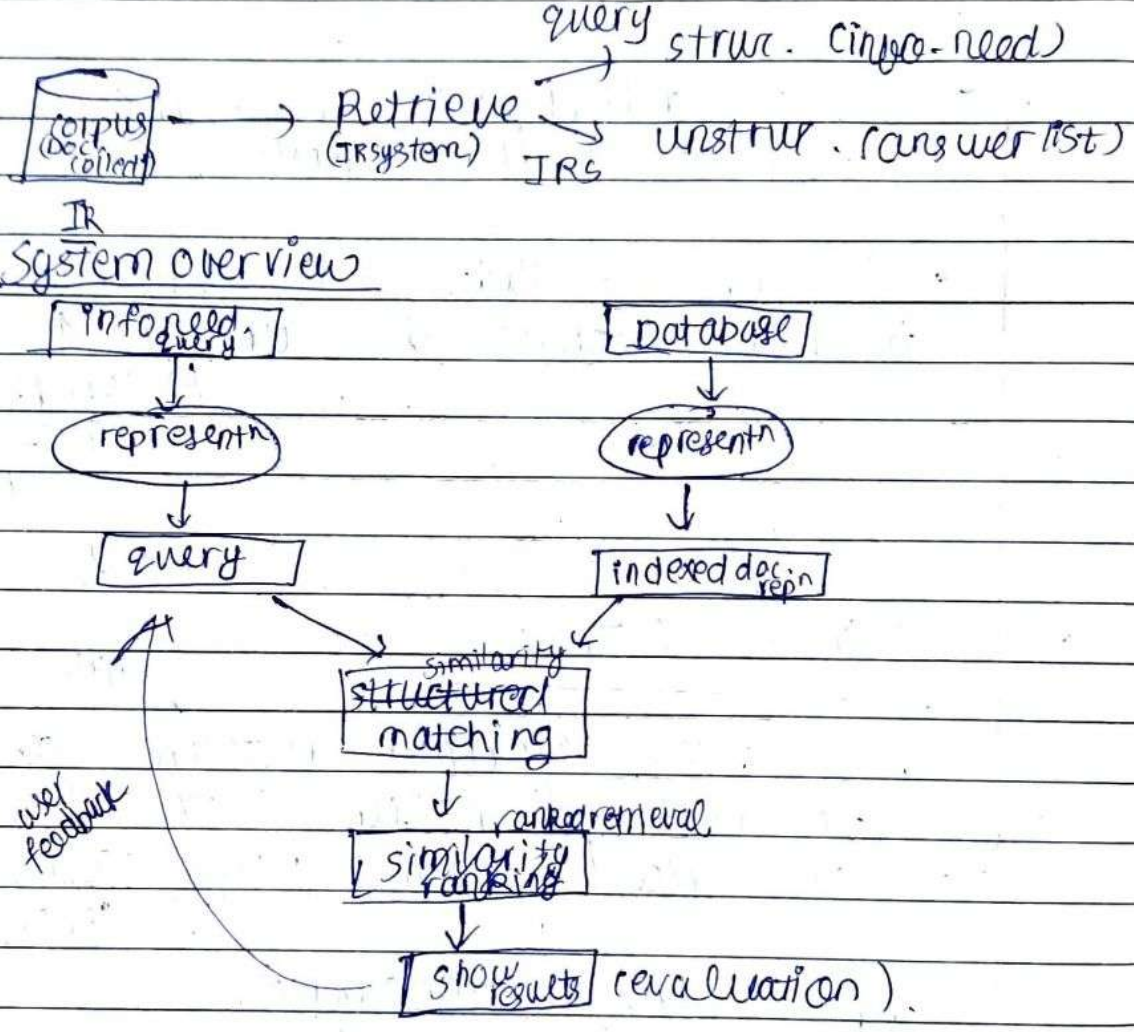
- Earlier, structured ↑, unstruc. ↓
  but now, struc. ↓, unstruc. ↑.
  ↳ companies focus more here

→ Inform^n Retrieval

- It is finding material of unstructured nature that satisfies an info-need from a large collect^n (like docs)
  ↳ corpus
  (like text)
- Apart from web searches,
  eg ⇒ legal info. retrieval.
  - Laptop search
  - email search
  - corporate retrieval

• aim - To find relevant inform^n from large info. set / document set.

↳ show ranked results (for most relevance)



* query also represented as vector

⇒ • IRS is built on corpus.
• B/w IRS & corpus we have a representation.

Page No.
Date :

⇒ IRS V/s Data mining

⇒ • IR is ability to query a computer system & retrieve relevant results.
eg - google web search engine

→ • DM is ability to retrieve info. from one or more data sources in order to combine it, cluster it, visualize it & discover patterns in data.
↳ discover hidden pattern

• we can cluster all docs in one page so that query becomes easy.

→ • BigData is the ability to manipulate huge amt. of data (eg - HDFC) in order to perform DM on the data.
↳ to reduce retrieval time.

→ Terminologies in IR

- websearch
- multimedia IR (eg - lyrics of song is query & we want song video)
- cross language IR (eg - query is India, we want info. related to India as well as "Bharat")
- Recommender system

how to evaluate?

→ How to retrieve docs?

- Precision :- A fraction of retrieved docs relevant to user info need.
  eg- out of 100 docs, 60 retrieved, but user thinks only 40 are relevant.
- ~~Fraction~~ Recall :- Frac^n of relevant docs in collect^n that are retrieved

  · If recall more, then more docs retrieved so precision less.
  · So, we must maintain a balance b/w them both.

all unique words

→ Well known conferences (& For Termpaper - only from 2020 onwards)

- SIGIR (special interest group on IR)
- ACM conference on multimedia retrieval (ICMR)
- KDD (Knowledge discovery & DM)

→ eg- In 37 play docs. of shakespeare, let our query - words having Brutus & Caeser but not calpwenia?

- possible answers
  ⇒ manually search
  ⇒ grep command on Linux
  ↳ time consuming, one script doesn't ~~work~~ words

⇒ Ranked retrieval - not supported using grep

Best solution - we preprocess the doc corpus in advance in such a way we can process query faster.

⇒ we use - Term document incident matrices

	play1 col 1	col 2	...	play37 col 37
all unique words → word1	0	1		
word2				
!				
wordn				

- 0 - word not present
1 - word present.
- every cell is filled in this way.

eg- doc1 = I am
doc2 ⇒ Hi I.
so, words = 3 unique
2 docs. - now fill this

- Now ~~fro~~ for Brutus & C, we extract row of Brutus.

Brutus ⇒ 110100
same for caesar
caesar ⇒ 110111 → Bitwise and
110100 ⇒

- Now we extract vector of calpurnia.
calpurnia ⇒ 010000
⇒ we complement as we ~~don't~~ want not calpurnia.

So, Not Calpurnia = 10111

- Now we "and" this with Brutus & Caesar

```
 110100
 101111
--------
 100100
```

→ The 2 one's represent our docs. which satisfy the query (play 1 & play 4)

→ Problems with term doc. incident matrix

- Frequency of word not considered
- Position of word not considered
  - ↳ eg- in research, if word present in keyword, then more relevant to query.

- eg- N = 1 million docs., each with 1000 words
  - avg. 6 bytes / word
  - ∴ 6 GB of docs.
  - if in harddisk, then everytime we need to bring it in main memory.
  - if in main memory then expensive

- let 5 lakh unique words
  ∴ matrix is 5 lakh rows × 1 mil. cols.

- sparse - dispersed
- dense - collected

Page No.
Date :

- also matrix is extremely sparse.
any column has max^m 1000 ones
(out of all 5 lakhs words only 1000) would be one in worst case
- so sparsity = 1000 x 100 / 500000 = 0.2
- (also, at max 1000 x 1 mil = 1 billion ones in doc.)

→ Inverted index
↗ better way
- we only record those docs. in which term/word is present.
eg- Brutus only present in docs. 1, 2, 4, 11, 31, 45, 73, 74
they have 1 in Brutus.

- similarly for Caesar & Calpurnia.
Brutus = (1, 2, 4, 11, 31, 45, 73, 74)
↳ we can use array if static corpus
Calpurnia → (1, 5, 7) only 3, so space wasted.
↳ Fixed array would waste space.
- we can use Linked list when corpus is dynamic
- so array for ~~time~~ seeking time
- so, we must use optimized Data structure
- we can also use Hashing

→ postlist
- any row is posting list (1, 2, 3 ...)
(3, 4, 5, ...)
dictionary (words) → Brutus - (1, 2 ...)
Caesar - (3, 4, ...)

- To find Brutus & Caesar using linked list, we can use merge operation of linked lists. (Intersection)

⇒ merge → we need atleast space of 2 (1 for doc.id, 1 for next pointer info.)

- In merge - eg for Brutus & Caesar,
  1) Locate Brutus in ~~Dictina~~ dictionary
     - retrieve its postings
  2) Locate Caesar in the dictionary
     - retrieve its posting.

Brutus - 2 → 4 → 8 → 16 → 32 → 64 → 128
Caesar - 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

- Now we apply the merge funct^n by running through the 2 postings simultaneously, O(x+y) worst case (like in merge sort)

Intersect function:
- if data at 2 ~~point~~ running pointers match, the we store it in answer & increment both pointers
- else, ~~the pointer having~~ in case of mismatch, pointer having lesser data gets incremented

→ · To create term from document, till now we just created the term. (word)

· In every row, no repetition of term.
· If term present put 1, else 0. ∴ Boolean Retrieval.

· So before the query, we need to present the document for constructing the matrix.
· Then we extract vectors based on our query.

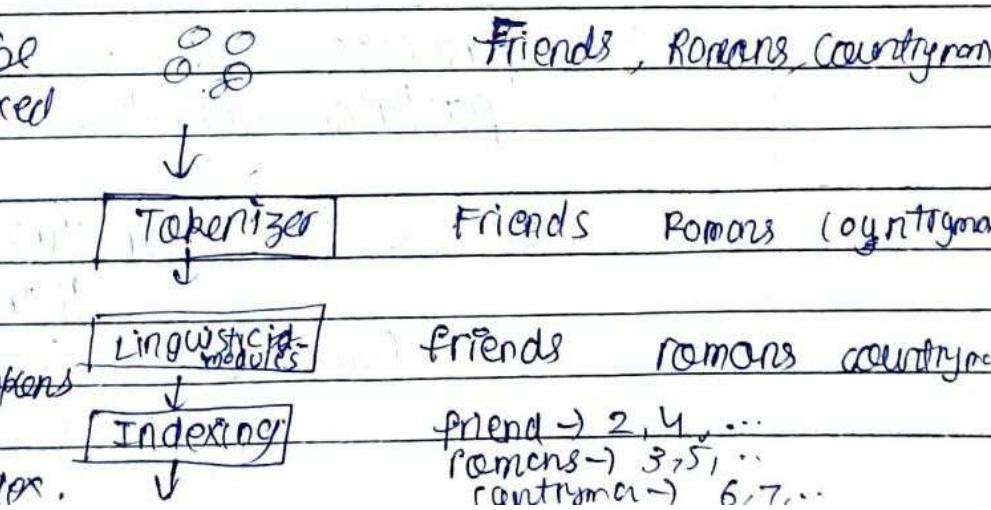
sparsity = total no. of words in doc / total unique words

· Sparsity is biggest problem of Term doc. matrix

→ · Inverted index solves these problems
· we only store those docs containing the words
· The words are dictionary & the docs present (vector) is posting.
· To process query, we use merge operation.

part of Boolean retrieval

→ **How are inverted index prepared?**



steps of indexer:
- To generate tokens we pass the docs & identify the terms.
- Then we arrange the terms in alphabetical order.
- Accordingly, ids are assigned.

words	id
caesar	1
caser	2

- We convert this to document frequency.

caeser	2

→ doc. frequency.

every optimiz^n → eg- query ⇒ Brutus & calpurnia & caesar.
what is the best order for optimiz^n?

eg- Brutus ⇒ | 2 | 4 | 8 | 16 | 32 | 64 |

calpurnia ⇒ - - -
caesar ⇒ - - -

- We process the 2 smallest postings first.

so, order ⇒ Brutus & calpurnia then caesar.
(order of ↑ frequency).
⇒ process the smallest one first.

- Boolean model is the simplest model.
- It was primary commercial tool for 3 decades
  eg- WestLaw uses it.

→ - Before parsing a doc., we need to see:
  - what is format of file (pdf, word, xml etc.)
  - Language of doc. (eg - mail = french, attachment = german)
  - Type of character set (cp1252, UTF-8 etc.)
- Each of this is a classific^n problem.

- Docs. being indexed may be of diff. language - a single ~~single~~ index multiple lang.
- Multiple lang. in 1 doc. may exist
- Open source libraries may help.

→ To what unit we consider a doc.?
  → eg- 1 ppt is a doc., or 100 ppts a doc.
  - 5 attachments ⇒ 5 docs or 1 doc. (in emails).
  - a group of files (eg- ppt of latex split over HTML pages)

→ Some preprocessing techniques

(1) ⇒ Tokenization → to generate tokens we parse the doc.

- I/p = "friends, humans & countrymen"
- after each space/ punctuation we generate a token (usually). (here 4)

- Token is an instance of sequence of characters
- Each token generated is a candidate for index entry in dictionary.

⇒ Issues in Tokeniz^n

1) eg- Finland's capital → Finale
   - Finland and s?
   - Finlands
   - Finland's
   } all 3 diff. Then how to compare.

   (only) or all 3 ✗ equivalence class

   eg- Hewlett-Packard
   - Hewlett & Packard as 2 tokens?

   eg- state-of-the-art
   - co-education
   - lowercase, lower-case, lowercase?
   - It can be effective to get the user to put it in possible hyphens

   eg- San Francisco - 1 or 2 tokens?

2) Date Format

   - 3/20/91, Mar. 12, 1991 20/3/91
   - Aadhar/License Nos. space
     - (891) 123 456
     - (891) 123-234

3) → Language

- eg- some languages do not have space. (german)
- in some languages we read from right to left (& not.. from left to right)
  - ↳ Japanese.
    - ↳ mix of languages.

(2) ⇒ Stop words

(Linguistic Modules)

- with a stop list, you exclude from the dictionary entirely the commonest words.
  eg- a, an, the ....
  (little semantic content)
- They occupy 30% of doc. For top 30 words.
- But in flights, A to B, King of Denmark
  - ↳ to is imp. here.
  - so this stopword important.

- If space (storage) is an issue, we need to remove them.
- But if space not an issue, we can index them as terms.

→

(3) → Normaliz^n of terms → all mapped to same token
↳ equivalence class.

- We need to normalize words in indexed as well as query words into the same form.
  - Eg. to match U.S.A. & USA
    ↳ Both retrieved
- Result is terms - a term is a normalized word type, which is an entity in our IR system dictionary.

(4) →

So, we need to make system smart.

→ But,

French - résumé & resume
↳ they become same after normaliz^n

(5) →

German - Tuebingen & Tübingen

→ Case folding

- convert everything to Lowercase.
  ↳ issues
    ↳ eg - General Motors (company)
    - Fed or fed
    - SAIL, sail

(6) →

Page No.
Date :

↗ exam

- query - C.A.T , we must not get cat.

eg- window
↙ ↳ operating system
house's window
(small w) ? (capital W)

(4) → Thesauri & soundex

- Handling synonyms & homonyms -
  - car = automobile. } both should be retrieved
  - color = colour
- equivalence classes must be done → (car, vehicle) → automobile
- spelling mistakes can be solved by soundex using phonetic heuristic

(5) → Lametiz^n → but does handle propernoun (eg americans) not ↳ america

- converts word to root form
eg. am, are, is → be
car, cars, car's, cars' → car.
- Reduce inflectional / variant forms to base forms.

(6) → Stemming → language dependent.

- chops off the words.
eg- automate(s), automation, automation, → automat.

- Reduce terms to their roots before indexing.

→ Porter's stammer → an algorithm for stemming.

← Typical rules in Porters algo.:

weight of word sensitive rules

- 1) - sses → ss
  2) - ies → i
  3) - ational → ate
  4) - tional → tion

} chopping rules

- after chopping if (m>1),
  - EMENT

eg: replacement → replace (m>1) ✓
cement → c (m>1) ✗
↳ m=1 not >1.
so, no chopping
∴ cement remains.

eg- seperational → seperate (3rd rule)

(7) Part of speech tagging → Tags every word (POS)

- Required in recommender system.
- It is defined as the task of labeling each word in a sentence with its appropriate part of speech
  eg- whether word is noun, adjective etc

→

→ POS is good for:
- IRS
- Text to speech
- word sense disambiguation
- Recommender system is of 2 types
⇒ non personalized
↳ average rating considered (like most popular restaurant etc.) and shown to every user.
⇒ personalized - based on personal user interest.

content based
eg - we watch several movies
- It works based on feature similarity of items (eg - songs, director, genre etc.)
- problem:-
always only similar item shown, no diversity

collaborative filtering
↓
- based on similarity of users
eg - A likes 5 movies
B likes 5 movies
- Both give similar ratings to movies.
- So A & B have similar choices
- so, a new movie comes. A watches it. If A gives it high rating, it can be recommended to B.

works based on star rating
↓
we cannot understand ratings just based on stars.
↙
user don't give review perfectly just stars
there are many features (fine grains)

- Instead of rating, we can use sentiment analysis
- user comments on features of item.

we can analyse it.

1) ~~2 types~~ If we analyze entire review, then its document analysis
2) Other is feature/aspect level analysis.
   - eg- mobile's voice quality is good
   - Here feature = voice quality, & opinion is good +ve.

eg- mobile is good... ~~as~~ and I brought it....
- opinion = good /+ve

3) sentence level.
   - we analyse the sentence.
   - eg- The phone is bad.
     - opinion = -ve.

→ - For every feature, we extract the rating & calculate similarity based on user.
- How to extract feature & opinion from review?
→ If a word is a frequent noun, then its a feature.
- we can use POS tagging for this.
→ opinion could be an adjective / adverb
- we can see nearer words, if there are any adverb or adjectives, they can be opinions.

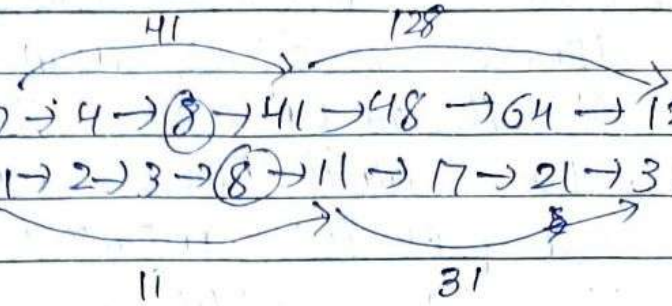
* Heuristic doesn't give optimized answer, but relatively good soln.

post list size ↑ ↗ post list size

Page No. / Date :

→ merge gives O(m+n)

→ How to improve posting list merges through skip pointer



- where do we place skip pointers?

a heuristic approach ✓

⇒ if length of list is n, at the gap of $\sqrt{n}$ place a skip pointer
(eg- if list = 16, skip pointer at $\sqrt{16}$ = 4 intervals)

can be used for "and", not "or"

eg- if we are 6 have processed 8,
p1 reaches 41
now p2 reaches 11.
is a skip pointer available at 11?
Yes there is.
also, 31 < 41
so, we skip, and p2 reaches 31.

→ Trade off

- more skips → shorter skip spans → more likely to skip
  ∴ But lots of comparisons to skip pointers.
- Fewer skips → few pointer comparisons, but then long long skip spans → few successful skips.

- Best case ⇒ $O(\sqrt{n})$
- I/O oper's are also imp.

→ Phrase queries

Eg - we want to retrieve "stanford university" - a phrase, we do not want "I am in Ahmedabad university", then ~~it should~~ this doc. should not be retrieved.

- So, our previous method is not good for phrase queries.
- There also, may not be space ⇒ "stanforduniversity".
  - ↳ user is free to enter
  - ↳ 10% of queries have phrase terms.

heuristic ⇒
- we change the index ⇒ biword indexing
- simple approach
- If doc. contains - "Friends, Romans & countryman", we had 3 tokens
- with biword indexing - 2 terms ⇒ "Friends Romans" & "Romans countryman".
- If only "Romans," then none of the 2 tokens match.

- But in "state-of-the-art", there is no guarantee for answer
- we can get false +ve
  - ↳ model predicted +ve, but actually not.
- In 4 to 5 words in the query biword indexing fails.

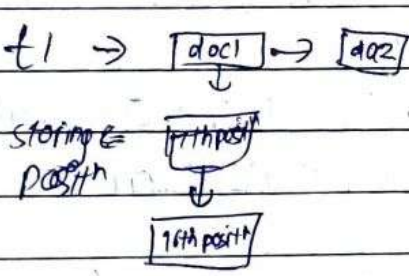
eg - Longer phrases can be processed by breaking them down.

- eg- stanford university palo alto
  ↳ "stan. univ." & "palo alto".
  ↓
  not correct.

Problems - Index blowups due to more no. of words.

heuristic 2) - Positional index help.
- we can store posit^n of word in the document.

- till now:-



} we store doc. in which term present & posit^n where term present.

eg - <be: 993427;

be appears in 1,2,4,5

1: 7,18,33,72,86,231 (be is present in posit^n 7,18... in doc1)
2: 3,149
4: 17,191,291,430,434
5: 363,367,... >

→ · doc. freq ⇒ in how many docs. is term present
→ · collect^n freq ⇒ in doc1 10 times + in doc2 20 times
∴ 10+20 = 30

Page No.
Date :

· For phrase queries, we use merge algo. recursively at doc. level.

→ eg - we want ⇒ "to be or not to be"

⇒ to :

→ doc. no. 2 → position in doc.

→ 2: 1, 17, 74, 222, 551; 4: 8, 10, 190, 429, 433; 7: 13, 23, 191 ...

⇒ be :

→ 1: 17, 19; 4: 17, 191, 291, 430, 434; 5: 14, 19, 101 ...

⇒ Here, 429, 430, 433 & 433 is possible pf explored.

· we can get "to be or not to be" 429 & 430 are consecutive.

→ Vector space / Ranked retrieval (TF-ID model (RR)

· earlier in boolean retrieval - doc. either match or not

· Boolean is not good for user experience also

· appliⁿ processing is too much

· users wait to much in worst case

↳ too many words then too much time.

retrieved results have order; so fewer or famine (no doc or many doc) not a problem

· doc. gets retrieved, or entirely not retrieved.
· many doc. may be retrieved, or 0 maybe retrieved.

⇒ · Rather than boolean model, in ranked retrieval system returns an ordering over the top docs. in collection for query
· Free text query - Rather than query of operators, user's query is in human language
· Ranked ret. is used for free text processing.

· Here, Large result set is not an issue.
- we just show the top K (≈ 10) results
- we don't overwhelm the user.

· scoring as basis for R.R
· we assign score in [0, 1]
0 ↵ ↳ perfect match
no match.

· If query term doesn't occur in doc. score = 0
- Else if more frequent query for doc., score higher.

→

Page No.
Date :

→ Jaccard similarity (b/w A & B)

⇒ Jac(A, B) = |A ∩ B| / |A ∪ B|
⇒ Jac(A, A) = 1
⇒ Jac(A, B) = 0 if A ∩ B = 0

- A & B don't have to be same size
- Jac(A, B) assigns value in range 0 to 1.

eg - doc 1 = caesar died in march
doc 2 = the long march
query = ides of march.

→ Term doc count matrix

$Jac(q, d1) = \frac{1}{6}$

$Jac(q, d2) = \frac{1}{5}$

- Here as d2 is shorter, it is more relevant to query
  ↳ it is not necessarily true
  ↳ this is a problem with Jac. sim.

- if d2 = long ides march
  $Jac(q, d2) = \frac{2}{5}$

Problems of Jaccard:
1. Jac. doesn't consider term freq.
2. Rare terms are more imp. than frequent terms

- We need normaliz^n to make vector magnitude same.
- To prevent shorter-longer doc: problem, we normalize them.

For normaliz^n, $\vec{a}.\vec{b} = |A \cap B| / \sqrt{|A||B|}$

↓ to normalize.

→ · Now lets consider freq. of words in doc.
↳ so each vector is a count vector

Term doc. count matrix →

document.

doc A

eg- Antony (55) → Antony appears 55 times in A.

→ · Considering bag of words model for IR, we ignore posit^n & index for IR

∴ A B C & C B A are same vectors
↳ bag of words model (no ordering)

- The term freq. (tf) = no. of times a words occurs in doc.

- A doc. with 10 occurrences of a term is more relevant than 1 occurrence of ~~a doc~~ term.
⇒ ✱ (But not 10 times ~~as~~ relevant!)

∴ Relevance doesn't increase proportionally with tf

∴ Log freq. weithing :-

- Log freq. of term t in doc d

$$\Rightarrow w_{t,d} = \begin{cases} 1 + \log_{10} tf_{t,d} & tf_{t,d} > 0 \text{ if } tf_{t,d} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

∴ 0 → 0 (0 comes) → also $w_{t,d}$
1 → 1
2 → 1.3
10 → 2
1000 → 4
→ these are weights of relevance.
∴ Not linearly increases.

∴ Score = $\sum_{t \in q \cap d} (1 + \log tf_{t,d})$

query ∩ document ; common terms.

- The score is 0 if query term not present in the doc.

- Rare terms are more informative than frequent terms
  - like stopwords.

* log change magnitude, not linear (so, normalizing)

eg "arachnocentric" is a rare word.
↳ so give more weightage.
. so, a query with this word, then the doc. containing the word is highly relevant.

. To give weightage to the terms (based on in how many docs. is it present)
df_t is term doc. frequency of t.
df_t is an inverse measure of informativeness of t
$df_t \leq N$
. we define

$$idf_t = \log_{10}(N/df_t)$$

eg- for 1 mil. docs,
if calpurnia appears only in 1 doc.
i.e. $df_t = 1$,

$$idf_t = \log_{10}\left(\frac{10^6}{1}\right) = \boxed{6}.$$

. if "the" appears in all docs,
i.e. $df_t = 10^6$,

$$idf_t = \log_{10}\left(\frac{10^6}{10^6}\right) = \boxed{0}$$
↳ no info.

→. collect. frequency of t is total occurrences of t in entire corpus/all docs.
. so, it doesn't help us in knowing importance of that word.
↳ it may appear in only 1 doc, but 10000 times, so CF = 10000 is bad

→ tf-idf indexing

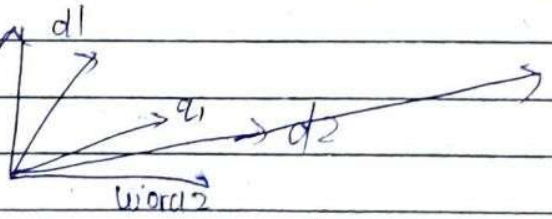
$$w_{i,d} = [1 + \log(tf_{i,d})] \times \log_{10}(N/df_t)$$

- $tf_{i,d}$ → occurrences for relative relevance
- $\log_{10}(N/df_t)$ → rarity

→ We represent our documents as vectors.
- Terms are axes
- Docs are points / vectors.
- Very high dimensional
- Very sparse matrix

→ - query also must be vector.
- Represent queries as vectors.
- Rank docs. according to proximity / similarity of query
- similarity $\propto \frac{1}{\text{distance}}$.

- Euclidian distance is not a good measure. → it is large for vectors of diff. lengths.



- even though q_1 & d_2 are similar distribution, euclidian may give large distance

Cosine similarity is a good measure

- Instead of distance we could use similarity measure → eg- cosine similarity.
- It gives b/w 0 & 1 as cos is b/w 0 & 1
  - 0 → fully match → vectors overlap
  - 90° → do not match.
  - near to 1 → both match
  - near to 0 → do not match.

- $\cos(V_1, V_2) = \frac{\vec{V_1} \cdot \vec{V_2}}{|\vec{V_1}||\vec{V_2}|}$

→ normalized length cosine sim :-
$\cos(\vec{q}, \vec{d}) = \vec{q} \cdot \vec{d}$

eg- query → "gold silver truck"
d1 = "shipment of gold damaged in a fire"
d2 = "delivery of silver arrived in a silver truck"
d3 = "shipment of gold arrived in a truck"

→ unique terms = shipment, of, gold, damaged, in, a, fire, delivery, silver, arrived, truck.

ordered → a, arrived, damaged, delivery, fire, gold, in, of, silver, shipment, truck.

Page No.
Date:

	q	d_1	d_2	d_3	df
a	0	1	1	1	3
damage arrived	0	0	1	1	2
damaged	0	1	0	0	1
delivery	0	0	1	0	1
fire	0	1	0	0	1
gold	1	1	0	1	2
in	0	1	1	1	3
of	0	1	1	1	3
silver	1	0	2	0	1
shipment	0	1	0	1	2
truck	1	0	1	1	3 2

re calculate

- Here, corpus is small, hence we didn't normalize.
- But if yes, the we would have put

eg - $1 + \log_{10} 2 \Rightarrow 1.3$.

df	$idf \Rightarrow (\log \frac{N}{df_i})$
3	0
2	0.17
1	0.47
1	0.47
1	0.47
2	0.17
3	0
3	0
1	0.47
1	0.47
2	0.27

weights

q	d_1	d_2	d_3
0	0	0	0
0	0	0.17	0.47

$\rightarrow idf \times tf_{d_3}$

↳ this d_1 is $\vec{d_1}$

df
3
2
1
1
1
2
3
3
2 1
2 1
3 2

- now cosine similarity with d_1, d_2 & d_3 is calculated with Q.
- Most similar is the answer.

Q- d_1 = supervised unsupervised artificial learning artificial
d_2 = intelligence machine learning display
d_3 = graphics processor processor
d_4 = processor intelligence ~~artilligence~~ artificial
d_5 = raster display processor

- const. normalized term freq. -inverse doc. freq. index for the doc. called^.

→ unique words = supervised, unsupervised, artificial, learning, intelligence, machine, display, graphics, processor, raster

	d_1	d_2	d_3	d_4	d_5	d_6	idf	$1+\log(tf)$
artificial	2	0	0	1	0	2	0.397	1.477
supervised	1	0	0	0	0	1	0.698	~~1.301~~
unsupervised	1	0	0	0	0	1	0.698	~~1.301~~
learning	1	1	0	0	0	2	0.397	1.301
intelligence	0	1	0	1	0	2	0.397	1.301
machine	0	1	0	0	0	1	0.698	1
display	0	1	0	0	1	2	0.397	1.301
graphics	0	0	1	0	0	1	0.698	1
processor	0	0	2	1	1	3	0.221	1.602
raster	0	0	0	0	1	1	0.698	1

Page No.
Date :

TF-IDF numericals

→ Resolve 2 ques^n's (practice)

Q1) query = "gold silver truck"
d1 = "shipment of gold damaged in a fire"
d2 = "delivery of silver arrived in a silver truck"
d3 = "shipment of gold arrived in a truck"

→ unique words = gold, silver, truck, shipment, of, damaged, in, a, fire, delivery, arrived,

query·idf	word ↓ doc →	query	d1	d2	d3	df	idf	H(tf1)	H(tf2)	H(tf3)	tf1·idf	tf2·idf	tf3·idf
0	a	0	1	1	1	3	0	1	1	1	0	0	0
0	arrived	0	0	1	1	2	0.176	0	1	1	0	0.176	0.176
0	damaged	0	1	0	0	1	0.477	1	0	0	0.477	0	0
0	delivery	0	0	1	0	1	0.477	0	1	0	0	0.477	0
0	fire	0	1	0	0	1	0.477	1	0	0	0.477	0	0
0.176	gold	1	1	0	1	2	0.176	1	0	1	0.176	0	0.176
0	in	0	1	1	1	3	0	1	1	1	0	0	0
0	of	0	1	1	1	3	0	1	1	1	0	0	0
0	shipment	0	1	0	1	2	0.176	1	0	1	0.176	0	0.176
0.477	silver	1	0	2	0	1	0.477	0	1.3	0	0	0.62	0
0.176	truck	1	0	1	1	2	0.176	0	1	1	0	0.176	0.176

sim. of query & d1 = $\frac{0+0+0+0+0+0.176\times0.176+0+0+0+0.477\times0+0}{\sqrt{(0.176)^2+(0.477)^2+(0.176)^2}\times\sqrt{(0.477)^2+(0.176)^2}}$

$= \frac{0.03}{0.387} = 0.077$

Sim. of query & d2

$$= \frac{0.176 \times 0 + 0.477 \times 0.62 + 0.176 \times 0.176}{\sqrt{(0.176)^2 + (0.477)^2 + (0.176)^2} \times \sqrt{(0.176)^2 + 0.477^2 + (0.62)^2 + (0.176)^2}}$$

$$= \frac{0.326}{0.441} = 0.739$$

Sim. of query & d3

$$= \frac{0.176 \times 0.176 + 0.477 \times 0 + 0.176 \times 0.176}{\sqrt{(0.176)^2 + (0.477)^2 + (0.176)^2} \times \sqrt{(0.176)^2 + (0.176)^2}}$$

$$= \frac{0.06}{0.189} = 0.317$$

So, similarity ranking:-

d2 > d3 > d1 for query →

Page No.
Date:

(Q2) d1 = "supervised unsupervised artificial learning artificial"
d2 = "intelligence machine learning display"
d3 = "graphics processor processor"
d4 = "processor intelligence artificial"
d5 = "raster display processor"

⇒ Construct tf-idf matrix.

Unique words = supervised, unsupervised, artificial, learning, intelligence, machine, display, Graphics, processor, raster

docs →							log(N/df)	normalized tf 1+log(tf)					tf-idf (w)				
words ↓	d1	d2	d3	d4	d5	df	idf	1	2	3	4	5	tf1 idf1	tf2 idf2	tf3 idf3	tf4 idf4	tf5 idf5
artificial	2	0	0	1	0	2	0.397	1.3	0	0	1	0	0.519	0	0	0.397	0
display	0	1	0	0	1	2	0.397	0	1	0	0	1	0	0.397	0	0	0.397
graphics	0	0	1	0	0	1	0.699	0	0	1	0	0	0	0	0.699	0	0
intelligence	0	1	0	1	0	2	0.397	0	1	0	1	0	0	0.397	0	0.397	0
learning	1	1	0	0	0	2	0.397	1	1	0	0	0	0.397	0.397	0	0	0
machine	0	1	0	0	0	1	0.699	0	1	0	0	0	0	0.699	0	0	0
processor	0	0	2	1	1	3	0.221	0	0	1.3	1	1	0	0	0.287	0.221	0.221
raster	0	0	0	0	1	1	0.699	0	0	0	0	1	0	0	0	0	0.699
supervised	1	0	0	0	0	1	0.699	1	0	0	0	0	0.699	0	0	0	0
unsupervised	1	0	0	0	0	1	0.699	1	0	0	0	0	0.699	0	0	0	0

Page No.
Date :

→ Document classific^n

eg- Naive Bayes, Bernoulli

→ Role of doc. classific^n in IRS

↳ entertainment
↳ doc1
↳ politics
↳ doc.3
} mapping of doc.

- we have training & testing.
- Repo. of doc. with labels available (eg- doc-1 :- sports etc.)
- when new doc comes, we can automatically classify entertainment or politics /others

- By creating clusters we can reduce our search space.
- more relevant queries
- reduction of search time. (optimiz^n of retrieval time)
- eg - in email :- spam or not spam
- more precision & recall.
- if image, audio, videos etc. already classified, then helpful for specific query like - cat image

incl
mus
occ

- Naive Bayes most popular for text classific^n.

→ Supervised learning

- A doc. d
  classes :- $C = (C_1, C_2, \ldots, C_j)$
  - A training set of doc. with a label from c.

Bernoulli Trial ⇒ 1 variable, 2 values ⇒ 0 or 1
multivariate Bernoulli trial ⇒ multiple variable with 0 or 1 for each test.

$$P(c/d) = \frac{P(c) \times P(d/c)}{P(d)}$$ ↳ for Naive Bayes

· Determine

· A learning classifier to classify class y.
· we assign class y to d
$y(d) \in C$
· we train a model to classify on test object/doc.

⇒ we use Naive Bayes

~~$P(A/B) = P(B)$~~

$$P(c/d) \propto P(c) \prod_{1 \le k \le n_d} P(t_k/c)$$

$\prod$ → multiplication of P(term presence) for all terms

$P(c)$ → prob. of d belonging to class c

n_d → division term ignored as same for all $\frac{3}{5} > \frac{2}{5}$ ↳ $3 > 2$

· If class c is popular, then new doc. is highly probable for new test doc d.
(eg 90 of 100 are C, so C is probable for test d).

· n_d = length of doc. (no. of tokens)
· $P(t_k/c)$ is conditional prob. of term t_k occuring in doc. of class C
· $P(t_k/c)$ as a measure of how much evidence t_k contributes that c is correct class
↳ we check for each term t, how many times it appears in all docs of c.

including multiple occurrences.

Page No.
Date :

- Our goal in naive Bayes is to find best class.
- Best class is most likely or maximum a posterial (MAP) class. Cmap is calculated
- use argmax.

C_{map} $P(C) = \frac{N_c}{N}$ → docs of class c. → Prior prob.
N = total docs in corpus

Conditional prob. : $P(t/c) = \frac{T_{ct}}{\sum_{t \in V} T_{ct'}}$ → eg- t appears 3, 2, & 1 time in doc1 23, $\therefore T_{ct} = 3+1 = 6$

- we make naive bayes positional independance assumption

$P(t_{k_1}/c) = P(t_{k_2}/c)$

↓
1st place or any other position
↳ same prob.

- we need to add 0s conditional probability due to term which may not be in training phase, prob. becomes 0
↳ so need for Laplacian correction.

eg class a, b, c

$P(W/a) = 0$ ↳ if W not in a.

So, we add +1 to numerator, & no. of classes in denominator.

∴ P(c) eg-Test = chinese chinese chinese Tokyo Japan.

3 classes - c
1 class - j

$P(c) = \frac{3}{4}$

$P(j) = \frac{1}{4}$

$P(\text{chinese}/c) = \frac{5+1}{8+6}$ ← 5 chinese in c → laplacian correction

vocabulary: c entire corpus
c ↳ total chinese in doc of c
total words in c

$P(\text{Tokyo}/c) = \frac{0+1}{8+6} = \frac{1}{14}$

$P(\text{Japan}/c) = \frac{0+1}{8+6} = \frac{1}{14}$

$P(c/d_5) \propto \frac{3}{4} \times \left(\frac{3}{7}\right)^3 \times \frac{1}{14} \times \frac{1}{14}$ ← chinese appears 3 times.

- Similarly for P(chinese/j), P(Tokyo/j) & P(Japan/j)
multiple. ⇒ P(s/d5)
. whichever greater is class.

$P(w/c) = \frac{\text{count}(w,c)+1}{\text{count}(c) + |V| \text{ to vocabulary}}$

Page No.
Date :

→ Bernoulli

- Doesn't see term freq. only presence or absence.
- P(t/c) estimated as fract^n of doc. of class c containing term t.
- Probability of non occurence is also calculated.
- Binary values put - 0 & 1.

~~eg P(chinese/c) = 3+1~~

- we calculate probability of each unique term in corpus (to check if term may not appear).

⇒ $P(\text{chinese}/c) = \frac{3+1}{3+2}$ → in how many it appears in class c.
↳ total classes ((c))
↓ total docs of c

$P(\text{Japan}/c) = P(\text{Tokyo}/c) = \frac{0+1}{3+2} = \frac{1}{5}$

$P(\text{chinese}/J) = \frac{1+1}{1+2}$

similarly others.

$\therefore P(c/d) \propto P(c) \times P(\text{Chinese}/c) \times P(\text{Japan}/c)$
$\times P(\text{Japan Tokyo}/c)$
$\times (1 - P(\text{macau}/c)) \times P(\text{shanghai}/c)$

multinomial

Naive Bayes question

training set	doc	words	in c=China?
	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test →	d 5	Chinese Chinese ~~Tokyo~~ Chinese Tokyo Japan	?

→ $P(\text{Chinese}/c) = \frac{5+1}{8+6} = \frac{6}{14} = \frac{3}{7}$

$P(\text{Tokyo}/c) = \frac{0+1}{8+6} = \frac{1}{14}$

$P(\text{Japan}/c) = \frac{0+1}{8+6} = \frac{1}{14}$

$P(c) = \frac{3}{4}$

$\therefore P(c/d_5) \propto \frac{3}{4} \times \left(\frac{3}{7}\right)^3 \times \frac{1}{14} \times \frac{1}{14} \approx 0.0003$

→ $P(\text{Chinese}/\bar{c}) = \frac{1+1}{3+6} = \frac{2}{9}$

$P(\text{Tokyo}/\bar{c}) = \frac{1+1}{3+6} = \frac{2}{9}$

$P(\text{Japan}/\bar{c}) = \frac{1+1}{3+6} = \frac{2}{9}$, $P(\bar{c}) = \frac{1}{4}$

$P(d_5/\bar{c}) \propto \frac{1}{4} \times \left(\frac{2}{9}\right)^3 \times \frac{2}{9} \times \frac{2}{9} \approx 0.0001$

$\therefore$ c is more probable

. . Laplacian correction is for smoothening.

Page No.
Date :

Steps

- From training corpus extract vocabulary (→ unique words)
- Calculate req. $P(C_j)$ & $P(x_k / C_j)$ terms.

For each class C_j in C do:

- $\Rightarrow$ docs$_j$ ← subset of docs for which the target class is C_j

$$P(C_j) = \frac{|docs|_j}{\text{total docs}} \quad (\text{i.e. } \frac{\text{docs of class } C_j}{\text{total docs}})$$

$$P(x_k / C_j) = \frac{n_{jk} + \alpha}{n_j + \alpha(|vocabulary|)}$$

where
k = 1 to all unique words in query

n_{jk} → in how many docs x_k appears. → for Laplacian correctn

n_j = no of words in doc$_j$

$|vocabulary|$ = no of unique words in corpus.

positions = all word position in current doc. which contain tokens found in vocab.

we calculate $\Rightarrow C_{NB} = \underset{C_j \in C}{\text{argmax}} \left(P(C_j) \times \prod_{i \in \text{positions}} P(x_i / C_j) \right)$

↳ for class having max^m value.

→

Page No.
Date :

Bernoulli Naive Bayes classifier

→ check absence for all remaining words in vocabulary

- Considers presence & absence of words (0 or 1)
- ∴ 2 possible values for a word → 0 or 1.

$$P(x_i/c_j) = \frac{N(\text{no. of time word appears in docs containing label } C \text{ } X_i = x_i, C = c_j) + 1}{N(C = c_j) + k}$$

→ laplacian correction

$N(C = c_j)$ → no. of docs with label C

k → no. of values for a word (here 2 - 0 or 1)

	doc ID	words	In c = china?
training set	1	Chinese Beijing Chinese	yes
	2	chinese chinese Shanghai	yes
	3	chinese macao	yes
	4	Tokyo Japan chinese	no.
test set	d 5	Chinese Chinese Chinese Tokyo Japan	?

$$P(\text{Chinese}/c) = \frac{3+1}{3+2} = \frac{4}{5}$$

(3 → chinese appears in 3 docs of c; 3+2 → total docs of c; 2 → appears or not (0,1))

$$P(\text{Tokyo}/c) = \frac{0+1}{3+2} = \frac{1}{5}$$

$$P(\text{Japan}/c) = \frac{0+1}{3+2} = \frac{1}{5}$$

$$P(\text{Beijing}/c) = \frac{1+1}{3+2} = \frac{2}{5}, \quad P(\text{Macao}/c) = \frac{1+1}{3+2} = \frac{2}{5} = P(\text{Shanghai}/c)$$

$P(c) = \frac{3}{4}$ ∴ $P(d) \propto \frac{3}{4} \times \frac{4}{5} \times \frac{1}{5} \times \frac{1}{5} \times (1-\frac{2}{5}) \times (1-\frac{2}{5}) \times (1-\frac{2}{5}) \approx 0.005$

$P(\text{Chinese}/\bar{c}) = \frac{1+1}{1+2} = \frac{2}{3}$

$P(\text{Tokyo}/\bar{c}) = \frac{1+1}{1+2} = \frac{2}{3} = P(\text{Japan}/\bar{c})$

$P(\text{Beijing}/\bar{c}) = P(\text{Macao}/\bar{c}) = P(\text{Shanghai}/\bar{c})$

$= \frac{0+1}{1+2} = \frac{1}{3}$

$P(\bar{c}) = \frac{1}{4}$

$\therefore P(\bar{c}/d_5) = \frac{1}{4} \times \frac{2}{3} \times \frac{2}{3} \times \frac{2}{3} \times (1-\frac{1}{3}) \times (1-\frac{1}{3})$

≈ 0.022

$\therefore \boxed{\bar{c}}$ is predicted ~~for~~ ds

→ How to improve relevance through feedback

- Objective is to provide relevant docs to user
- We want to improve relevance of docs ⇒ goal
- eg- Purchase implies high rating
  - if we click of retrieved result, like we click 4th docs out of 5 retrieved, then 1st 3 are not relevant, but 4 was. This is a kind of feedback
- we do query expansion.
- we want to provide better relevance to users.
- Improved relevance model, can be provide to same or other users, when same query was asked.
  ↳ all this is query expansion.

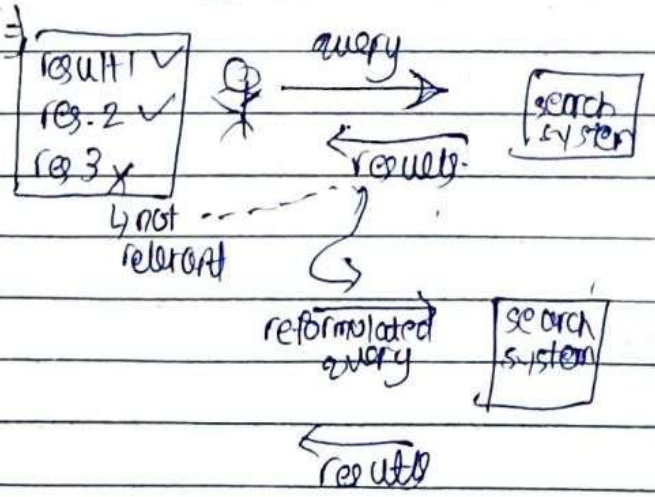
eg - we ask "plane" (synonyms as well as wrongly spelled ones are retrieved)
- results :-
  = flight
  = aircraft
  = by air
  = aeroplane
  = by air
  = fly
  = ~~fight~~ figt
  = aircft.

→ How to ensure good results

automatic query refinement

local
- use query or retrieved results to reformulate & improve query.
- for relevance
eg - pseudorelevance
- indirect relevance feedback.
- relevance feedback

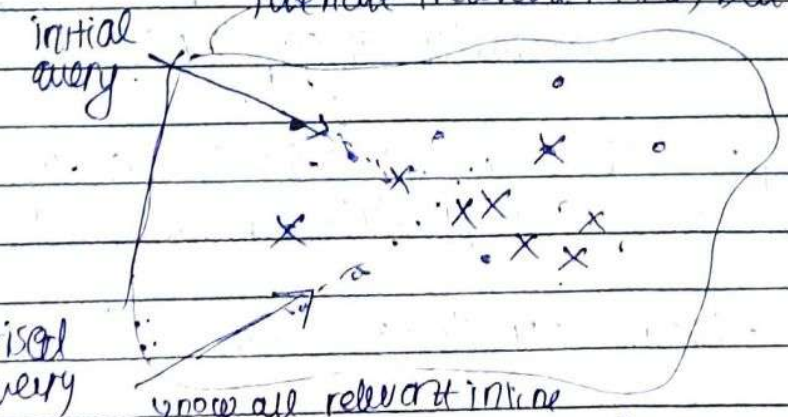
global
- we do not use query or results to improve & reformulate query for relevance
eg - thesaurus
- spelling correct^



Page No.
Date :

→ · Refmet takes explicit feedback from user.
· It asks if resul is relevant or not
· But its bad
· user doesn't have time & patience to provide explicit feedback
· But, still it is one of the way.
* · Indexed content is unknown to user
↳ we don't know if other relevant docs. are present but not shown
· "Inform^ need" changes after looking at the results.
↳
eg- we search for a specific song. But looking at song results, we change ~~to~~ our mind & search other song..
- so need change & its necessary to do query expansion..

→ Rocchio algorithm (Relevance feedback (local))
↳ we have relevant in line, but others are not



· can we shift our query from non relevant to relevant doc.? → do query expansion

- In practice we only know few relevant & non relevant.

$\vec{q_0}$ = initial query vector
D_r = set of relevant docs ⇒ ↑ its weight
D_{NR} = set of non rel. docs ⇒ ↓ its weight.

$$\vec{q_m} = \alpha \vec{q_0} + \beta \frac{1}{|D_r|} \sum_{d_j \in D_r} \vec{d_j} - \gamma \frac{1}{|D_{nr}|} \sum_{d_j \in D_{nr}} \vec{d_j}$$

- α, β, γ are weights.

- we represent query as weighted vector.
- use Rocchio formula to get new query vector.
- Now calculate cosine similarities
- Rocchio model increases recall
- +ve feedback is much more valuable than -ve feedback.
  (relevant)

eg- initial query = "cheap CDs cheap DVDs extremely cheap CDs"

relevant ← d_1 = "CDs cheap software cheap CDs"
not relevant ← d_2 = "cheap drills DVDs".

- Now construct term. freq. vector.

	Cheap	CDs	...
q_0	3	-	
d_1	2	-	
d_2	1	-	

- eg- to move (2,2) to (5,2), add 3+0x.

$q_{new} = 1 \times q_0 + 0.75 \times d_1 - 0.25 \times d_2$

- do this for each term.

eg- $q_{new} = 1 \times 3 + 0.75 \times 2 - 0.25 \times 1$

$= 4.25$

$\therefore q_{new} =$ | 4.25 | - | ... | - |

- we can have $q_{new} = -0.5$, this is wrong, so -ve values are put as 0.

$\therefore q_{new} = 0$

→ Pseudo Relevance (Blind) Relevance Feedback

- No user judgement
- we completely trust our web search engine
- we assume top k given by model are relevant to query
- now we can use rocchio algorithm to change the query.

→

problem

- search :" Dhoni abilities from 1st class cricket".
  - now, southafrica tests, etc. are also relevant hence retrieved.
  - so matrix includes southafrica, newzealand etc.
  - The ~~vec~~ query ~~w~~ vector will thus drift.
  - so, query ~~drift~~ drift problem occurs, as more weightage to terms which are not relevant.

→ Indirect (implicit) Relevance Feedback

- we do not ask explicit feedback.
- we consider clickstream mining
  - ↳ user clicks on nth doc, then top n-1 not relevant, & n is relevant
- query expansion would ~~be~~ be done based on implicit user preference (clicks)

→ In global approaches, we do not consider either query or retrieved results.

→

→ Global (user/ query independent)

- automatic thesaurus gener^n.
  eg - fast = rapid
  four = boundary } cricket
  sound = noise.
  tall = height
- how to improve relevance?
- slangs?
  ↳ brother ≈ bro
- we can do co-occurrence analysis for the doc.

eg - mainframes (doc) — mainframe (doc)

primarily, we are .... — usually, we refer

primarily & usually are used like synonyms.

eg - term incident matrix :-

words / books	Book 1	B 2	B3	B4
w1	100	1	110	2
w2	50	9	47	8
w3	25	3	24	4
w4	75	6	73	7

- Book 1 & 3, Book 2 & 4 are similar.

- So, we can do co-occurence analysis.
- 2 terms are similar if term vectors are similar.
- Context is important
  - ↳ same video may not be relevant on weekend, with but relevant on week.

eg - term-doc matrix

term-doc matrix . term doc matric. (Transpose)
= term-term matrix.

from term-term matrix, we can see results :-

	t1	t2	t3	t4	t5	t6
t1	3	1	2	1	2	3
t2	1					1
t3	2					2
t4	1					1
t5	2					2
t6	3	1	2	1	2	3

similar

so, t1 ≃ t6

→ Precision & Recall

- Precision - fract^n of retrieved docs. are relevant.

$$P\left(\frac{\text{relevant}}{\text{retrieved}}\right)$$

- Recall - fract^n of relevant docs that are retrieved

$$P\left(\frac{\text{retrieved}}{\text{relevant}}\right)$$

	relevant (→ true +ve)	non relevant
retrieved	tp	fp (→ false +ve)
not retrieved	fn (→ false -ve)	tn (→ true -ve)

↳ confusion matrix

- Precision P = $\frac{tp}{tp+fp}$ → correct predict^n / ↳ are retrieved docs.

- Recall = $\frac{tp}{tp+fn}$ → correctly predicted / ↳ total relevant in corpus

- There has to be balance b/w precision & recall
- If we try to increase one, other may decrease
  eg - we retrieve more docs for more recall, but precis^n may decrease, & vice versa
- Diff. b/w recall & precision must not be more than 5%.
  eg - if precis^n = 95%, recall = 10%, then diff. > 5%

- For recommender systems,

	item1	i2	i3	...	in
User1					
U2					
U3					
⋮					
Un					

→ we try to predict each cell rating.

- Then we calculate error by Σ(actual rating − predicted rating)^2 (Root mean square error)
- This is our accuracy parameters.

eg- IRS retrieved 20 docs.
- There are 100 relevant in corpus.
8 are correct & relevant

∴ Precision = $\frac{8}{20}$

Recall = $\frac{8}{100}$

eg - Retrieved docs → RRNNNNNNRNRNNNR NNNNR

precision = $\frac{6}{20}$

- let 8 relevant docs (total)

∴ recall = $\frac{6}{8}$

Page No.
Date :

→ we use HM as we want both precision & recall to go up for F1 measure to go up

→ FMeasure (F1Score)

- Takes both precision & recall into account

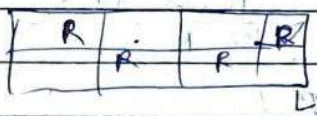
$$\therefore F = \frac{2PR}{P+R} = \frac{2}{\frac{1}{R}+\frac{1}{P}}$$

Q- Why F measures take harmonic mean? (not arith. or geometric)?

→ more conservative

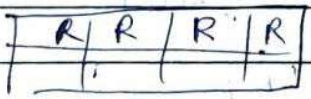
A- arithmetic - biased towards high values (tail)

HW geometric mean - ? - gives consistent result for normalized values.

→ eg - 

case 1

↳ here ranking is not good



case 2

↳ Both have same precision & recall, but case 2 is better, as 1 to 8 are ranked, and case 2 correctly retrieves ranking wise.

here the outputs docs; the retrieved ones are at 1-4, higher ranking. relevant

→ solution - precision at K-th index.

→

0 retrieved
→ 1 cell

P@1 = $\frac{0}{1}$ = 0 [ ] ← At 1st index / 1 location

P@2 = $\frac{1}{2}$ [| R] ← At 2nd index

P@3 = $\frac{2}{3}$ [| R | R]

P@4 = $\frac{2}{4}$ [| R | R |] → adding more & more docs only reduce relevant values

· similarly if total 100 relevant docs, recall@1 = 0/100, recall@2 = 1/100 etc.

Disadv. - If only 4 total relevant docs, & if we retrieve 10 docs, max precisⁿ only 0.4.

→ Interpolated precision

cutoff at kth relevance level.

P@1 = $\frac{1}{2}$ [| R] ← 1st R
(1st locatⁿ where R appears.)

P@2 = $\frac{2}{3}$ [| R | R] ← 2nd R

↳ eg- if total 5 relevant, then 5 levels. 1st locatⁿ = where first R appears, similarly 2nd locⁿ where 2nd R appears & likewise.

interpol. avg. precisⁿ = $\frac{0.5 + 0.66}{2}$ ↳ if we are only interested in 2 levels of precisⁿ

· If multiple queries are given we calc. average MAP the queries

☆ case 1 → To calculate avg precisⁿ

[| R | | R | R | | R | | R]

avg-precisⁿ = $\frac{\frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2}}{5}$

eg - [ ] [R] [ ] [R] [ ] [ ] [R] (level 2, level 3)

∴ AP at level 4 : $\frac{\frac{1}{3} + \frac{1}{3} + \frac{1}{3} + 0}{4}$ (level 1 ✓, no level 4)

AP = interpolated AP

∴ If 10 docs,

$$avg = \frac{\frac{1}{2} + \frac{1}{2} + \ldots \text{10 times}}{10}$$

case 2 → For Avg. precis^n (AP) at level 4,

$$\frac{P@1 + P@2 + P@3 + P@4}{4}$$

→ MAP (mean avg. precision) is important - calculates precis^n for all queries for all relevance levels

query -

eg - 1) [R] [ ] [R] [ ] [ ] [R] [ ] [ ] [R] [R] → 5 levels of relevance

$$\therefore \frac{P@1 + P@2 + P@3 + P@4 + P@5}{5}$$

$$= \frac{\frac{1}{1} + \frac{2}{3} + \frac{3}{6} + \frac{4}{9} + \frac{5}{10}}{5}$$

$$= \frac{1 + 0.66 + 0.5 + 0.5 + 0.44}{5} = 0.62$$

query-2) [ ] [R] [ ] [ ] [R] [ ] [R] [ ] [ ] [ ] → 3 levels of relevance

$$\frac{P@1 + P@2 + P@3}{3}$$

$$= \frac{\frac{1}{2} + \frac{2}{5} + \frac{3}{7}}{3} = \frac{0.5 + 0.4 + 0.428}{3} = 0.44$$

· If multiple queries are given we can average MAP of the 2 queries

ex / location

Index

more & more only relevant reduce values

relevant docs, ... 0.4

eg - if total 5 relevant, then 5 levels.

... location = where

2nd ... precise.

∴ final MAP = $\frac{0.62+0.44}{2} = \boxed{0.53}$

Q = case 1: P@1

Q = query 1 =>

→ Also,

$$\text{Precision} = \frac{[\{\text{relevant docs.}\} \cap \{\text{retrieved docs}\}]}{[\{\text{retrieved docs}\}]}$$

$$\text{recall} = \frac{[\{\text{relevant docs}\} \cap \{\text{retrieved docs}\}]}{[\{\text{relevant docs.}\}]}$$

(MAP)

→ Mean avg. precisⁿ is the average of average precisⁿ of all queries.

eg. q1 = | R | R | R | R | R |

q2 = | | R | | R | | R |

q3 = | | R | | R | | R |

AP for q1 = $\frac{\overbrace{1+\frac{1}{2}+\frac{1}{2}+\frac{1}{2}+\frac{1}{2}}^{\text{level 1}}}{5} = \frac{1}{2}$ (level 5)

AP for q2 = $\frac{P@\text{level 1} + P@\text{lv 2} + P@\text{lv 3}}{3}$

$= \frac{\frac{1}{3}+\frac{2}{6}+\frac{3}{9}}{3} = \frac{1}{3}$

AP for $q_3 = \frac{\frac{1}{3} + \frac{2}{6} + \frac{3}{9}}{3} = \frac{1}{3}$

MAP = $\frac{\text{AP of } q_1 + \text{AP of } q_2 + \text{AP of } q_3}{3}$

$= \frac{\frac{1}{2} + \frac{1}{3} + \frac{1}{3}}{3}$

$\Rightarrow \therefore MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk})$

★ Problems - we cannot compute MAP if we do not know the total no. of relevant results for any query.
- This also happens in webpage

→

- For recommender RMSE
- For webpage - MAP (meaning. precision)

→ Crawling

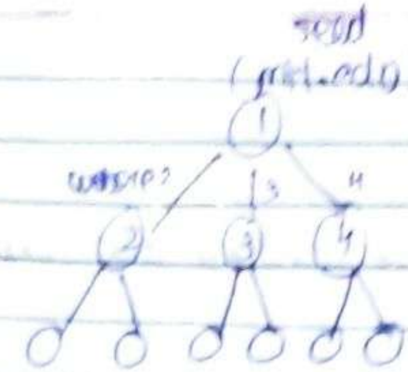
⇒ Role of crawling / crawler

- We discuss IRS where corpus is available.
- But, we need a proper corpus
- So, crawler is responsible to create proper corpus.
- Every second, multiple webpages are added on internet
- So, corpus must be ~~upd~~ up to date.
- It is also called "spider" / "robot"

Basic crawler opn

- Begin with known "seed" URLs
  eg- cisco.com, MIT.edu (authoritative) needed to fetch other docs.
  ↳ initial known pages
  ↳ some authoritative pages
  ↳ some websites we trust.
- Fetch & parse them
  - extract URL they point to
  - place the extract URLs on a queue
- Fetch each URL on the queue & repeat.

eg- we have MIT.edu. → seed URL
- If we have this URL, then crawler extracts links to other external websites (this is parsing).
- From those webpages, again parse & retrieve docs.
- We can use queue.

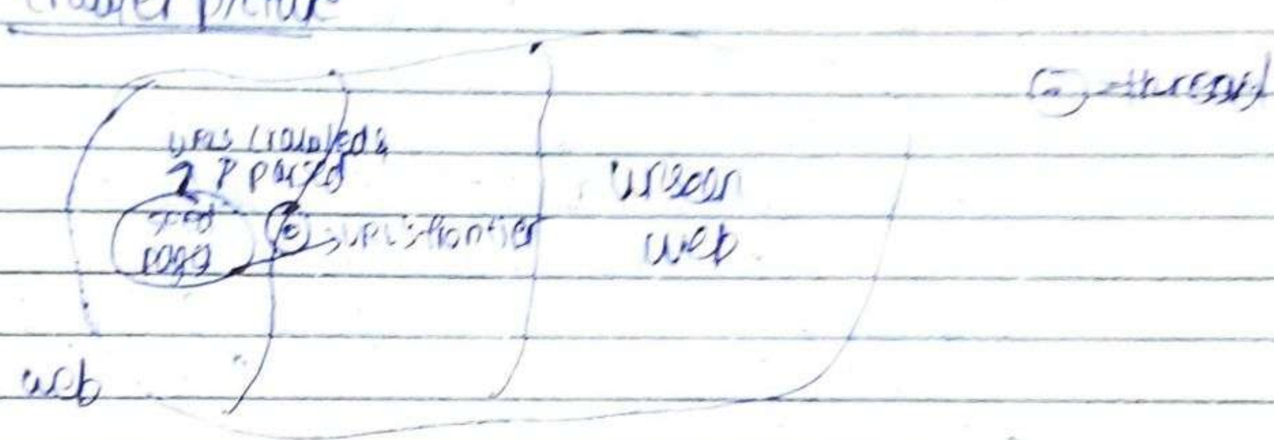


(BFS)

- we can use breadth first search (queue can be used)
- so, this method of crawling can be a soln

- so, crawler may or may not be req depending on IPs
- But its req. for website.

crawler picture



URL frontier - set of URL's not yet parsed

eg queue:- [1|2|3|4|5|6]

URL frontier

crawled & parsed

- But not polite, as till 3 not parse, we keep on hitting it

→ Problems / complic's

- Not feasible with 1 machine.
  - distributed architecture needed
    ↳ more processing

→

- malicious pages
  - spam pages
    - title of page & content different
  - Spider trap
    - Start with seed, then next page then again new page then again bad page → spam
    - efforts of crawlers are wasted
- So, we need to deal with them.
- Even non malicious pages pose challenge.
  - Latency / bandwidth to remote servers vary
    - eg- busy server
      - gives quick reply, other busy
  - webmaster stipul's
    - manager of website
    - constraints for how deep can crawler crawl the content of website,
    - rules in robot.txt.
    - ★ crawler needs to ~~fool~~ follow this protocol & rules / restrictions.
    - crawler gets limited access.
      - eg- only educational crawler can crawl mit.edu.
        - so, only some pages allowed to be crawled.

eg - no robot should should visit any URL starting with "/yoursite/temp". except robot called "search engine"

- user-agent :- *
  disallow : "yoursite/temp"
  allow : "searchengine"

- other problems :-
  - site mirrors & duplicate pages
    ↳ visiting same/similar pages

- politeness - crawler must not bombard the server, don't hit a server too often.
  eg- if nirmauni.ac.in doesn't give reply, wait for sometime then send.

→ What crawler must do

- Be polite, even if explicit politeness (in robot.txt) not mentioned - implicit politeness
- only crawl allowed pages.
- respect robot.txt.
- Be robust & handle spider traps.
- Be capable of distributed operⁿ, or multi-threaded environment.
- Be scalable, to ↑ crawl rate by adding more machines

→ all threads & machines should be busy.

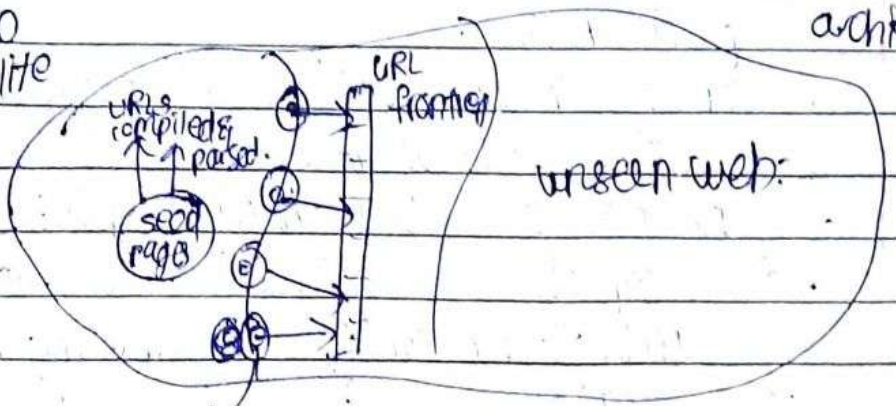
- Perf. / efficiency - permit full use of available processing & network resources

scalable - [← webpage →] even if more breadth of webpages, even then crawler must must not ↓ speed (↳ crawl rate)

- Fetch higher quality pages first.
- continuous operation - Continuous continue fetching fresh copy of previously fetched page
  ↳ eg- news pages should be crawled more often compared to nirmauni.ac.in (priority of fetch rate)
- Extensible - adapt to new data formats / protocols.

→ updated crawling picture → we have seen discuss., so we need multi-threaded architecture

- We want to prevent impolite ness



* all threads must be busy

crawling thread

→ URL frontier

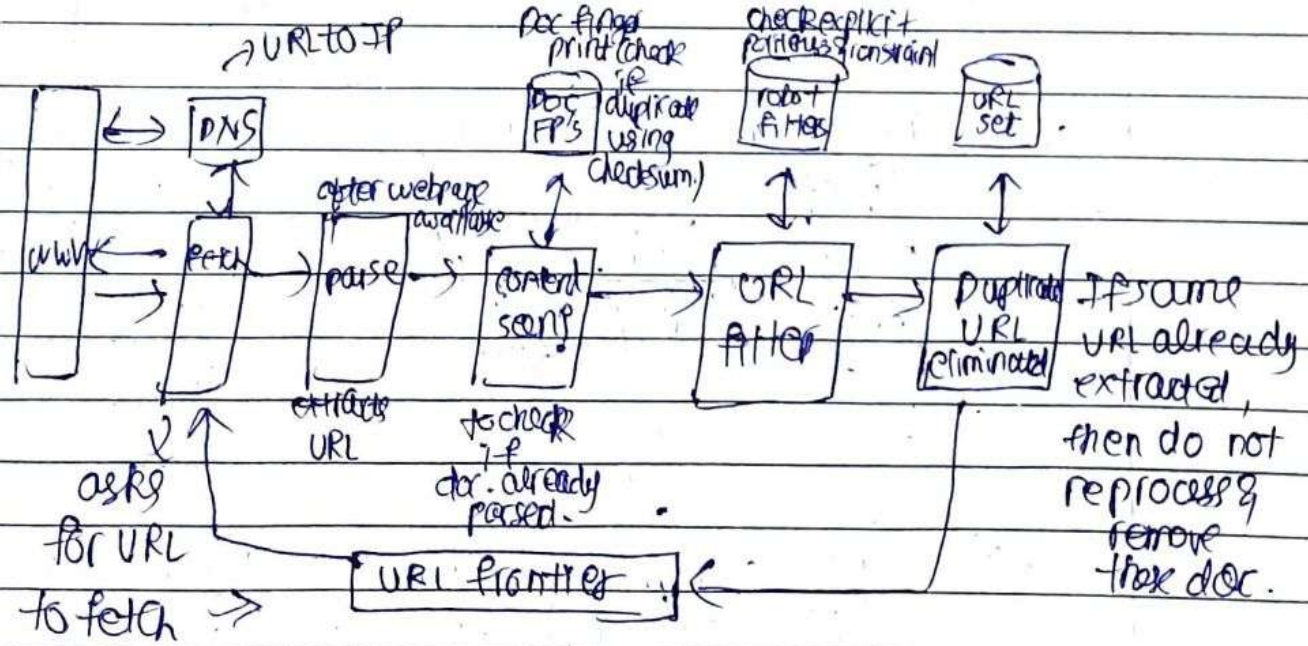
- set of URL yet to be parsed
- can included multiple pages from same host.
- must avoid trying to fetch them at same time.

→ Implicit & explicit politeness

- Explicit → use robot.txt.
  implicit → understand on own not to hit server too much.

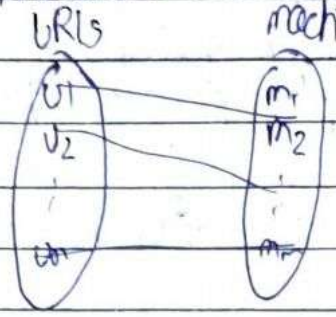
→ Basic crawler architecture

- If we have 1 machine, then all in 1 machine
- If 20 machines, then architecture distributed & run simult. on all machines.



- Deals with explicit but not implicit politeness.

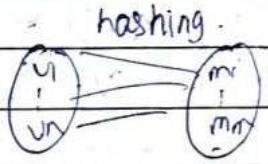
- We can use concept of hashing. ~~URL~~ URL to machine.



} URL mapped to machine.

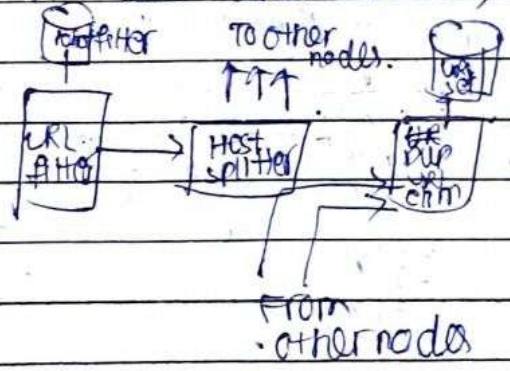
- Suppose the ~~come~~ m_1 comes across u_2, but its not responsible for it, then it forwards the request to the responsible machine.

→ • To achieve distributed archi., server must be divided
so, parsing done by particular machine
• We use hashing to map URL's to machine.



• Nodes are our machines which are geographically distributed
• We have a URL frontier for each node, which sends it to URL elim. of appropriate node

so, in our architecture,



- If m_1 comes to know u_1 is not its responsibility, it forwards u_1 to m_i

→ simple priority queue fails as:- many links out of a page go to its own site creating a burst of access to that site
eg cisco.com may be links going to same cisco server

URL frontier considers

2 main measures

- politeness - do not hit a server too frequently
- freshness - some pages like ~~news~~ news, are ~~crawler~~ crawled more often than others. ↳ we can crawl this page based on refresh rate of page. If refresh rate ↑, then crawl more often.

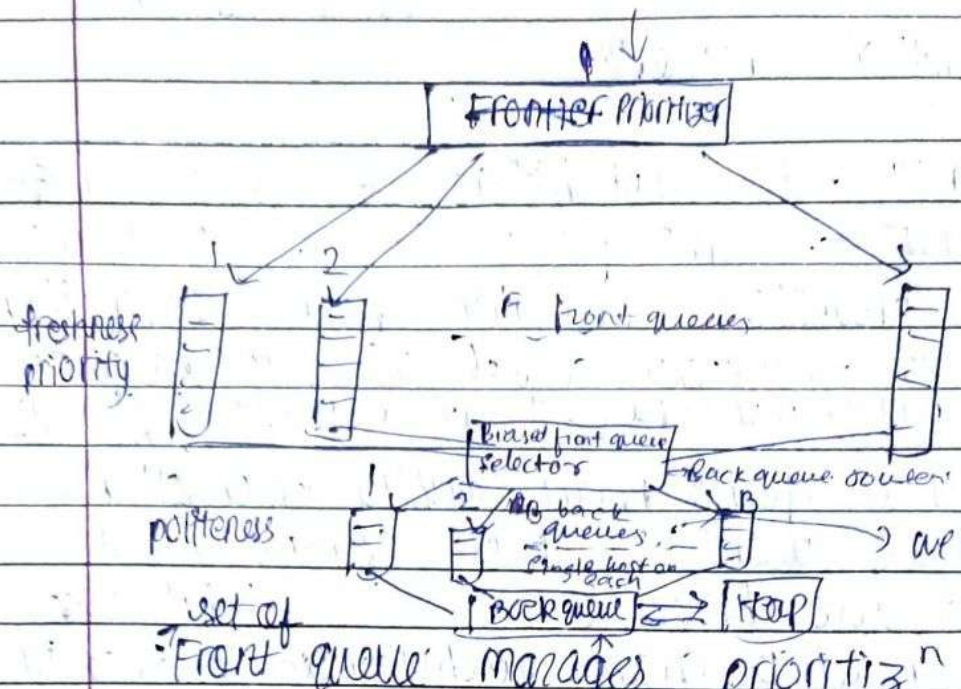
- But these goals may conflict each other.
  - As we want to keep threads busy, and also be polite ↳ hit servers.
  - so we need to change architecture of frontier. (simple queue won't work)
  - eg- on educational websites, there are many links on 1 page (eg- Nirmauni Law Engg. Pharm etc.)
    ↳ so crawler hits Nirmauni too often.

→ • even if we restrict only 1 thread to fetch from a host, can hit it repeatedly
• Heuristic- if server replies after 1ms, multiply it by 10, ∴ after 10 ms send request again.
- insert time gap b/w successive req. ~~from~~ to a host >> time for server for most recent reply time / fetch time
↳ usually, we multiply by 10 times

→

✱ self notes →

Page No.
Date:



- Front queue manages prioritizn
- Back manages politeness

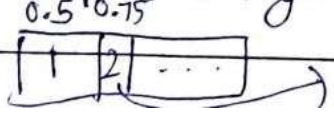
↳ we can have set of URL from diff web servers.

↳ set of to crawl with freshness (based on refresh rate)

→ Mercator URL frontier

- URL flow from top to frontier.
- prioritizer assign URL integer priority b/w 1 to k. (all queues are FIFO).
- Then it appends priority to corres queue.
- Heuristic to assign priority - refresh rate can be checked from previous crawls (eg - news have high refresh rate)

1) we can select prioritized queue directly using its priority assigned or
2) we give probability



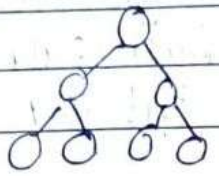
→ generate random no. & choose priority for URL to one of queue

Now for back queues :-

- For each URL we have a set of timestamp
- We use minheap (for selecting minimum timestamp)

Eg - lets say timestamp is 10s, so, we can send after that it is 20s

- so, root node gets changed.



- we req. at 10:25, we get response at 10:35. So, we delete 10:25 by some other (lets say 10:26), & the 10:35 is inserted in heap.

- so, back queues help in politeness.

→ when a back queue req. a URL, ask front queue to give URLs.

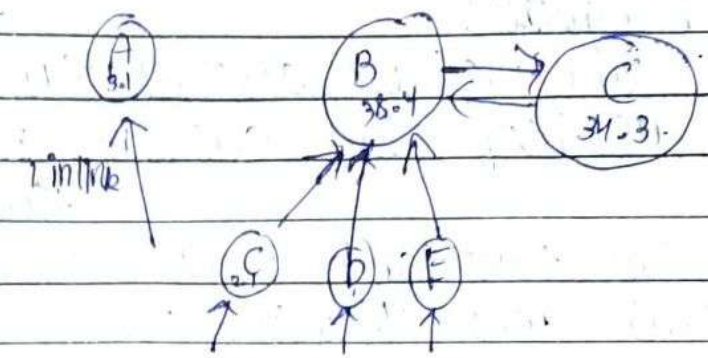
→ How to check if page is authoritative?

use page rank - assigning rank to page.

- we have - outlink - how many websites on our page (easy-to-fake)
- inlink - how many websites reach our page. (more reliable as it is difficult to fake)

Page No.
Date:

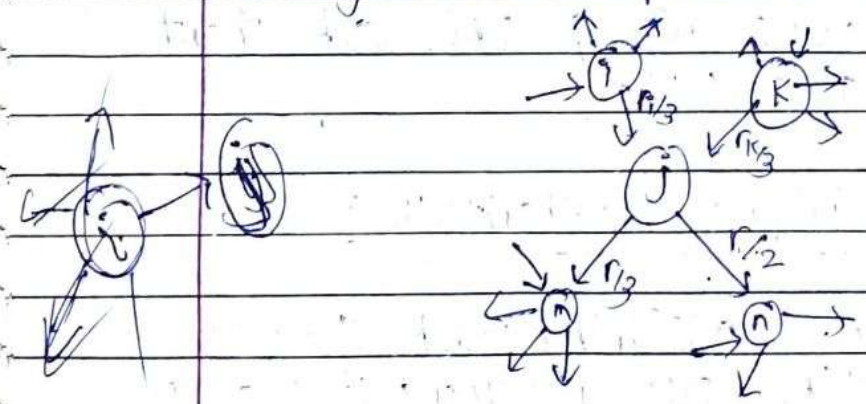
eg -



- so, B has many inlinks, so more also
- Also, C has high score, as B is authoritative, and gives inlink on C.

⇒ - each link's vote is ∝ to importance of its source page.

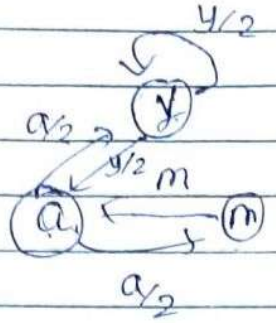
- If page j has importance r_j, has n outlinks, each links gets $\frac{r_j}{n}$ votes
- j's own importance is sum of its inlinks



$$j = \frac{r_i}{3} + \frac{r_K}{3} + \frac{r_m}{3} + \frac{r_n}{2}$$

- If ~~page is~~ vote from page is more important if it is pointed to by other pages.

rank r_j for $j \Rightarrow \therefore r_j = \sum_{i \to j} \frac{r_i}{d_i}$ } for each inlink.



· importance of $r_y = 2$ (2 inlinks):

$\therefore r_y = \frac{r_y}{2} + \frac{r_a}{2} + 0$

$\frac{r_y}{2}$ → only for inlinks. for y; 2 outlinks

$\frac{r_a}{2}$ → 2 outlinks from a

0 → no relⁿ btw m

$r_a = \frac{r_y}{2} + r_m$

$r_m = \frac{r_a}{2}$ → no relⁿ with y.

· 3 eqn's, 3 unknowns. (no unique soln, as no constants) so no scale factor; now unique const.

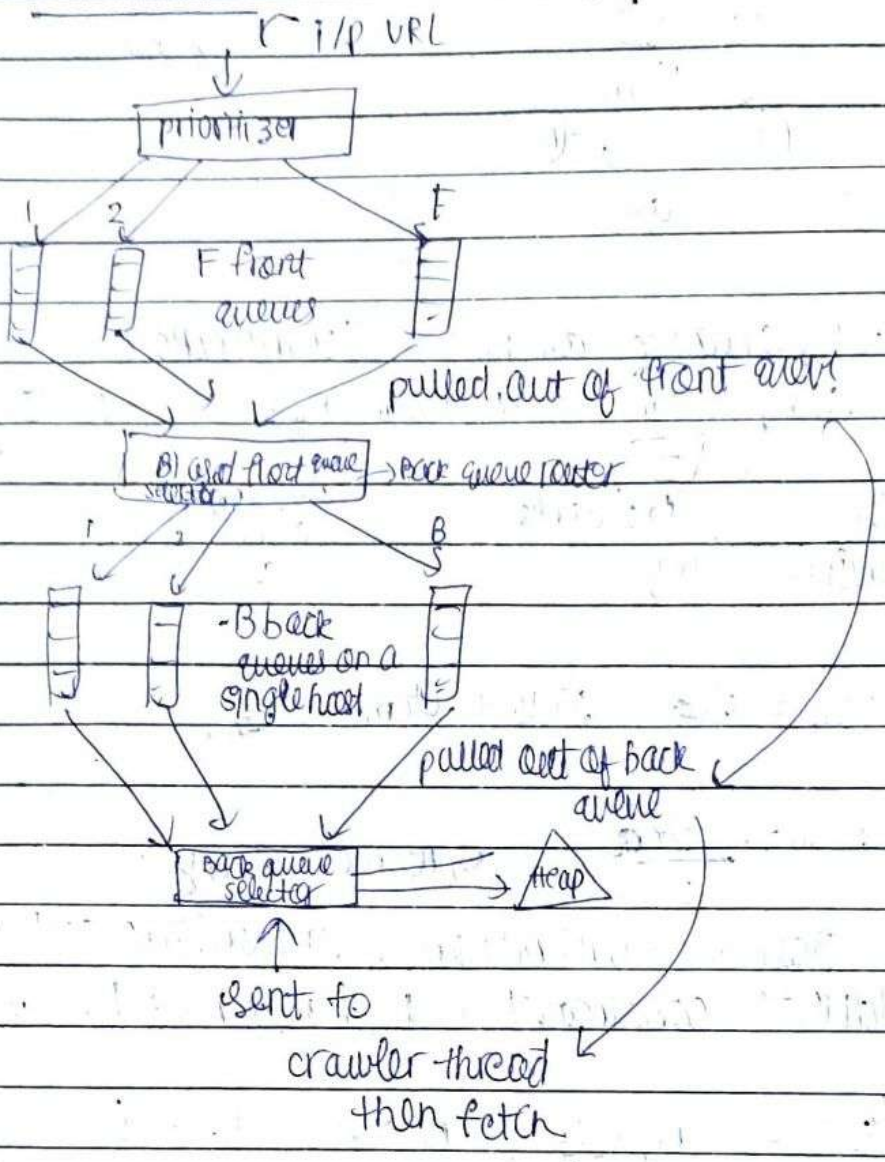
** additional constraint ⇒ $r_y + r_a + r_m = 1$.

soln ⇒ $r_y = \frac{2}{5}$, $r_a = \frac{2}{5}$, $r_m = \frac{1}{5}$

· we can use Gaussian elimⁿ to solve, but for bigger values, we can use matrix formulaⁿ.

Page No.
Date :

Self → URL frontier

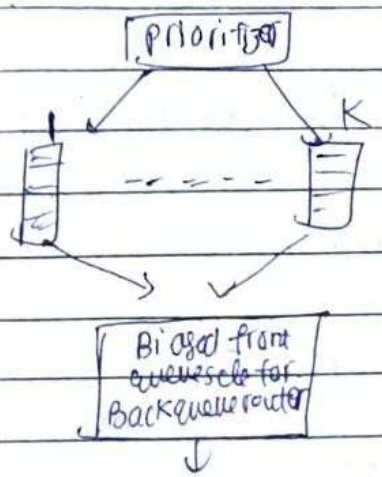


URL's
↓
prioritizer
↓↓↓↓↓↓
K front queues
↓↓↓↓↓↓
Biased front queue selector. Back queue router
↓↓↓↓↓↓
B back queues single host on each
↓↓↓↓↓↓
Back queue selector
↓
crawl thread req. URL - then fetch

} URL frontier mercator scheme.

- All this is mercator architecture
  ↳ old classical crawler.
- Front queue prioritize based on refresh rate
  (eg - news crawled more)

For front queue,



- There are K front queues, and URL incoming is assigned priority from 1 for Priority 1 need more freq. crawling than others (eg - news?)
- Prl. K need less freq. crawling
- So, we append URL to corres. queue.
- To assign priority,
  heuristic :- refresh rate sampled from previous crawls

For back queue

- How to pull URL's from front queues?
- We may pull in a round robin way (first queue F1, F2, .. FK). But its wrong, as we need to prioritize lower no. of queues
- So we need more URLs from queue 1 than queue K.

→

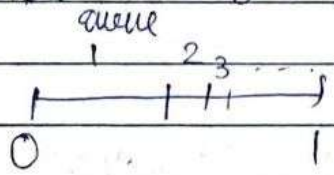
There are 2 ways to pull from front queues:

1) → First pull an item queue 1.
- Then pull next time, pull an item from queue 1 & 2
- Then 1, 2, 3

1
1 2
1 2 3

- we can see than 1's URL will be pulled more often, then 2, then 3 ...
- So, its kind of round robin.

2) → we assign prob. to each queue
- eg- if random no. b/w 0 to 0.5, pick URL from queue 1
- If 0.5 to 0.75, pick from queue 2, & so on

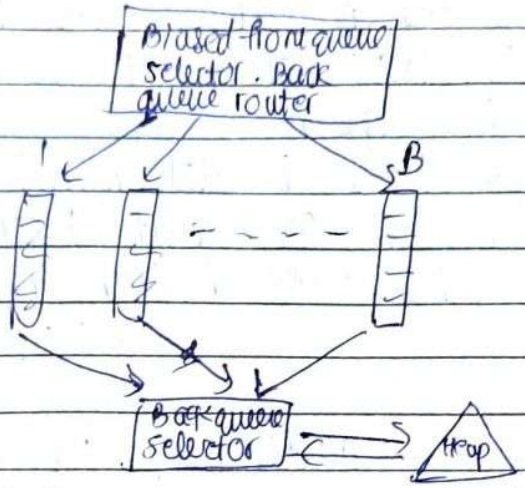


- so, 1 has larger prob. of queue being pulled & thus algo. is biased for more priority queue.

→ - So we have solved freshness problem by front queues
- URL going in are in any order, but coming out based on priority.

→

- Now, for the back queues,
- URL pulled out will go in a back queue from 1 to B.
- In a front queue, it is possible for it to have URL's of diff servers.
  - eg as CNN.com & BBC.com have same priority, & so their links may be on same front queue.



- each back queue has links only from 1 server
  - eg - queue 1 has only links from cisco server
    2 has only from stanford server, etc.
- we maintain a min-heap.
  - heap has B vertices, each vertex corres. to each back queue (or each server)
- The value of the key: threshold time
  - time before which you can't hit server.

eg - vertex (10:15 PM cisco) → we can't send query to cisco before 10:15 PM.

- SO, root has least value of timestamp
- SO, first we send query to root URL.

- So, we extract ~~URL~~ root URL
- After that, suppose removed URL was of queue 2, we go to back queue2 give it to fetcher.
- So, in back queue -
  - one entry for each back queue
  - The entry at earliest time t_e at which the host corres. to back queue can be hit again.
  - The earliest time is determined from
    - ⇒ last access to that host.
    - ⇒ any other time buffer heuristic we choose.

* - suppose we pulled all URL's from back queue1, & it is empty (server of cisco)
  - We ask front queue to give a URL:
  - If it is stanford website, we put it in stanford queue.
  - we again ask for URL, till we get atleast 1 URL of cisco, so that atleast 1 URL in back queue 1.

- Now there are millions of server, so we require millions of back queues, which is not feasible.
- So, once a queue gets empty, we relabel it from cisco to mit.edu.
  - ↳ some other server queue.

- Back queue processing :-
  - A crawl thread seeking to crawl
  - Extract root of the heap
  - Fetches URL at head of corres. back queue

  - Checks if queue q is now ~~emt~~ empty, if so, pulls a URL v from front queues
    - if there's already a back queue for v's host, append v to q & pull another URL from front queues, repeat
    - else add v to q
  - When q is non empty create a heap entry for it
- So when ~~A~~ root removed we again take new URL from the corres. back queue, calculate new timestamp & reinsert it in heap (now it could be anywhere in the heap)

- We maintain a table from hosts to back queues

Host name	Back queue
cisco	3
mit	2
⋮	⋮

- each back queue is kept non empty while crawl is in progress.
- each back queue has URL only from 1 host

number of back queues B :-
- Keep all threads busy while respecting politeness.
- Mercator heuristic - 3 times as many back queues as crawler threads.

Class :-

- For bigger graphs, we can represent using adjacency matrix.

eg- if $i \to j$ then $M_{ji} = \frac{1}{d_i}$ else. $M_{ji} = 0$

d_i → out degree

matrix representation.

M is column stochastic
↳ sum of all values in col. = 1.

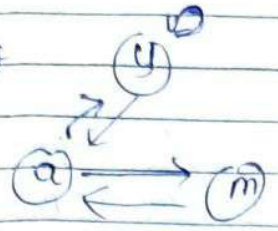
- Rank vector r → vector with an entry per page
$\sum r_i = 1$ (r_i is importance of i^{th} page).
$r = M \cdot r$.

→ now, $r = M \cdot r$ can be mapped like eigenvector.

Suppose. $1 \cdot r = M \cdot r$

similar to $Ax = \lambda x$

∴ r is eigenvector of Matrix M.

eg 

	y	a	m
y	1/2	1/2	0
a	1/2	0	1
m	0	1/2	0

→ matrix M

(for link i→j, we fill mji)

↳ basically we fill inverted

- we need to assume initial r vector..

set $r_j = \frac{1}{N}$ ↳ no. of nodes. (here = 3).

⇒ power iter^ method

initially set ⇒ $r = \frac{1}{N}$

$\therefore r = [1/3 \ 1/3 \ 1/3]$ $\begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$ ↳ 3 nodes so (3×1) dimens^.

next iter^ → $r = M \cdot r$

- keep doing this this r and r of next iter^ become same. (convergence).

∴ eg - $r = \begin{bmatrix} 1/2 & 1/3 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$

M r

$= \begin{bmatrix} 1/3 \\ 1/2 \\ 1/6 \end{bmatrix}$ ← new r.

- now again
  $r = M.r$
- keep doing this till new value of r & previous value of r almost same (or exactly same) ↳ within specified threshold

:

let final r = after converged. $\begin{bmatrix} 6/15 \\ 6/15 \\ 3/15 \end{bmatrix}$ → score of page y, page a, page m } these r give our score/rank

- so we solved a big problem using power iter^n without gaussian elimin^n. or flow model.

using Hubs & authority →

- we assumed that only 1 score is assigned to a page.
- we use hubs & authority algo. for this.

eg- we have some newspapers ↳ hubpage

- these papers have ~~of~~ refer other good papers → authoritative page.

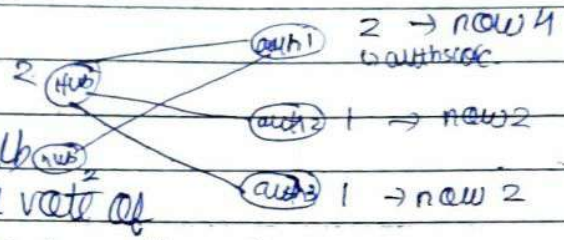
∴ hubs give outlinks to auth. pages
auth. page have inlinks., & useful inform^n

- 2 ways to check quality
  - quality as an expert → Hub
  - quality as a content provider → authority
    ↳ ~~pow~~ how good is content.

⇒ Auth. page have useful info. (usually).
- Hubs ~~He~~ have list of all auth pages (eg - newspapers)

- To count & score,

- If there is a link from hub to auth. page, then we count it as a unit.



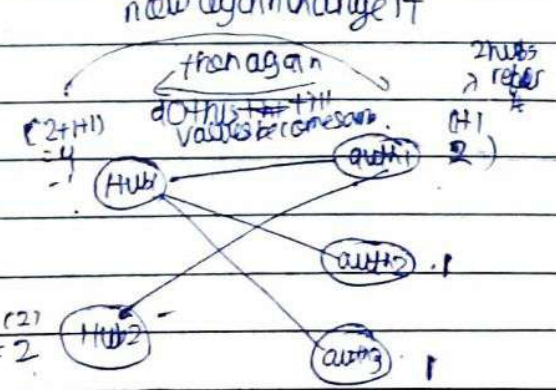
- this hub gives a vote of 1 to each auth. page.
- auth. page1 has score of 2 (2 hub links).
- ~~A~~ using score of auth. page we can calculate hub score.
- we add the sum of score of all auth. pages being refered by a hub, this gives the hubscore
- we keep calculating scores (hub & auth.) till convergence.

of that hub.

⇒ This HITS algorithm

~~Hyper~~

now again change it



IRS (contd.)

- Every page has 2 score → hub score
  authority score

- A good hub links to many good auth. & good auth. linked to good hubs.
- We have self reinforcing recursive fund".
- Represented as h & a
  h → hub score
  a → auth. score

→ HITS - Hyperlink induced Topic Selectn

Initialize ⇒ $a_j^{(0)} = \frac{1}{\sqrt{n}}$, $b_j^{(0)} = \frac{1}{\sqrt{n}}$

- keep iterating till convergence

$\forall j$ auth = $a_j^{t+1} = \sum_{j \to i} h_j^t$

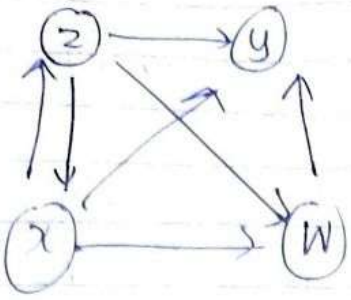
$\forall i$ hub = $h_i^{t+1} = \sum_{i \to j} a_j^t$

$\forall i$: normalize :-

$$\sum_i (a_i^{(t+1)})^2 = 1 \quad \Sigma (h_j^{t+1})^2 = 1$$

→

eg -



$H(z) = H(y) = H(x) = H(y) = 1$

$a(z) = H(x) = 1$
$a(y) = H(z) + H(x) + H(w) = 3$
$a(x) = H(z) = 1$
$a(w) = H(x) + H(z) = 2$

total $a = 1+3+1+2 = 7$
∴ normalized :- $a(z) = \frac{1}{7}$, $a(y) = \frac{3}{7}$,

$a(x) = \frac{1}{7}$, $a(w) = \frac{2}{7}$.

now, $H(z) = a(x) + a(w) + a(y)$ (3 outlinks)
$= \frac{1}{7} + \frac{2}{7} + \frac{3}{7} = \frac{6}{7} = 0.858$

$H(y) = 0 \rightarrow$ no outlinks
$H(x) = a(z) + a(y) + a(w)$
$= 0.858$
$H(w) = a(y) = \frac{3}{7} = 0.429$

~~normalized~~ = total $H = H(z) + H(y) + H(x) + H(w)$
$= 2.145$

∴ normalized :- $H(z) = \frac{0.858}{2.145}$

$H(y) = 0$

$H(x) = \frac{0.858}{2.145}$

$H(w) = \frac{0.429}{2.145}$

⇒ we repeat this, till values become nearly same. (convergence)
. This is to make good hub & auth. pages.
. This is HITS algorithm.

→ Need of Recommender system

- If user knows what he wants, then he directly searches.
- He can also look & choose
- But what if more choices? → more confusion.
- so, to deal with this, we need recommender system
  ↳ type of ~~Eer~~ IRS
- Provides personalized service to user.

⇒ · Earlier we had retail store
  ↳ limited shelf-space.
  also TV → only 1 channel at a time
  movie → one at a time.
  ⇒ so, we have limited choices

⇒ Now web enables near 0 cost dissemin^n (action of spreading info) of info about products
↳ scarcity to abundance.

⇒ So, many choices require better filter
↳ recommendk engine.

eg- one book of ~~author~~ author became famous due to other book
↳ which became famous.

- as book A becomes famous, book B of same ~~at~~ author gets recommended, so, B become more famous even though it was published earlier, but got famous now.

→ ⇒ Non personalized recommender system
↳ eg- recomm. restaurants based on avg. rating

- editorial & hand curated
  - list of favourites
  - List of essential items
- simple aggregates
  - top 10, most popular, recent buys → take avg. ratings.
- Tailored to indiv. users
  - amazon, netflix etc.

→ Formal model.

Let C = set of customers
Let S = set of items.

U: C × S → R
↓ utility matrix
R = set of ratings, & R is totally ordered set.
eg - 0 to 5 stars, 0 means bad, 5 is best.
↳ ordered

eg - utility matrix → user v/s item

	Movie 1	2	3	4
person 1	1		0.2	
2		0.5		0.3
3	0.2		1	
4				0.4

- A person may or may not rate a movie
- so, matrix can be sparse.
- we predict the ratings which are missing, then we recommend.

→ Key problems

1). (Dataset collect^) ← gathering known ratings.
2). Extrapolate unknown from known ratings.
↳ we are interested here
3). evaluating extrapol^ method → how to measure success / rating

★ - we are not interested in what we don't like
only interested in what we like - may or may not be true.

1) gathering ratings

- explicit
  - ask people to rate
  - many users don't like this
- implicit
  - if user purchases then high rating
  - learn from user actions.
  - what about low rating ? → issue.

2) Extrapolating utilities.

- key problem :-
  - most people don't rate
  - so sparse matrix (U matrix)
  - cold start problem
    - ↳ new items have no rating
    - ↳ new users have no history.

⇒ 3 approaches to recommender system → to find value :-

1) content based — works based on similar attributes.
2) collaborative
3) Latent Factor based.
   - ↳ similar to SVD : matrix factorizn.

- content based → · we watch a movie, our profile is prepared based on attributes of movie. If new movie's feature match with our matched movie, recommend new movie.
  - problem :- overspecialization, only similar movies recommended.

- collaborative filtering
  - suppose user X
  - set of N other users have similar choices to X.
  - & we check ratings of ~~N users X user~~ N users ~~to predict based on pred ratings of N other users.~~ to predict rating of X user.

eg- set N = $\underset{N_1}{3}, \underset{N_2}{4}$ ratings for X, rating can be $= \frac{3+4}{2}$

$= 3.5$,

given X is similar to N users. (here N=2)

- to find if X is similar, we use different techniques.

→

	movie1	m2	m3	m4	m5	m6
A	4			5	1	
B	5	5	4			
C			2	4	5	
D		3		5		3

- Consider users x & y with rating vectors r_x & r_y
- We see, that A & B are not similar universally.

eg. similarity (A, B) = $|r_A \cap r_B| / |r_A \cup r_B|$

Jaccard. = $\frac{1}{5}$ → 1 common → m_1

m_1, m_2, m_3, m_4, m_5 → total in A & B.

$sim(A, C) = \frac{2}{4}$ → 2 common → m_4, m_5

→ total 4 → m_1, m_3, m_4, m_5

- Jaccard only sees common items.
- So, its a problem. →

⇒ So, we can use cosine similarity.

$sim(A, B) = \cos(r_A, r_B)$ → (4,0,0,5,1,0) → (5,5,4,0,0,0)

→ considering missing values as 0.

$sim(A, B) = 0.38$, $sim(A, C) = 0.32$

$sim(A, B) > sim(A, C)$, but not by much.

- Now 0.38 & 0.32 are nearly same problem - This happened as we considered missing values as 0 → bad rating! → just because we don't rate don't mean we dislike

- Instead, we can replace missing ratings by average rating.
- This is called ~~centered~~ centered cosine similarity.