What Is Information Retrieval?

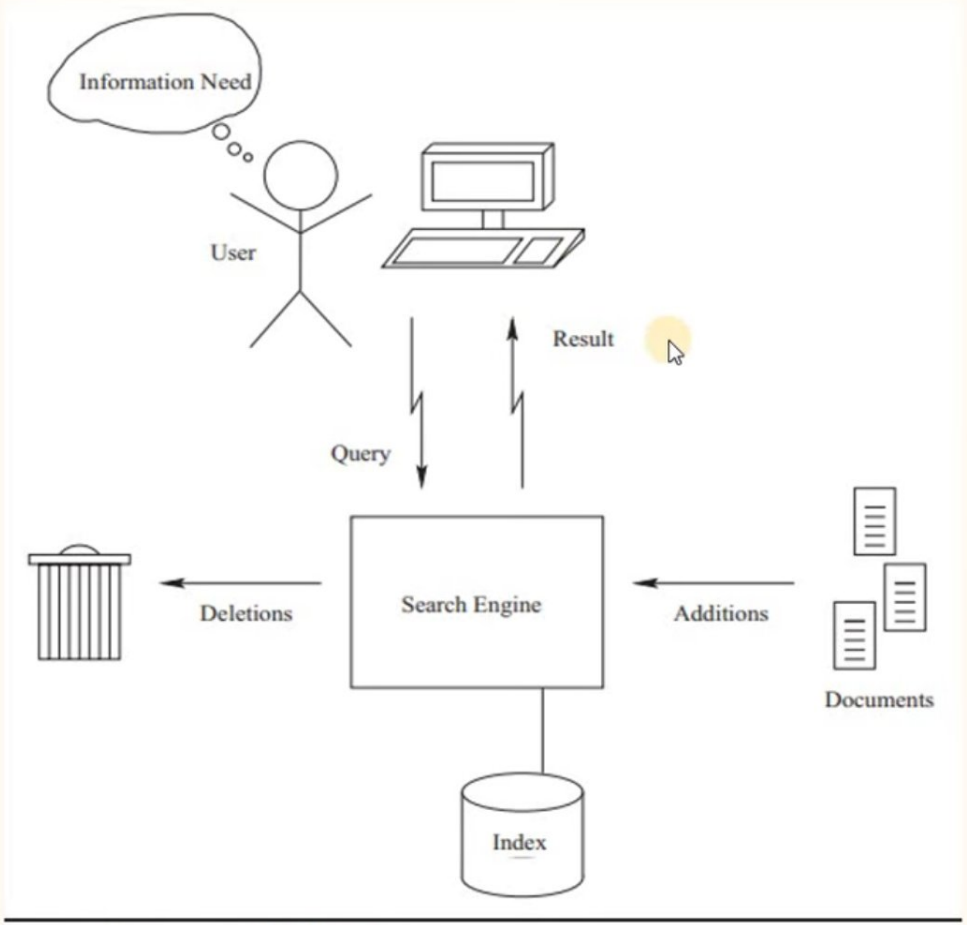
- Information retrieval (IR) is concerned with representing, searching, and manipulating large collections of electronic text and other human-language data.

- Information retrieval (IR) is finding material (usually documents) of an **unstructured** nature (usually text)

Basic Terms

- **Corpus** : Large repository of document stored in computer.
- **Information Need** : A topic about which we want to get information.
- **Relevance** : Some of the documents that may contain what I want to search.

So I express my need in the form of **query**



Components of an IR system.

Structured data

- Structured data is organized in tabular format (ie. rows and columns) and there is a relationship between different rows and columns.

- it's highly organized and formatted and easy to store, process, and access.

Unstructured data

- Unstructured data is data which is not organized in any predefined manner.
- It can be textual, numbers, dates, or BLOBs (Binary Large OBjects).
- Some key points to know about unstructured data:
  - Approximately 80% of the worldwide data is unstructured
  - Requires more storage
  - Rich media types (images, videos, audio) can also be analyzed with advanced technology
  - Some **examples** include text data, social media comments, documents, phone call transcriptions, various log files like server logs, sensor logs, image, audio, video, etc.

Semi-structured data

- Semi-structured data is a hybrid of both structured and unstructured data.
- It has some organizational framework but does not have the complete structure that is required to fit in a relational database.
- Semi-structured data has a self-describing structure that contains tags or attributes to separate various entities within data.
- Example: XML data.

An example information retrieval problem

To understand problem of IR in general , consider our data Operating System playlist.

Now information that I want to retrieve is "which videos in my playlist contains the terms : **Process** and **CPU** but not **deadlock**"

Traditional Solution

- to perform sorting of linear scan .
- This process is commonly GREP referred to as grepping through text, after the Unix command grep, which performs this process.
- The way to avoid linearly scanning the texts for each query is **to index the documents in advance.**

Better Solution

Processes the corpus (all documents) in advance.

For each video (document) , record which term appear in it.

The result is a binary term-document incidence matrix.

Term-Document Incidence Matrix

	Process control block	Process scheduling	CPU utilization	Deadlock in operating system	Disk scheduling algorithm	Critical section
process	1	1	0	0	0	1
kernel	0	0	0	1	0	0
CPU	1	1	1	0	0	0
scheduling	0	1	0	0	1	0
deadlock	0	0	0	1	0	0

Row represent : distinct terms that appear in video
Column represent : Different video in Operating System playlist

	Process control block	Process scheduling	CPU utilization	Deadlock in operating system	Disk scheduling algorithm	Critical section
process	1	1	0	0	0	1
kernel	0	1	0	1	0	0
CPU	1	1	1	0	0	0
scheduling	0	1	0	0	1	0
deadlock	0	0	0	1	0	1

Our Query : Process AND CPU but not deadlock

110001 AND 111000 AND 111010 (take complement)

Take bitwise AND operation : 110000

	Process control block	Process scheduling	CPU utilization	Deadlock in operating system	Disk scheduling algorithm	Critical section
process	1	1	0	0	0	1
kernel	0	1	0	1	0	0
CPU	1	1	1	0	0	0
scheduling	0	1	0	0	1	0
deadlock	0	0	0	1	0	1

Our Query : Process AND CPU but not deadlock

110001 AND 111000 AND 111010 (take complement)

Take bitwise AND operation : 110000

Answer : Process control block and process scheduling video contain word process and cpu but not deadlock

More realistic scenario

Suppose corpus has 1 million documents(text)

Number of distinct term : 100,000

Now if we create Term Document Incidence matrix for this scenario

Number of cell in matrix = 100,000 * 10,00,000

$= 0.1 * 10^{12}$

= 100GB

(if one cell is stored in 1 byte of memory)

More realistic scenario

Suppose corpus has 1 million documents(text)

Number of distinct term : 100,000

Now if we create Term Document Incidence matrix

Number of cell in matrix = 100,000 * 10,00,0

= $0.1 * 10^{12}$

= 100GB

(if one cell is stored in 1 byte of memory)

Term-Document Incidence Matrix is Sparse

	Process control block	Process scheduling	CPU utilization	Deadlock in operating system	Disk scheduling algorithm	Critical section
process	1	1	0	0	0	1
kernel	0	1	0	1	0	0
CPU	1	1	1	0	0	0
scheduling	0	1	0	0	1	0
deadlock	0	0	0	1	0	1

Term-Document Incidence Matrix is Sparse

	Process control block	Process scheduling	CPU utilization	Deadlock in operating system	Disk scheduling algorithm	Critical section
process	1	1	0	0	0	1
kernel	0	1	0	1	0	0
CPU	1	1	1	0	0	0
scheduling	0	1	0	0	1	0
deadlock	0	0	0	1	0	1

To overcome this problem , this matrix is converted into an **inverted index**

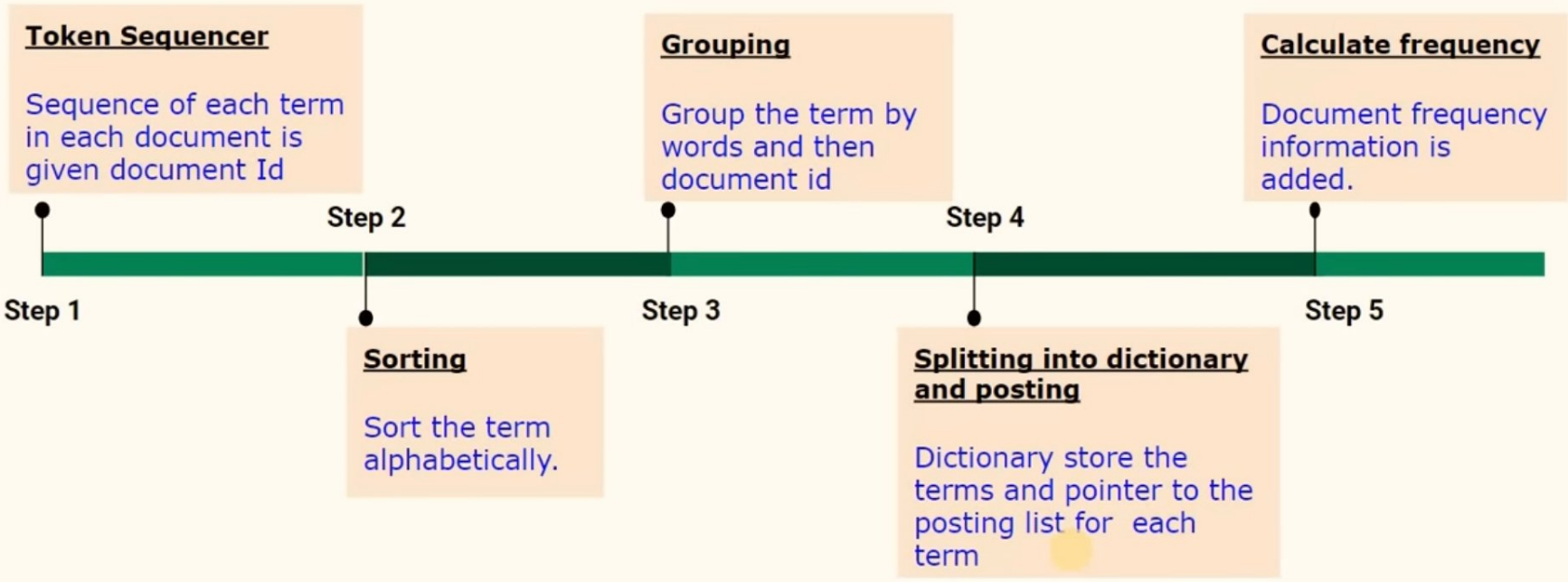
What is an Inverted index ?

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.**

It is a hashmap like data structure that directs you **from a word to a document or a web page.**

Building an Inverted Index

To create an inverted index,



Building an inverted index - Example

Doc1 :

Ram was average student in study but once he became friend of Shyam he started attending lecture regularly.

Doc2:

Shyam was enthusiastic student. He had helped his friend Ram a lot.Now Ram and Shyam are good friends.

Building an inverted index - Example

Term	DocID
Ram	1
was	1
average	1
student	1
in	1
study	1
but	1
once	1
he	1
became	1
friend	1
of	1
Shyam	1
he	1
started	1
attending	1
lecture	1
regularly	1

Term	DocID
Shyam	2
was	2
very	2
enthusiastic	2
student	2
He	2
had	2
helped	2
his	2
friend	2
Ram	2
a	2
lot	2
Now	2
Ram	2
and	2
Shyam	2
are	2
good	2
friend	2

Term	DocID
a	2
and	2
are	2
attending	1
average	1
became	1
but	1
enthusiastic	2
friend	1
friend	2
friend	2
good	2
had	2
he	1
he	1
He	2
helped	2
his	2
in	1
in	1
lecture	1

Building an inverted index - Example

Term	DocID
a	2
and	2
are	2
attending	1
average	1
became	1
but	1
enthusiastic	2
friend	1
friend	2
friend	2
good	2
had	2
he	1
he	1
He	2
helped	2
his	2
in	1
lecture	1
lot	2

Term	DocID
Now	2
in	1
lecture	1
lot	2
Now	2
of	1
once	1
Ram	1
Ram	2
Ram	2
regularly	1
Shyam	1
Shyam	2
Shyam	2
started	1
student	1
student	2
study	1
very	2
was	1
was	2

Term	Doc. Frequency
a	1
and	1
are	1
attending	1
average	1
became	1
but	1
enthusiastic	1
friend	2
good	1
had	1
he	1
He	1
helped	1
his	1
in	1
lecture	1
lot	1
Now	1
of	1
once	1
Ram	2
regularly	1
Shyam	2
started	1
student	2
very	1
was	2

Building an inverted index - Example

Term	DocID
a	2
and	2
are	2
attending	1
average	1
became	1
but	1
enthusiastic	2
friend	1
friend	2
friend	2
good	2
had	2
he	1
he	1
He	2
helped	2
his	2
in	1
lecture	1
lot	2

Term	DocID
Now	2
in	1
lecture	1
lot	2
Now	2
of	1
once	1
Ram	1
Ram	2
Ram	2
regularly	1
Shyam	1
Shyam	2
Shyam	2
started	1
student	1
student	2
study	1
very	2
was	1
was	2

Term	Doc. Frequency	Posting List
a	1	2
and	1	2
are	1	2
attending	1	1
average	1	1
became	1	1
but	1	1
enthusiastic	1	2
friend	2	1
good	1	2
had	1	1
he	1	1
He	1	2
helped	1	2
his	1	2
in	1	1
lecture	1	1
lot	1	2
Now	1	2
of	1	1
once	1	1
Ram	2	1
regularly	1	1
Shyam	2	1
started	1	1
student	2	1
very	1	2
was	2	1

Building an inverted index - Example

Term	DocID
a	2
and	2
are	2
attending	1
average	1
became	1
but	1
enthusiastic	2
friend	1
friend	2
friend	2
good	2
had	2
he	1
he	1
He	2
helped	2
his	2
in	1
lecture	1
lot	2

Term	DocID
Now	2
in	1
lecture	1
lot	2
Now	2
of	1
once	1
Ram	1
Ram	2
Ram	2
regularly	1
Shyam	1
Shyam	2
Shyam	2
started	1
student	1
student	2
study	1
very	2
was	1
was	2

Term	Doc. Frequency	Posting List
a	1	2
and	1	2
are	1	2
attending	1	1
average	1	1
became	1	1
but	1	1
enthusiastic	1	2
friend	2	1 → 2
good	1	2
had	1	1
he	1	1
He	1	2
helped	1	2
his	1	2
in	1	1
lecture	1	1
lot	1	2
Now	1	2
of	1	1
once	1	1
Ram	2	1
regularly	1	1
Shyam	2	1
started	1	1
student	2	1
very	1	2
was	2	1

Building an inverted index - Example

Term	DocID
a	2
and	2
are	2
attending	1
average	1
became	1
but	1
enthusiastic	2
friend	1
friend	2
friend	2
good	2
had	2
he	1
he	1
He	2
helped	2
his	2
in	1
lecture	1
lot	2

Term	DocID
Now	2
in	1
lecture	1
lot	2
Now	2
of	1
once	1
Ram	1
Ram	2
Ram	2
regularly	1
Shyam	1
Shyam	2
Shyam	2
started	1
student	1
student	2
study	1
very	2
was	1
was	2

Term	Doc. Frequency	Posting List
a	1	2
and	1	2
are	1	2
attending	1	1
average	1	1
became	1	1
but	1	1
enthusiastic	1	2
friend	2	1 → 2
good	1	2
had	1	1
he	1	1
He	1	2
helped	1	2
his	1	2
in	1	1
lecture	1	1
lot	1	2
Now	1	2
of	1	1
once	1	1
Ram	2	1 → 2
regularly	1	1
Shyam	2	1
started	1	1
student	2	1
very	1	2
was	2	1

Building an inverted index - Example

Term	DocID
a	2
and	2
are	2
attending	1
average	1
became	1
but	1
enthusiastic	2
friend	1
friend	2
friend	2
good	2
had	2
he	1
he	1
He	2
helped	2
his	2
in	1
lecture	1
lot	2

Term	DocID
Now	2
in	1
lecture	1
lot	2
Now	2
of	1
once	1
Ram	1
Ram	2
Ram	2
regularly	1
Shyam	1
Shyam	2
Shyam	2
started	1
student	1
student	2
study	1
very	2
was	1
was	2

Term	Doc. Frequency	Posting List
a	1	2
and	1	2
are	1	2
attending	1	1
average	1	1
became	1	1
but	1	1
enthusiastic	1	2
friend	2	1 → 2
good	1	2
had	1	1
he	1	1
He	1	2
helped	1	2
his	1	2
in	1	1
lecture	1	1
lot	1	2
Now	1	2
of	1	1
once	1	1
Ram	2	1 → 2
regularly	1	1
Shyam	2	1 → 2
started	1	1
student	2	1 → 2
very	1	2
was	2	1

Boolean Retrieval Query

Boolean retrieval model : In this model we represent query which is in boolean expressions of terms.

Terms are combined with **AND , NOR , NOT**

Boolean Retrieval Query

Boolean retrieval model : In this model we represent query which is in boolean expressions of terms.

Terms are combined with AND , NOR , NOT

X AND Y: represents doc that contains both X and Y.

X OR Y: represents doc that contains either X or Y.

NOT X: represents the doc that do not contain X.

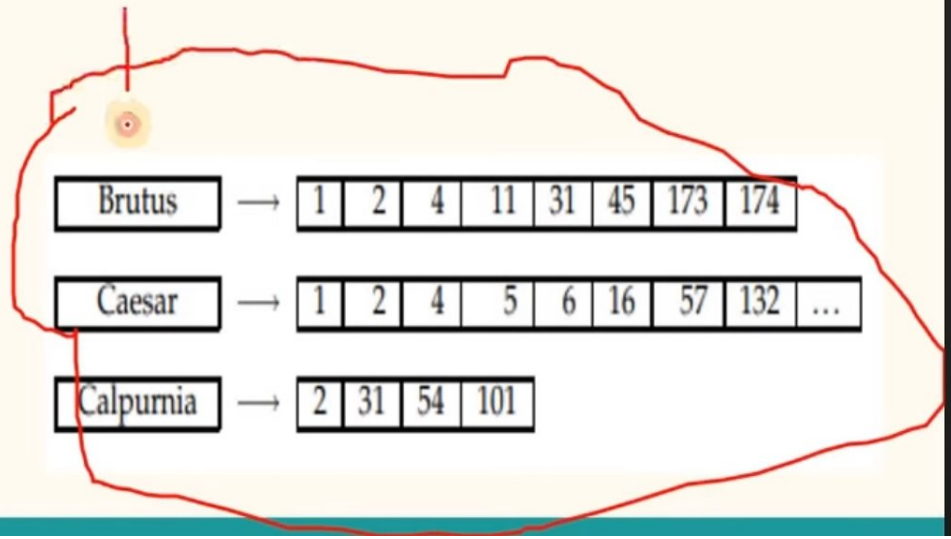
Processing Boolean queries

How do we process a query using an inverted index and the basic Boolean retrieval model? Consider processing the simple conjunctive query:

Brutus AND Calpurnia

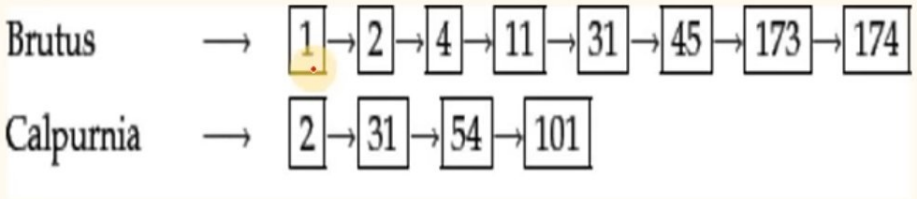
over the inverted index partially shown in Figure 1.3 (page 7). We:

1. Locate Brutus in the Dictionary
2. Retrieve its postings
3. Locate Calpurnia in the Dictionary
4. Retrieve its postings
5. Intersect the two postings lists, as shown below:

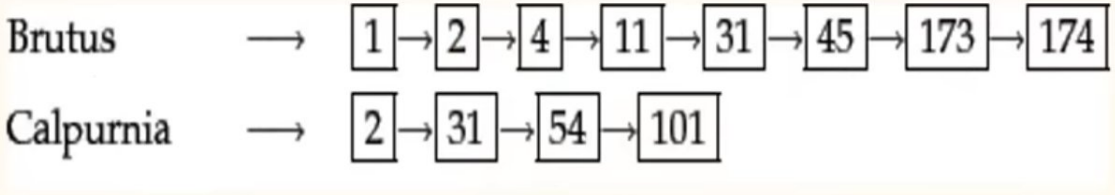


Implementation(Brutus AND Calpurnia)

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```



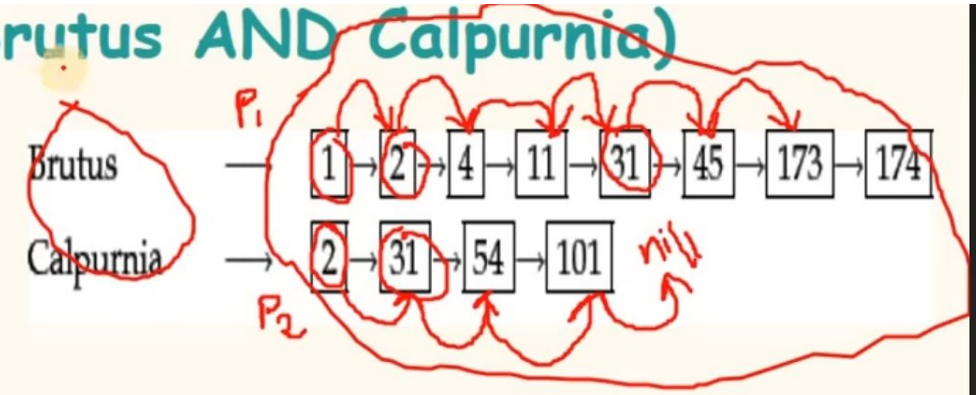
Implementation(Brutus OR Calpurnia)



Answer : based on Union

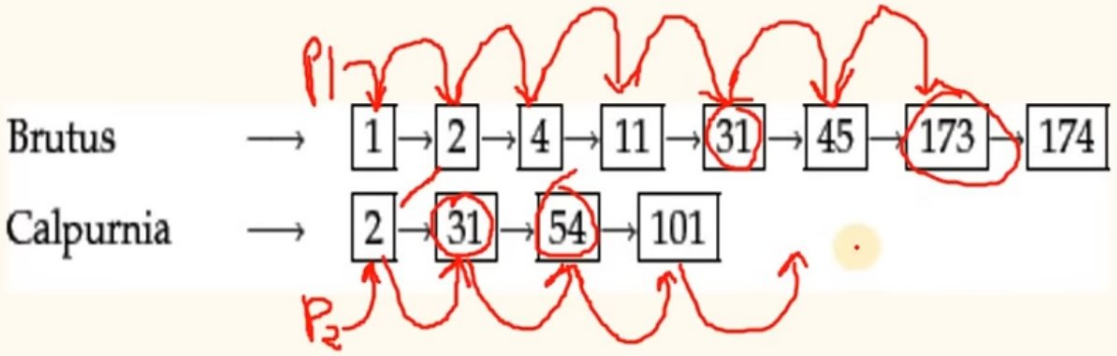
Implementation(Brutus AND Calpurnia)

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```



Answer : based on Intersection

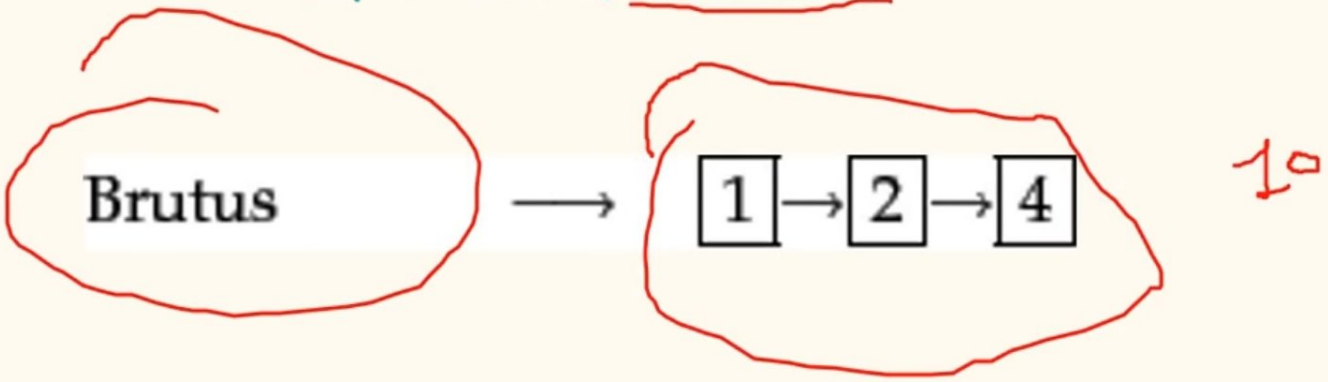
Implementation(Brutus OR Calpurnia)



1	2	4	11	31	45	54	101	

Answer : based on Union

Implementation(NOT Brutus)



10

3	5	6	7	9	9	10		

Answer : based on NOT

Tokenization

tokenization is the task of chopping up text into pieces, called **tokens**. perhaps at the same time throwing away certain characters.

Input: Krishna , Happy, Learning , lifelong process.

Output :

Krishna	Happy	Learning	lifelong	process

These tokens are often loosely referred to as **terms or words**, but it is sometimes important to make a **type/token distinction**.

Token vs Type vs Term

Token

A token is an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing.

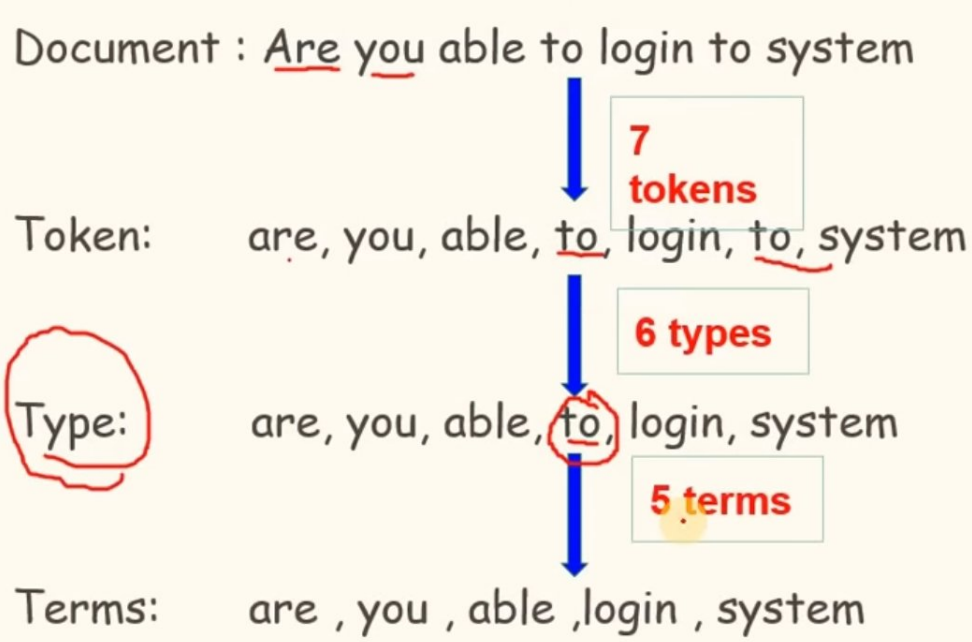
Type

A type is the class of all tokens containing the same character sequence.

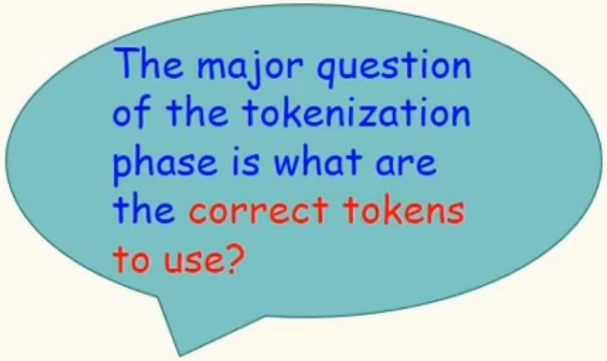
Term

A term is a (perhaps normalized) type that is included in the IR system's dictionary.

Token vs Type vs Term : Example



Issues with Tokenization



it looks fairly trivial: you chop on whitespace and throw away punctuation characters.

Example : apostrophe

Document : Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.

?

Neill	
ONeill	
O'	Neill
O	Neill

Example : apostrophe

Document : Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.

?

Neill	
ONeill	
O'	Neill
O	Neill

Example : apostrophe

Document : Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.

What should be for **aren't** ?

?

Neill
ONeill
O' Neill
O Neill

Co-education ?
Long-term ?
Lowercase ?

Language Issues

East Asian Languages (e.g., Chinese, Japanese, Korean, and Thai), where text is written without **any spaces between words.**

One approach here is to perform **word segmentation** as prior linguistic processing.

Language Issues

East Asian Languages (e.g., Chinese, Japanese, Korean, and Thai), where text is written without **any spaces between words.**

One approach here is to perform **word segmentation** as prior linguistic processing.

all such methods make mistakes sometimes, and so you are never guaranteed a consistent unique tokenization.

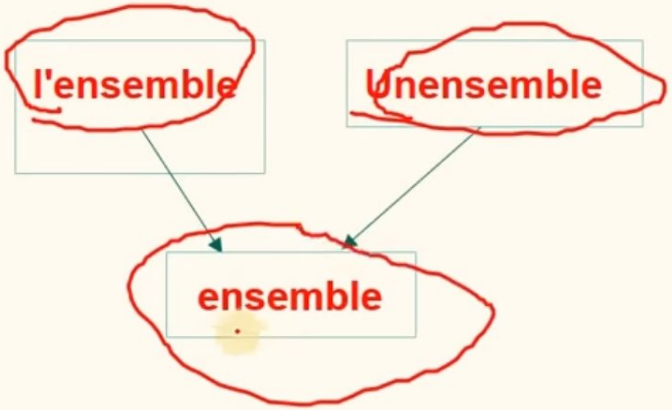
Language Issues

German writes compound nouns without spaces (e.g., Computerlinguistik `computational linguistics')

Retrieval systems for German greatly benefit from the use of a compound-splitter module, which is usually implemented by seeing if a word can be subdivided into multiple words that appear in a vocabulary.

Language Issues

French has a variant use of the apostrophe for a reduced definite article the before a word beginning with a vowel (e.g., l'ensemble)



Few more Examples

White Spaces

1. Los angeles
2. San francisco

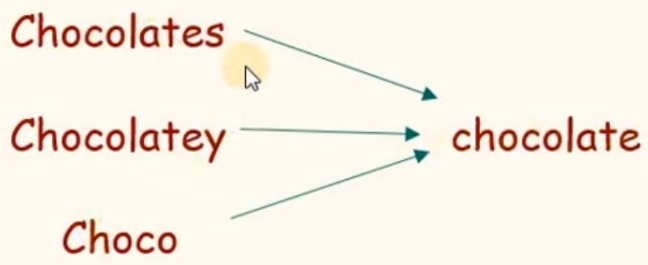
Date formats

- 2/6/2021 or June/2/2021 or 2-6-2021
- 2 , 6 , 2021 , June ....???

What is Stemming

Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form.

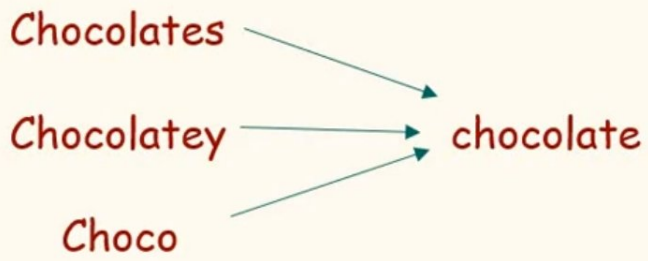
Stemming programs are commonly referred to as **stemming algorithms** or stemmers.

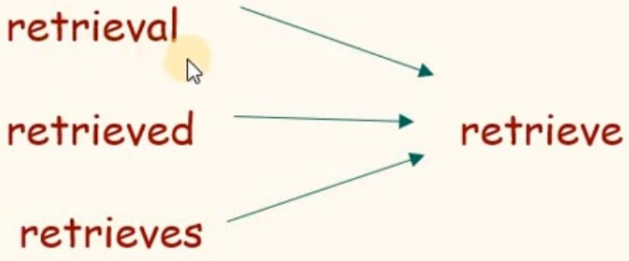


What is Stemming

Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form.

Stemming programs are commonly referred to as **stemming algorithms** or stemmers.





Applications of stemming :

1. information retrieval systems like search engines
2. It is used to determine domain vocabularies in domain analysis.

Stemming algorithms

- Porter's Stemmer algorithm
- Lovins Stemmer
- Dawson Stemmer
- Krovetz Stemmer
- Lancaster Stemmer
- N-Gram Stemmer
- Xerox Stemmer

Rule			Example		
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
SS	→	SS	caress	→	caress
S	→		cats	→	cat

Stemming algorithms

- Porter's Stemmer algorithm
- Lovins Stemmer
- Dawson Stemmer
- Krovetz Stemmer
- Lancaster Stemmer
- N-Gram Stemmer
- Xerox Stemmer

Rule			Example		
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
SS	→	SS	caress	→	caress
S	→		cats	→	cat

sitting -> sitt -> sit

- 'children' -> 'child'
- 'understood' -> 'understand'
- 'whom' -> 'who'
- 'best' -> 'good'

What are Stop words?

Stop Words: A stop word is a commonly used word (such as "the", "a", "an", "in") that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.

a	an	and	are	as	at	be	by	for	from
has	he	in	is	it	its	of	on	that	the
to	was	were	will	with					

Is it necessary ?

Using a stop list significantly reduces the number of postings that a system has to store.

But you need them for

➔ The phrase query "President of the India"

Is it necessary ?

Using a stop list significantly reduces the number of postings that a system has to store.

But you need them for

- ➔ The phrase query "President of the India"
- ➔ Relational Query like "flights to London"

Is it necessary ?

Using a stop list significantly reduces the number of postings that a system has to store.

But you need them for

- ➔ The phrase query "President of the India"
- ➔ Relational Query like "flights to London"
- ➔ Some song titles (To be or not to be, Let It Be, I don't want to be, ..)

Postings Lists :Document-Oriented Indices

- the numbering of positions in a document collection may be split into two components:

  A document number , an offset within the document

- the notation n:m to indicate positions within a document

- n is a document identifier (or docid) and m is an offset.

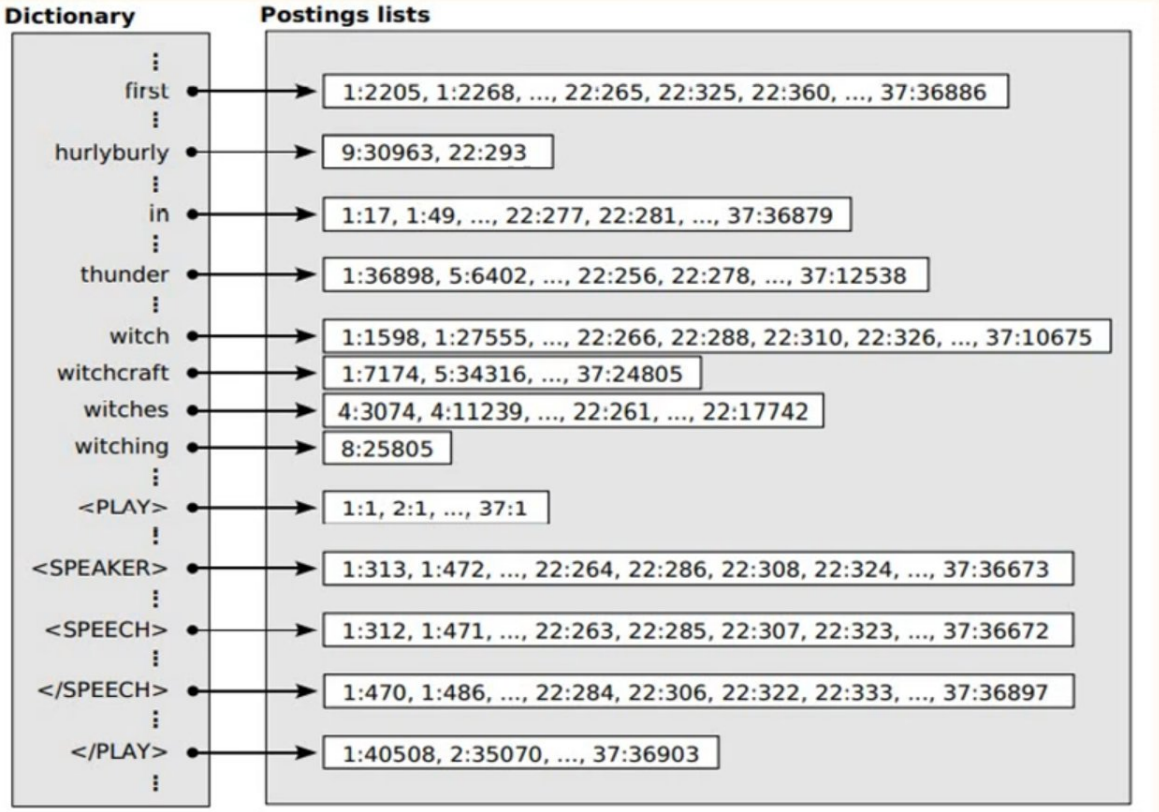


Figure 2.7 A document-centric index for Shakespeare's plays equivalent to the one shown in Figure 2.1 (page 34). Each posting is of the form *docid:within-document-position*.

For example,

first("hurlyburly") = 9:30963
first("witching") = 8:25805

last("thunder") = 37:12538
last("witching") = 8:25805

next("witch", 22:288) = 22:310
next("hurlyburly", 9:30963) = 22:293
next("witch", 37:10675) = ∞

prev("witch", 22:310) = 22:288
prev("hurlyburly", 22:293) = 9:30963
prev("witch", 1:1598) = $-\infty$

✓ schema-dependent inverted index

Positional Index

N_t	*document frequency*	the number of documents in the collection containing the term t
$f_{t,d}$	*term frequency*	the number of times term t appears in document d
l_d	*document length*	measured in tokens
l_{avg}	*average length*	average document length across the collection
N	*document count*	total number of documents in the collection

Positional Index

N_t	*document frequency*	the number of documents in the collection containing the term t
$f_{t,d}$	*term frequency*	the number of times term t appears in document d
l_d	*document length*	measured in tokens
l_{avg}	*average length*	average document length across the collection
N	*document count*	total number of documents in the collection

many postings may now share a common prefix in their docids.

We can separate out these docids to produce postings of the form

$(d, f_{t,d}, p0, \ldots, pf_{t,d})$

Positional Index

N_t	*document frequency*	the number of documents in the collection containing the term t
$f_{t,d}$	*term frequency*	the number of times term t appears in document d
l_d	*document length*	measured in tokens
l_{avg}	*average length*	average document length across the collection
N	*document count*	total number of documents in the collection

many postings may now share a common prefix in their docids.

We can separate out these docids to produce postings of the form

$$(d, f_{t,d}, p0, \ldots, pf_{t,d})$$

where $p0, \ldots, pf_{t,d}$ is the list of the offsets of all $f_{t,d}$ occurrences of the term t within document d.

Positional Index

N_t	*document frequency*	the number of documents in the collection containing the term t
$f_{t,d}$	*term frequency*	the number of times term t appears in document d
l_d	*document length*	measured in tokens
l_{avg}	*average length*	average document length across the collection
N	*document count*	total number of documents in the collection

many postings may now share a common prefix in their docids.

We can separate out these docids to produce postings of the form

$$(d, f_{t,d}, p0, \ldots, pf_{t,d})$$

where $p0, \ldots, pf_{t,d}$ is the list of the offsets of all $f_{t,d}$ occurrences of the term t within document d.

Using this notation, we would write the postings list for the term "witch" as

Positional Index

N_t	*document frequency*	the number of documents in the collection containing the term t
$f_{t,d}$	*term frequency*	the number of times term t appears in document d
l_d	*document length*	measured in tokens
l_{avg}	*average length*	average document length across the collection
N	*document count*	total number of documents in the collection

many postings may now share a common prefix in their docids.

We can separate out these docids to produce postings of the form

$$(d, f_{t,d}, p0, \ldots, pf_{t,d})$$

where $p0, \ldots, pf_{t,d}$ is the list of the offsets of all $f_{t,d}$ occurrences of the term t within document d.

Using this notation, we would write the postings list for the term "witch" as

(1, 3, 1598, 27555, 31463), ..., (22, 52, 266, 288, ...), ..., (37, 1, 10675)

Pointers: two conflicting forces

A term like Calpurnia occurs in maybe one doc out of a million - would like to store this pointer using log2 1M ~ 20 bits.

- A term like the occurs in virtually every doc, so 20 bits/pointer is too expensive.

Pointers: two conflicting forces

A term like Calpurnia occurs in maybe one doc out of a million - would like to store this pointer using log2 1M ~ 20 bits.

- A term like the occurs in virtually every doc, so 20 bits/pointer is too expensive.
- Prefer 0/1 vector in this case.

Compressing Postings Lists

The exact method depends on the type of the index (docid, frequency, positional, or schema-independent),

Consider the following sequence of integers that could be the beginning of a term's postingslist in a docid index:

$$L = (3, 7, 11, 23, 29, 37, 41, \ldots)$$

the elements in L form a monotonically increasing sequence , the list can be transformed into an equivalent sequence called as **Δ-values:**

Compressing Postings Lists

Positional Index

It is also apply for a document-centric positional index with postings of the form $(d, f_{t,d}, p1,\ldots,pf_{t,d})$.

For example, the list

$$L = ((3, 2, 157, 311),(7, 1, 212),(11, 3, 17, 38, 133),\ldots)$$

Can be represented as,

Compressing Postings Lists

Positional Index

It is also apply for a document-centric positional index with postings of the form $(d, f_{t,d}, p1,\ldots,pf_{t,d})$.

For example, the list

$$L = ((3, 2, 157, 311),(7, 1, 212),(11, 3, 17, 38, 133),\ldots)$$

Can be represented as,

$$L = ((3, 2, 157, 311),(4, 1, 212),(4, 3, 17, 38, 133),\ldots)$$

Postings Compression methods

parametric Codes

- ➔ conduct an analysis of some statistical properties of the list to be compressed.
- ➔ A parameter value is selected based on the outcome of this analysis

nonparametric codes.

- ➔ Ignores the actual Δ-gap.
- ➔ it assumes that all postings lists look more or less the same and share some common properties.

Elias's γ code for gap encoding (nonparametric)

Steps in Encoding:

The γ codeword for a positive integer k consists of two components, ,

1.**Sector** 2.**Body**

To encode a number X,

1. Find the largest N, with 2^N <= X(greater power of 2)
2. Encode N using Unary coding(i.e N zeroes followed by a one). - **Sector part**
3. Represent the integer ($X-2^N$) using N digits in Binary.- **Body part**

Elias's γ code for gap encoding (nonparametric)

Steps in Encoding:

The γ codeword for a positive integer k consists of two components, ,

1.**Sector** 2.**Body**

To encode a number X,

1. Find the largest N, with 2^N <= X(greater power of 2)
2. Encode N using Unary coding(i.e N zeroes followed by a one). - **Sector part**
3. Represent the integer $(X-2^N)$ using N digits in Binary.- **Body part**

Elias's γ code for gap encoding (nonparametric)

Steps in Encoding:

The γ codeword for a positive integer k consists of two components, ,

1. **Sector** 2. **Body**

To encode a number X,

1. Find the largest N, with 2^N <= X(greater power of 2)
2. Encode N using Unary coding(i.e N zeroes followed by a one). - **Sector part**
3. Represent the integer $(X-2^N)$ using N digits in Binary.- **Body part**

Example : Elias's γ encoding

For example, the gap integers for posting is as follow : 1, 5, 7, 16 .

Lets now encode first 7,

1. $7 = 2^2 + 3$
2. So N = 2 , Encode 2 using Unary coding(i.e 2 zeroes followed by a one). - **001 (sector part)**
3. integer $(X-2^N) = 7-4=3$, in binary using 2 digit is - **11 (Body part)**

So 7 is represented as ,

(001 11) in Elias's γ encoding

Example : Elias's γ encoding

Performing similar steps for remaining integer ,

K	selector(k)	body(k)
1	1	0
5	001	01
7	001	11
16	00001	0000

Example : Elias's γ encoding

3

Performing similar steps for remaining integer ,

K	selector(k)	body(k)
1	1	0
5	001	01
7	001	11
16	00001	0000

The binary representation of a positive integer k consists of

$\lfloor \log_2(k) \rfloor + 1$ bits.

Thus, the length of k's codeword in the γ code is

Example : Elias's γ encoding

3

Performing similar steps for remaining integer ,

K	selector(k)	body(k)
1	1	0
5	001	01
7	001	11
16	00001	0000

The binary representation of a positive integer k consists of

$\lfloor \log_2(k) \rfloor + 1$ bits.

Thus, the length of k's codeword in the γ code is

$|\gamma(k)| = 2 \cdot \lfloor \log_2(k) \rfloor + 1$ bits.

Introduction to Zipf's Law

The content and structure of electronic text may be encoded into various format supported by various word processing programs and desktop publishing systems.

Two formats are of special interest to us : **HTML and XML**

As a preliminary step before tokenization, documents in binary formats must be converted to raw text — a stream of characters

distribution of word frequencies is very skewed. There are a few words that have very high frequencies and many words that have low frequencies.

Example to Understand Zipf's Law

The set of distinct tokens, or symbols, in a text collection is called the vocabulary, denoted as V. Our collection of Shakespeare's plays has |V| = 22,987 symbols in its vocabulary.

V = {a, aaron, abaissiez, ..., zounds, zwaggered, ..., <PLAY>, ..., <SPEAKER>, ..., </PLAY>, ...}

Table 1.1 The twenty most frequent terms in Bosak's XML version of Shakespeare.

Rank	Frequency	Token	Rank	Frequency	Token
1	107,833	<LINE>	11	17,523	of
2	107,833	</LINE>	12	14,914	a
3	31,081	<SPEAKER>	13	14,088	you
4	31,081	</SPEAKER>	14	12,287	my
5	31,028	<SPEECH>	15	11,192	that
6	31,028	</SPEECH>	16	11,106	in
7	28,317	the	17	9,344	is
8	26,022	and	18	8,506	not
9	22,639	i	19	7,799	it
10	19,898	to	20	7,753	me

Zipf's Law : Modeling the distribution of terms

This distribution is described by Zipf 's law,

- ➔ which states that the frequency of the rth most common word is **inversely** proportional to r or, alternatively, the rank of a word times its frequency (f) is approximately a constant (k):

$$r \cdot f = k$$

- ➔ In terms of probability Zipf 's law is: $r \cdot P_r = c$ where P_r = is the probability of occurrence for the rth ranked word

Zipf's Law

Mathematically, the relationship may be expressed as

$$\log(\text{frequency}) = C - \alpha \cdot \log(\text{rank}),$$

or equivalently as

$$F_i \sim 1 / i^{\alpha},$$

- Fi is the frequency of the ith most frequent term
- the value of α may vary, but it is usually close to 1

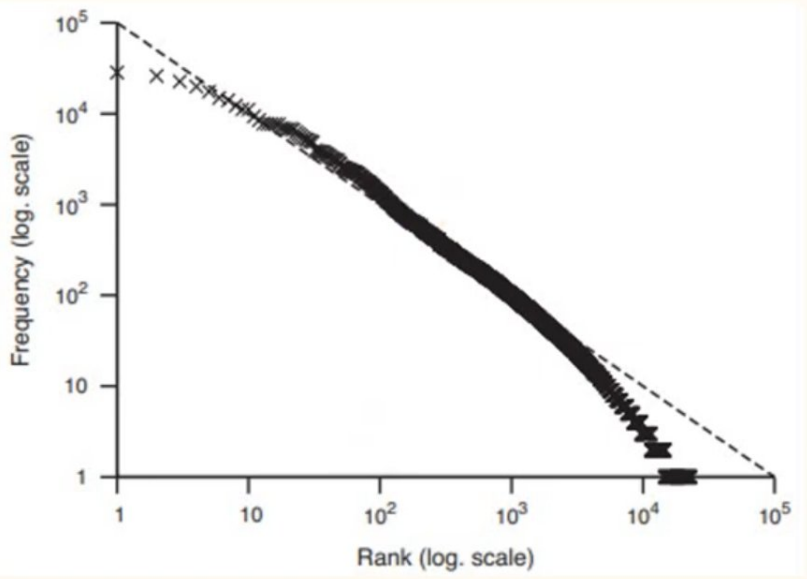


Figure :Frequency of words in the Shakespeare collection, by rank order. The dashed line corresponds to Zipf's law with α = 1.

Index Construction

Index Construction :The text collection is processed sequentially, one token at a time, and a postings list is built for each term in the collection in an incremental fashion.

- the process or machine that performs it called the INDEXER .(Indexing algorithm).

Text fragment:

⟨SPEECH⟩
⟨SPEAKER⟩ JULIET ⟨/SPEAKER⟩
⟨LINE⟩ O Romeo, Romeo! wherefore art thou Romeo? ⟨/LINE⟩
⟨LINE⟩ ...

Original tuple ordering (collection order):

..., ("⟨speech⟩", 915487), ("⟨speaker⟩", 915488), ("juliet", 915489),
("⟨/speaker⟩", 915490), ("⟨line⟩", 915491), ("o", 915492), ("romeo", 915493),
("romeo", 915494), ("wherefore", 915495), ("art", 915496), ("thou", 915497),
("romeo", 915498), ("⟨/line⟩", 915499), ("⟨line⟩", 915500), ...

New tuple ordering (index order):

..., ("⟨line⟩", 915491), ("⟨line⟩", 915500), ...,
("romeo", 915411), ("romeo", 915493), ("romeo", 915494), ("romeo", 915498),
..., ("wherefore", 913310), ("wherefore", 915495), ("wherefore", 915849), ...

Index Construction

IR system design also depends on the hardware on which system run.

- Access time of data (memory and disk)
- Read write operation (seek time)
- Buffering
- Processor utilization during data transfer

Index construction methods

in-memory index construction

- ➔ build an index entirely in main memory
- ➔ used if the collection is small relative to the amount of available RAM

disk-based index construction.

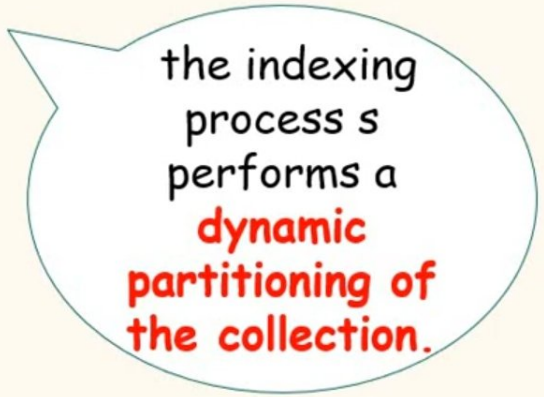
- ➔ used to build indices for very large collections.
- ➔ much larger than the available amount of main memory

Index construction methods

- In-Memory Index Construction (in memory method)
- Sort-Based Index Construction (in memory + on disk method)
- Merge-Based Index Construction (in memory + on disk method)

Merge-based indexing

- index is being created is small enough for the index to fit completely into main memory, then merge-based indexing behaves exactly like in-memory indexing.
- What if text collection is too large ?



Steps in Merge based indexing

1. build the inverted list structure I until memory runs out.(in - memory)
2. write the partial index I to disk, then start making a new one.(on - disk)
3. This procedure is repeated until the whole collection has been indexed.
4. At the end of this process, the disk is filled with many partial indexes(index partition), I1, I2, I3, ..., In. The system then merges these files into a single result.

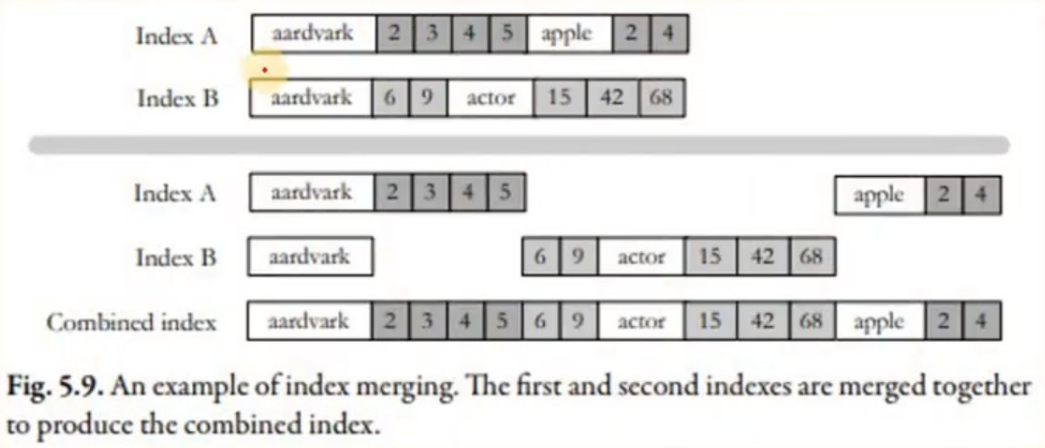


Fig. 5.9. An example of index merging. The first and second indexes are merged together to produce the combined index.

Analysis of Merge sort based Indexing

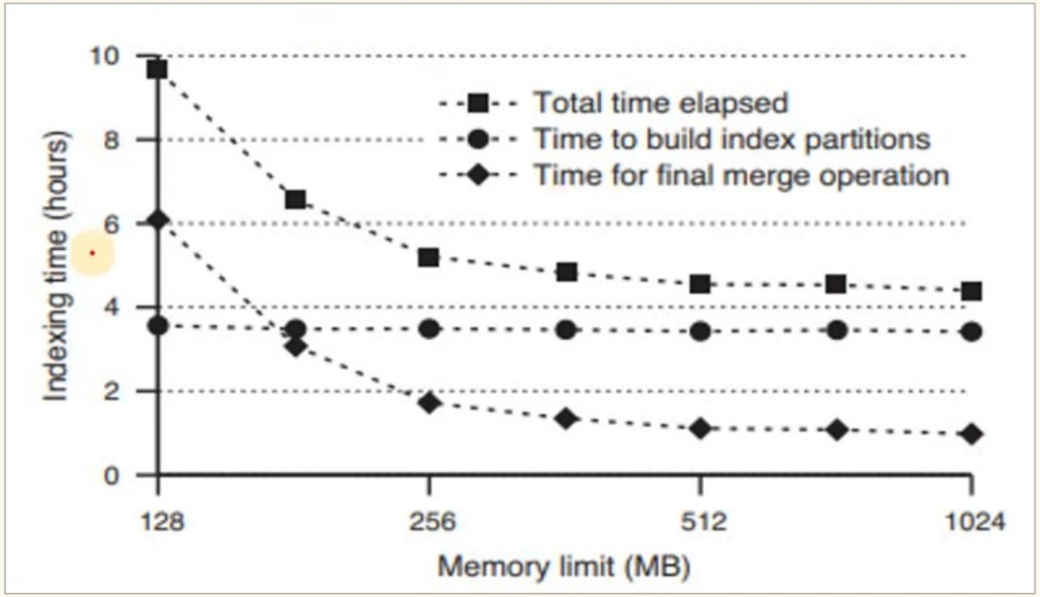
- Algorithm is very scalable but has some limitations, like size of **read write buffer** , M/n, where M is the amount of available memory and n is the number of partitions. Thus, if n becomes too large, merging becomes slow)

- It has two effects,
  - decreases the total amount of memory available for the read-ahead buffers.
  - increases the number of index partitions

- reducing main memory by 50% decreases the size of each index partition's read-ahead buffer by 75%

- Setting the memory limit to M = 128 MB, for example, results in 3,032 partitions that need to be merged, leaving each partition with a read-ahead buffer of only 43 KB.

Example : Analysis of Merge sort based Indexing



- Index partition is independent of available memory
- Merging partition suffer severely if small main memory is available

Dynamic Indexing

- Up to now, we have assumed that collections are static.
- Documents are inserted, deleted and modified in collection
- This means that the dictionary and postings lists have to be dynamically modified.

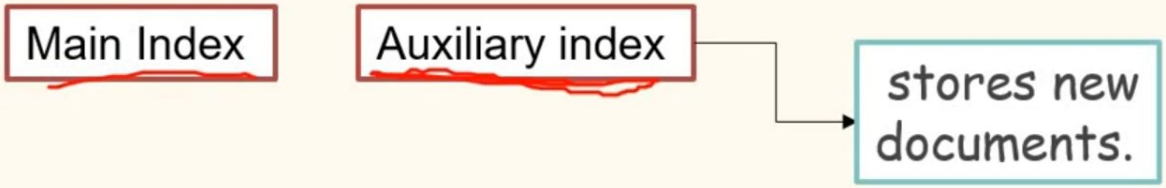
Dynamic Indexing

One approach is to periodically reconstruct the index from scratch.

- ✓ the number of changes over time is small
- ✓ a delay in making new documents searchable is acceptable
- ✓ if enough resources are available to construct a new index while the old one is still available for querying.

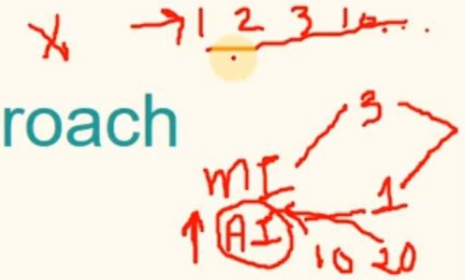
Dynamic Indexing

If new documents be included quickly,



- Maintain big main index on disk.
- New docs go into small auxiliary index in memory.

Dynamic indexing: Simplest approach



- Search across both, merge results
- Periodically, merge auxiliary index into main index

- Deletions:
  - Invalidation bit-vector for deleted docs
  - Filter docs returned by index using this bit-vector

Issue with auxiliary and main index

- Frequent merges
- Poor search performance during index merge

What is Retrieval Model

A retrieval model is a formal representation of the process of matching a query and a document.

It is the basis of the ranking algorithm that is used in a search engine to produce the ranked list of documents.

A good retrieval model will find documents that are likely to be considered relevant by the person who submitted the query.

Techniques such as query suggestion, query expansion, and **relevance** feedback use interaction and context to refine the initial query in order to produce better ranked lists.

two key aspects of **relevance** that are important for both retrieval models and evaluation measures.

1. the difference between topical and user relevance,

   A web page containing a biography of Abraham Lincoln would certainly be topically relevant to the query "Abraham Lincoln", and would also be topically relevant to the queries "U.S. presidents" and "Civil War", but **user relevance is same.**

1. whether it is binary or multivalued

   Binary relevance - relevant or non-relevant

   Multivalued relevance - relevant , non-relevant , unsure

Retrieval Model

- → **Boolean Retrieval**
- → **vector space**
- → **TFIDF**
- → **Okapi**
- → Probabilistic model
- → language modeling

Boolean Retrieval

- also known as exact-match retrieval
- Assumption : all documents in the retrieved set are equivalent in terms of relevance.
- two possible outcomes for query evaluation (TRUE and FALSE)
- query is usually specified using operators from Boolean logic (AND, OR, NOT).

As an example of Boolean query formulation

Query : lincoln

retrieve a large number of documents that mention Lincoln cars and places named Lincoln in addition to stories about President Lincoln.

Query : president AND lincoln

retrieve a set of documents that contain both words, occurring anywhere in the document.

Query : president AND lincoln AND NOT (automobile OR car)

removes too many relevant documents along with non-relevant documents

Query : president AND lincoln AND biography AND life AND birthplace AND gettysburg AND NOT (automobile OR car)

putting too many search terms into the query with the AND operator often results in nothing being retrieved.

Query : president AND lincoln AND (biography OR life OR birthplace OR gettysburg) AND NOT (automobile OR car)

Merit and Demerit of Boolean Retrieval

Merit :

- Results are very predictable and easy to explain to users.
- The operands of a Boolean query can be any document feature
- documents can be rapidly eliminated from consideration in the scoring process , so more efficient.

Demerit :

- the effectiveness depends entirely on the user

The Vector Space Model

In this model, documents and queries are assumed to be part of a t-dimensional vector space, where t is the number of index terms (words, stems, phrases, etc.).

A document D_i is represented by a vector of index terms:

$$D_i = (d_{i1}, d_{i2}, \ldots, d_{it}),$$

where d_{ij} represents the weight of the jth term

query Q is represented by a vector of t weights:

$$Q = (q_1, q_2, \ldots, q_t)$$

where q_j is the weight of the jth term in the query.

The Vector Space Model

A document collection containing n documents can be represented as a matrix of term weights,

	$Term_1$	$Term_2$	...	$Term_t$
Doc_1	d_{11}	d_{12}	...	d_{1t}
Doc_2	d_{21}	d_{22}	...	d_{2t}
⋮	⋮			
Doc_n	d_{n1}	d_{n2}	...	d_{nt}

The Vector Space Model

A document collection containing n documents can be represented as a matrix of term weights,

Column - term weight

Row- document

	$Term_1$	$Term_2$	...	$Term_t$
Doc_1	d_{11}	d_{12}	...	d_{1t}
Doc_2	d_{21}	d_{22}	...	d_{2t}
⋮	⋮			
Doc_n	d_{n1}	d_{n2}	...	d_{nt}

- The **term weights** are simply the count of the terms in the document.
- **Stopwords** are not indexed in this example,
- the words have been **stemmed**.

Example : The Vector Space Model

Given this representation, documents could be ranked by computing the distance between the points representing the documents and the query.

a similarity measure is used (rather than a distance or dissimilarity measure

the documents with the highest scores are the most similar to the query

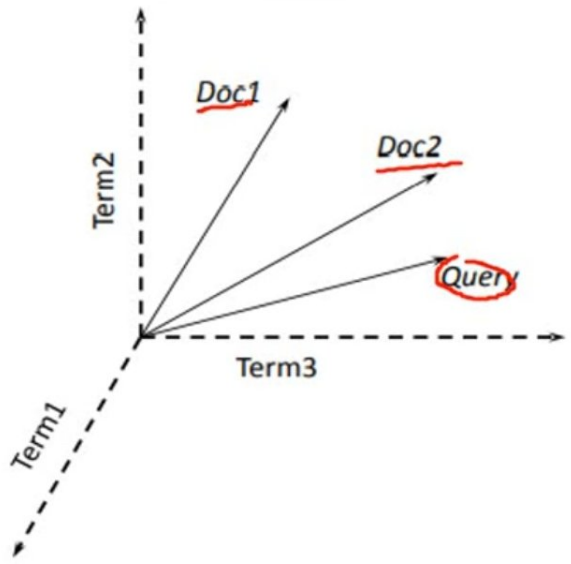


Fig. 7.2. Vector representation of documents and queries

Example : The Vector Space Model

The cosine correlation measures the cosine of the angle between the query and the document vectors.

When the vectors are normalized so that all documents and queries are represented by vectors of equal length, the cosine of the angle between two identical vectors will be 1 (the angle is zero), and for two vectors that do not share any non-zero terms, the cosine will be 0. The cosine measure is defined as,

$$Cosine(D_i, Q) = \frac{\sum_{j=1}^{t} d_{ij} \cdot q_j}{\sqrt{\sum_{j=1}^{t} d_{ij}^2 \cdot \sum_{j=1}^{t} q_j^2}}$$

Example : The Vector Space Model

As an example, consider two documents D1 = (0.5, 0.8, 0.3) and D2 = (0.9, 0.4, 0.2) indexed by three terms, where the numbers represent term **weights**.

Given the query Q = (1.5, 1.0, 0) indexed by the same terms, the cosine measures for the two documents are:

$$Cosine(D_1, Q) = \frac{(0.5 \times 1.5) + (0.8 \times 1.0)}{\sqrt{(0.5^2 + 0.8^2 + 0.3^2)(1.5^2 + 1.0^2)}} = \frac{1.55}{\sqrt{(0.98 \times 3.25)}} = 0.87$$

$$Cosine(D_2, Q) = \frac{(0.9 \times 1.5) + (0.4 \times 1.0)}{\sqrt{(0.9^2 + 0.4^2 + 0.2^2)(1.5^2 + 1.0^2)}} = \frac{1.75}{\sqrt{(1.01 \times 3.25)}} = 0.97$$

What is TF-IDF(term frequency-inverse document frequency)?

TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.

TF-IDF for a word in a document is calculated by multiplying two different metrics:

- The term frequency of a word in a document(t_f)
- The inverse document frequency(idf) of the word across a set of documents.The closer it is to 0, the more common a word is.

How is TF-IDF calculated?

So, if the word is very common and appears in many documents, this number will approach 0. Otherwise, it will approach 1.

(t_f * idf) results in the TF-IDF score of a word in a document. The higher the score, the more relevant that word is in that particular document.

mathematical form:

$$tf\,idf\,(t, d, D) = tf\,(t, d)\,.\,idf\,(t, D)$$

where,

$$tf\,(t, d) = \log\,(1 + freq\,(t, d))$$

$$idf\,(t, D) = \log\left(\frac{N}{count\,(d \in D : t \in d)}\right)$$

Probabilistic Models

Probability Ranking Principle doesn't tell us how to calculate or estimate the probability of relevance.

simple probabilistic model based on treating information retrieval as a classification problem.

Probabilistic Model: Information Retrieval as Classification

assumes relevance is binary, there will be two sets of documents, the relevant documents and the non-relevant documents.

the system should classify the document as relevant or non-relevant, and retrieve it if it is relevant.

P(R|D) is a conditional probability representing the probability of relevance given the representation of that document

P(NR|D) is the conditional probability of non-relevance

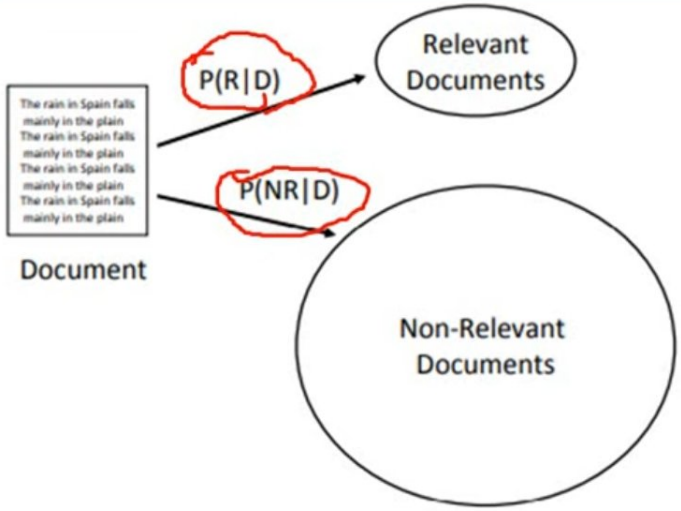


Fig. 7.3. Classifying a document as relevant or non-relevant

Probabilistic Model: Information Retrieval as Classification

let's focus on P(R|D)..... P(R|D) → P(D|R)

if we had information about how often specific words occurred in the relevant set, then, given a new document, it would be relatively straightforward to calculate how likely it would be to see the combination of words in the document occurring in the relevant set.

Let's assume that the probability of the word "president" in the relevant set is 0.02, and the probability of "lincoln" is 0.03. If a new document contains the words "president" and "lincoln", we could say that the probability of observing that combination of words in the relevant set is $0.02 \times 0.03 = 0.0006$, assuming that the two words occur independently

Probabilistic Model: Information Retrieval as Classification

So how does calculating P(D|R) get us to the probability of relevance? It turns out there is a relationship between P(R|D)and P(D|R) that is expressed by **Bayes' Rule** :

$$P(R|D) = \frac{P(D|R)P(R)}{P(D)}$$

where P(R) is the a priori probability of relevance (in other words, how likely any document is to be relevant), and P(D)acts as a normalizing constant.

classify a document as relevant **if P(D|R)P(R) > P(D|NR)P(NR)**. This is the same as classifying a document as relevant if:

likelihood ratio

$$\frac{P(D|R)}{P(D|NR)} > \frac{P(NR)}{P(R)}$$

If we use the **likelihood ratio** as a score, the highly ranked documents will be those that have a high likelihood of belonging to the relevant set.

Language Models

Language models are used to represent text in a variety of language technologies, such as speech recognition, machine translation, and handwriting recognition.

simplest form of language model, known as a **unigram language model**, is a probability distribution over the words in the language.

For example, if the documents in a collection contained just **five** different words, **a possible language model for that collection might be (0.2, 0.1, 0.35, 0.25, 0.1),** where each number is the probability of a word occurring. If we treat each document as a sequence of words, then the probabilities in the language model predict what the next word in the sequence will be.

For example, if the five words in our language were **"girl", "cat", "the", "boy", and "touched",** then the probabilities predict which of these words will be next.

With this model, for example, it is just as likely to get the sequence **"girl cat" (probability 0.2 × 0.1)** as **"girl touched" (probability 0.2 × 0.1).**

Ranking Based on Language Models

3-4-5

In applications such as speech recognition, n-gram language models that predict words based on longer sequences are used. An n-gram model predicts a word based on the previous n – 1 words.

language models is used to represent the topical content of a document.

For example,

- 1st Document is about - fishing in Alaska
- 2nd Document is about - fishing in Florida
- 3rd Document is about fishing games for computers

1. fishing and locations in Alaska
2. Fishing + more high probability words associated with locations in Florida
3. game manufacturers and computer use

Most of the words will have "default" probabilities that will be the same for any text, but the words that are important for the topic will have unusually high probabilities.

Relevance feedback and query expansion

- The same word can have different meanings (polysemy)
- Two different words can have the same meaning (synonymy)
- Vocabulary of searcher may not match that of the documents
- Consider the query = {plane fuel}
- While this is relatively unambiguous (wrt the meaning of each word in context), exact matching will miss documents containing aircraft, airplane, or jet
- Relevance feedback and query expansion aim to overcome the problem of **synonymy**

Relevance feedback and query expansion

Global methods are techniques for expanding or reformulating query :

- ✓ Query expansion/reformulation with a thesaurus
- ✓ Query expansion via automatic thesaurus generation
- ✓ Techniques like spelling correction

Local methods adjust a query relative to the documents that initially appear to match the query. The basic methods here are:

- ✓ Relevance feedback
- ✓ Pseudo relevance feedback, also known as Blind relevance feedback
- ✓ (Global) indirect relevance feedback

Relevance feedback

The idea of relevance feedback (RF) is to involve the user in the retrieval process so as to improve the final result set. The basic procedure is :

- ✓ The user issues a (short, simple) query.
- ✓ The system returns an initial set of retrieval results.
- ✓ The user marks some returned documents as relevant or nonrelevant.
- ✓ The system computes a better representation of the information need based on the user feedback.
- ✓ The system displays a revised set of retrieval result

Relevance feedback: Problems

Relevance feedback is expensive.

- Relevance feedback creates long modified queries.
- Long queries are expensive to process.

Users are reluctant to provide explicit feedback.

It's often hard to understand why a particular document was retrieved after applying relevance feedback.

Information retrieval system evaluation

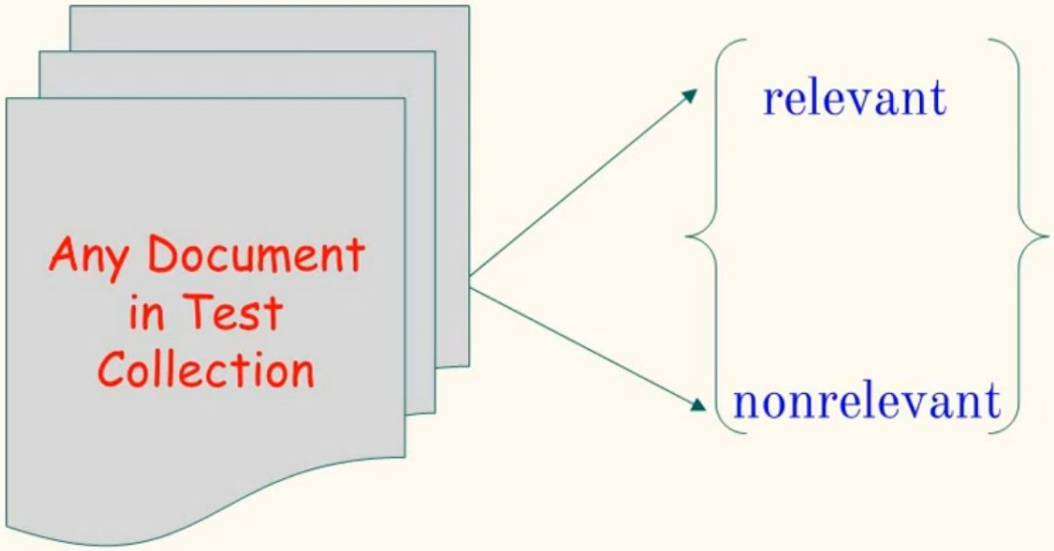
To measure ad hoc information retrieval effectiveness in the standard way, we need a test collection consisting of three things:

1. A document collection

2. A test suite of information needs, expressible as queries

3. A set of relevance judgments, standardly a binary assessment of either relevant or nonrelevant for each query-document pair

Information retrieval system evaluation



Decision ?

gold standard or ground truth judgment of relevance.

About Relevance

Relevance is assessed relative to an information need, not a query.

Example :

> Information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine.

This might be translated into a query such as:

wine AND red AND white AND heart AND attack AND effective

Evaluating Search Engines

Why Evaluate?

- effectiveness and efficiency
- cost(processors, memory, disk space, and networking)
- Comparison between technique

→ ability of the search engine to find the right information
→ how quickly this is done

→ effectiveness and efficiency - cost of system conf.
→ efficiency and cost targets- effectiveness

Traditional Effectiveness Measures

Assumption :

- Given a user's information need, represented by a search query, each document in a given text collection is either relevant or irrelevant with respect to this information need.

- The relevance of a document d depends only on the information need and d itself. It is independent of the search engine's ranking of the other documents in the collection.

Traditional Effectiveness Measures

Evaluation of unranked retrieval sets :

Precision (P) is the fraction of retrieved documents that are relevant

$$\text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = P(\text{relevant}|\text{retrieved})$$

Recall (R) is the fraction of relevant documents that are retrieved

$$\text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = P(\text{retrieved}|\text{relevant})$$

	Relevant	Nonrelevant
Retrieved	true positives (tp)	false positives (fp)
Not retrieved	false negatives (fn)	true negatives (tn)

Then:

$$P = tp/(tp+fp)$$
$$R = tp/(tp+fn)$$

Accuracy = (tp + tn)/(tp + f p + f n + tn).

In almost all circumstances, the data is extremely skewed: normally over 99.9% of the documents are in the nonrelevant category.

F measure

A single measure that trades off precision versus recall is the F measure, which is the weighted harmonic mean of precision and recall:

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \quad \text{where} \quad \beta^2 = \frac{1-\alpha}{\alpha}$$

where $\alpha \in [0, 1]$ and thus $\beta 2 \in [0, \infty]$.

The default balanced F measure equally weights precision and recall, which means making $\alpha = 1/2$ or $\beta = 1$.

When using $\beta = 1$, the formula on the right simplifies to:

$$F_{\beta=1} = \frac{2PR}{P+R}$$

Exercise :

An IR system returns 8 relevant documents, and 10 nonrelevant documents. There are a total of 20 relevant documents in the collection. What is the precision of the system on this search, and what is its recall? Calculate value of F measure for β = 1.

$$P = \frac{8}{\quad} =$$

$$R = \frac{8}{20} =$$

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