

# Information Retrieval

## Topic- Index Compression

### (Term statistics)

## Lecture-20

**Prepared By**

Dr. Rasmita Rautray & Dr. Rasmita Dash  
Associate Professor  
Dept. of CSE

# Content

- Term statistics
- Heap's law
- Zipf's law

# Why compression? (in general)

- Use less disk space (saves money)
- Keep more stuff in memory (increases speed)
- Increase speed of transferring data from disk to memory  
(again, increases speed)
  - [read compressed data and decompress in memory] is faster than [read uncompressed data]
- Decompression algorithms are fast.

# Why compression in information retrieval?

- First, we will consider space for dictionary
  - Main motivation for dictionary compression: make it small enough to keep in main memory
- Then for the postings file
  - Motivation: reduce disk space needed, decrease time needed to read from disk

# Lossy vs. lossless compression

- Lossy compression: Discard some information.
- Lossless compression: All information is preserved.

# Term Statistics

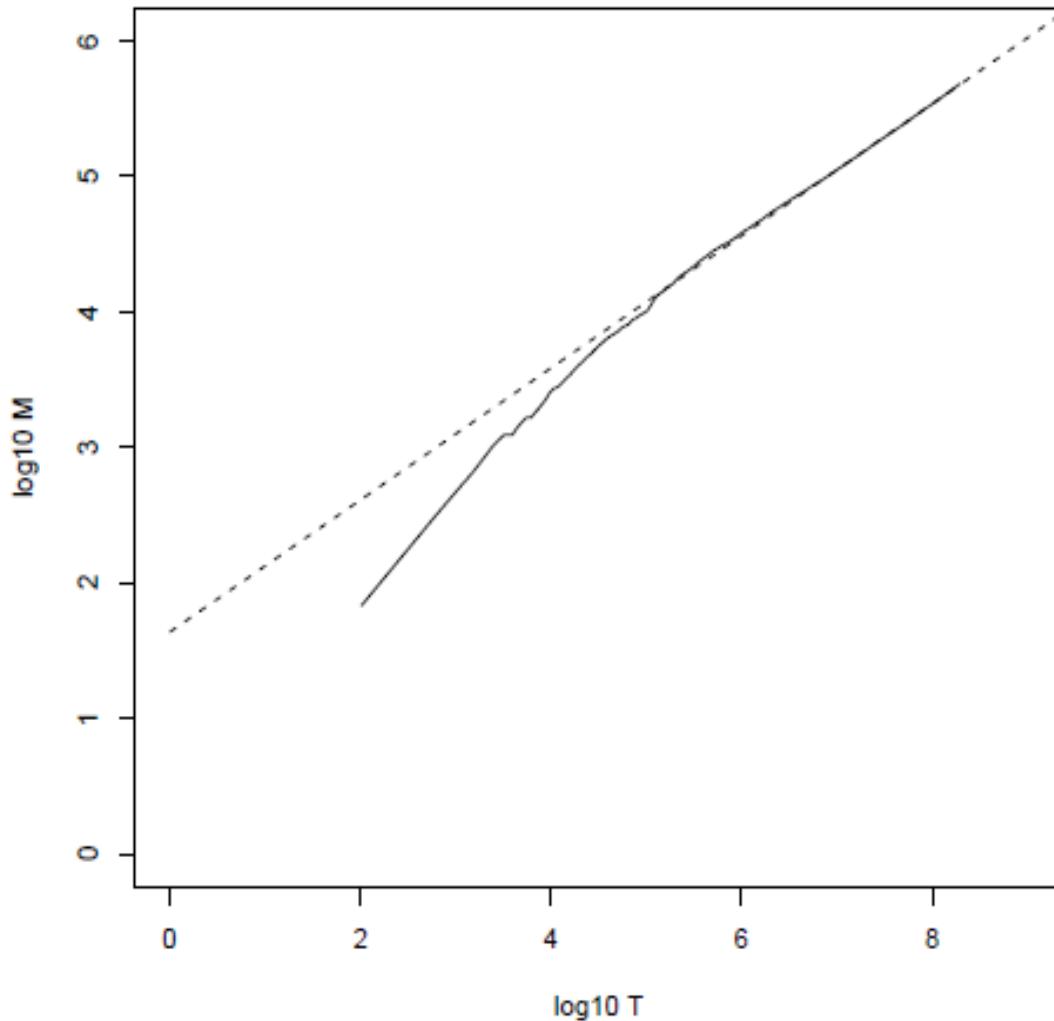
# How big is the term vocabulary?

- That is, how many distinct words are there?
- Can we assume there is an upper bound?
- Not really: At least  $7020 \approx 1037$  different words of length 20.
- The vocabulary will keep growing with collection size.

# Heaps' law

- Heaps' law:  $M = kT^b$
- M is the size of the vocabulary, T is the number of tokens in the collection.
- Typical values for the parameters k and b are:  $30 \leq k \leq 100$  and  $b \approx 0.5$ .
- Heaps' law is linear in log-log space.
  - It is the simplest possible relationship between collection size and vocabulary size in log-log space.
  - Empirical law

# Heaps' law for Reuters



Vocabulary size  $M$  as a function of collection size  $T$  (number of tokens) for Reuters-RCV1. For these data, the dashed line  $\log_{10} M = 0.49 * \log_{10} T + 1.64$  is the best least squares fit. Thus,  $M = 10^{1.64} T^{0.49}$  and  $k = 10^{1.64} \approx 44$  and  $b = 0.49$ .

# Empirical fit for Reuters

- Example: for the first 1,000,020 tokens Heaps' law predicts 38,323 terms:

$$44 \times 1000020^{0.49} \approx 38,323$$

- The actual number is 38,365 terms, very close to the prediction.
- Empirical observation: fit is good in general.

# Zipf's law

Zipf's law: The  $i^{\text{th}}$  most frequent term has frequency proportional to  $1/i$ .

$$cf_i \propto \frac{1}{i}$$

$cf$  is collection frequency: the number of occurrences of the term in the collection.

So if the most frequent term (*the*) occurs  $cf_1$  times, then the second most frequent term (*of*) has half as many occurrences  $cf_2 = \frac{1}{2}cf_1 \dots$

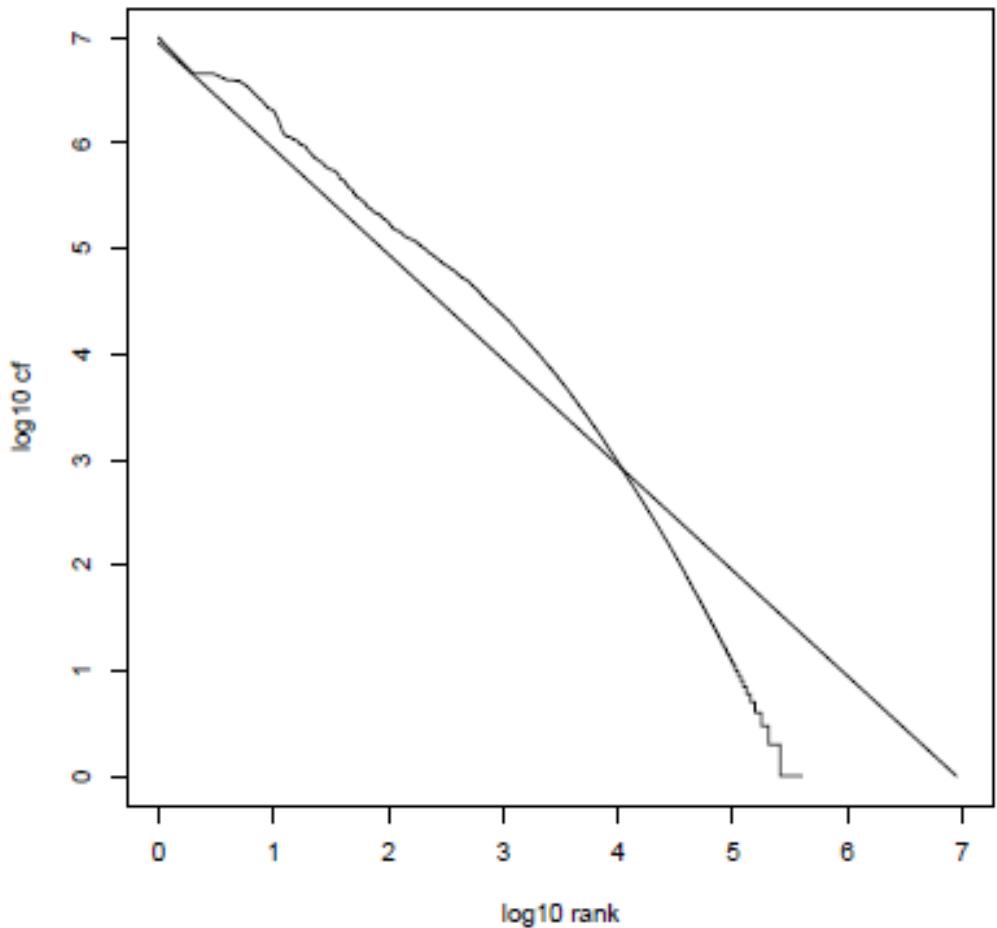
$\dots$  and the third most frequent term (*and*) has a third as many occurrences  $cf_3 = \frac{1}{3}cf_1$  etc.

Equivalent:  $cf_i = ci^k$  and  $\log cf_i = \log c + k \log i$  (for  $k = -1$ )

Example of a power law

# Zipf's law for Reuters

Fit is not great. What is important is the key insight: Few frequent terms, many rare terms.



Thank You

# Information Retrieval

## Topic: Index Compression (Part-1)

### Lecture-21

**Prepared By**

**Dr. Rasmrita Rautray & Dr. Rasmrita Dash**  
Associate Professor  
Dept. of CSE

# Dictionary compression

- The dictionary is small compared to the postings file.
- But we want to keep it in memory.
- Also: competition with other applications, cell phones,
- onboard computers, fast startup time
- So compressing the dictionary is important.

# Why compression in information retrieval?

- First, we will consider space for dictionary
  - Main motivation for dictionary compression: make it small enough to keep in main memory
- Then for the postings file
  - Motivation: reduce disk space needed, decrease time needed to read from disk

# Lossy vs. lossless compression

- Lossy compression: Discard some information Several of the preprocessing steps we frequently use can be viewed as lossy compression:
- Lossless compression: All information is preserved.

# Term Statistics

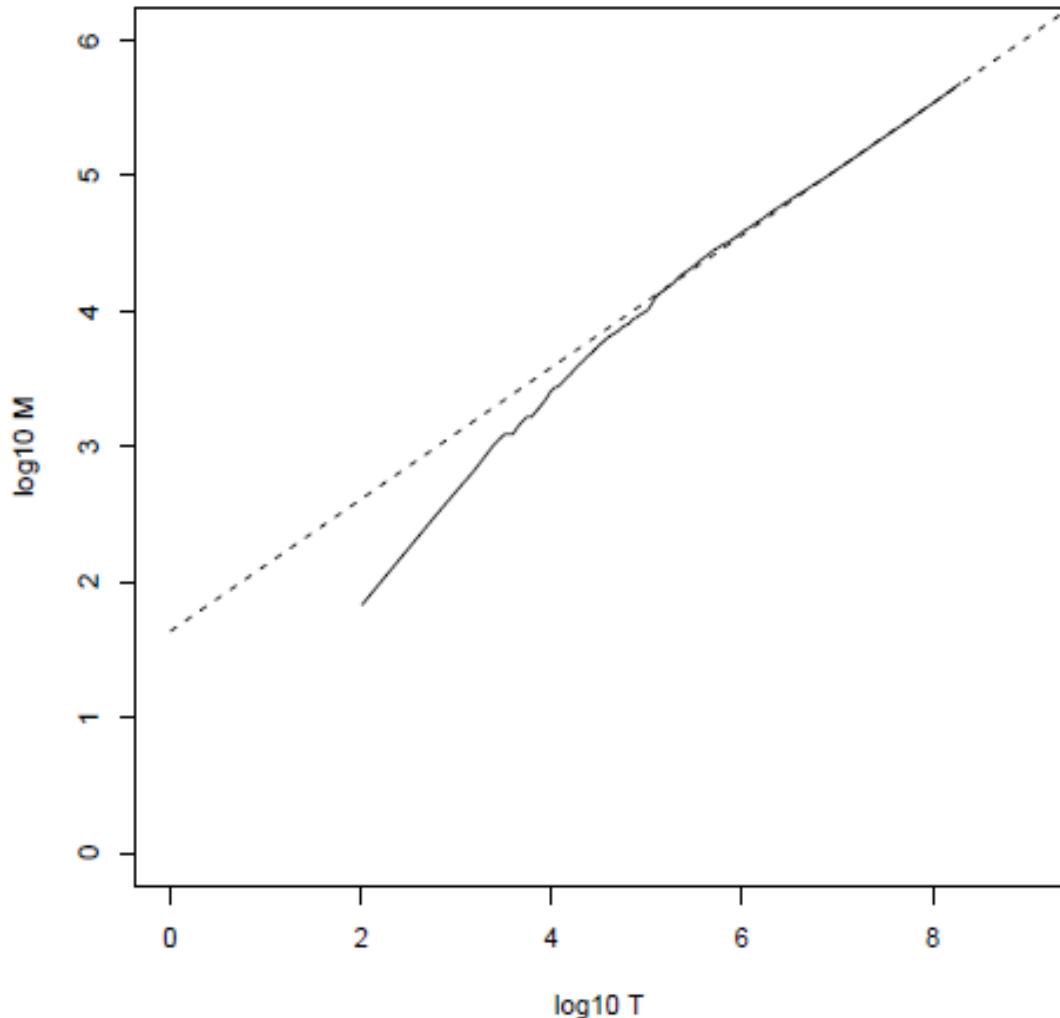
# How big is the term vocabulary?

- That is, how many distinct words are there?
- Can we assume there is an upper bound?
- Not really: At least  $7020 \approx 1037$  different words of length 20.
- The vocabulary will keep growing with collection size.

# Heaps' law

- Heaps' law:  $M = kT^b$
- M is the size of the vocabulary, T is the number of tokens in the collection.
- Typical values for the parameters k and b are:  $30 \leq k \leq 100$  and  $b \approx 0.5$ .
- Heaps' law is linear in log-log space.
  - It is the simplest possible relationship between collection size and vocabulary size in log-log space.
  - Empirical law

# Heaps' law for Reuters



Vocabulary size  $M$  as a function of collection size  $T$  (number of tokens) for Reuters-RCV1. For these data, the dashed line  $\log_{10} M = 0.49 * \log_{10} T + 1.64$  is the best least squares fit. Thus,  $M = 10^{1.64} T^{0.49}$  and  $k = 10^{1.64} \approx 44$  and  $b = 0.49$ .

# Empirical fit for Reuters

- Good, as we just saw in the graph.
- Example: for the first 1,000,020 tokens Heaps' law predicts 38,323 terms:

$$44 \times 1000020^{0.49} \approx 38,323$$

- The actual number is 38,365 terms, very close to the prediction.
- Empirical observation: fit is good in general.

# Zipf's law

Zipf's law: The  $i^{\text{th}}$  most frequent term has frequency proportional to  $1/i$ .

$$cf_i \propto \frac{1}{i}$$

$cf$  is collection frequency: the number of occurrences of the term in the collection.

So if the most frequent term (*the*) occurs  $cf_1$  times, then the second most frequent term (*of*) has half as many occurrences

$$cf_2 = \frac{1}{2}cf_1 \dots$$

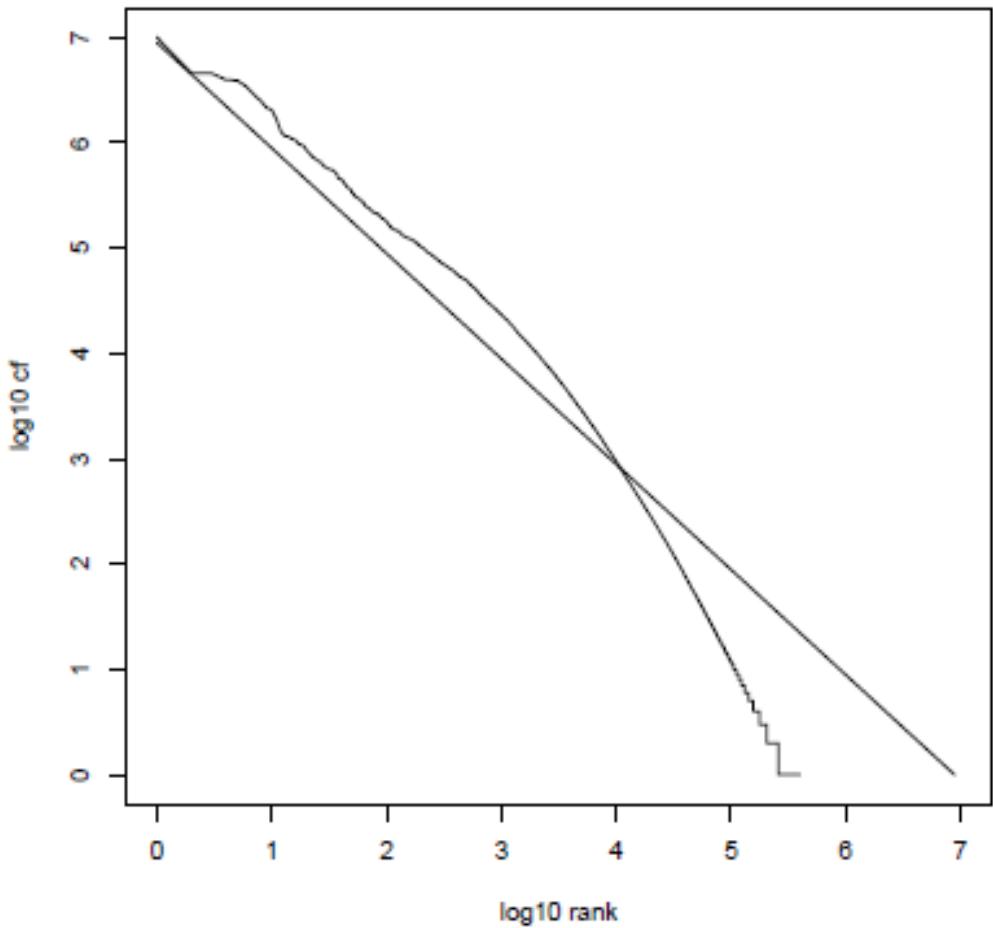
$\dots$  and the third most frequent term (*and*) has a third as many occurrences  $cf_3 = \frac{1}{3}cf_1$  etc.

Equivalent:  $cf_i = ci^k$  and  $\log cf_i = \log c + k \log i$  (for  $k = -1$ )

Example of a power law

# Zipf's law for Reuters

Fit is not great. What is important is the key insight: Few frequent terms, many rare terms.



# Dictionary compression

- The dictionary is small compared to the postings file.
- But we want to keep it in memory.
- Also: competition with other applications, cell phones, onboard computers, fast startup time
- So compressing the dictionary is important.

# Recall: Dictionary as array of fixed-width entries

term	document frequency	pointer to postings list	Space
a	656,265	→	
aachen	65	→	
...	...	...	
zulu	221	→	
space needed:	20 bytes	4 bytes	4 bytes

- for Reuters:  $(20+4+4)*400,000 = 11.2 \text{ MB}$

# Limitation of Fixed-width entries

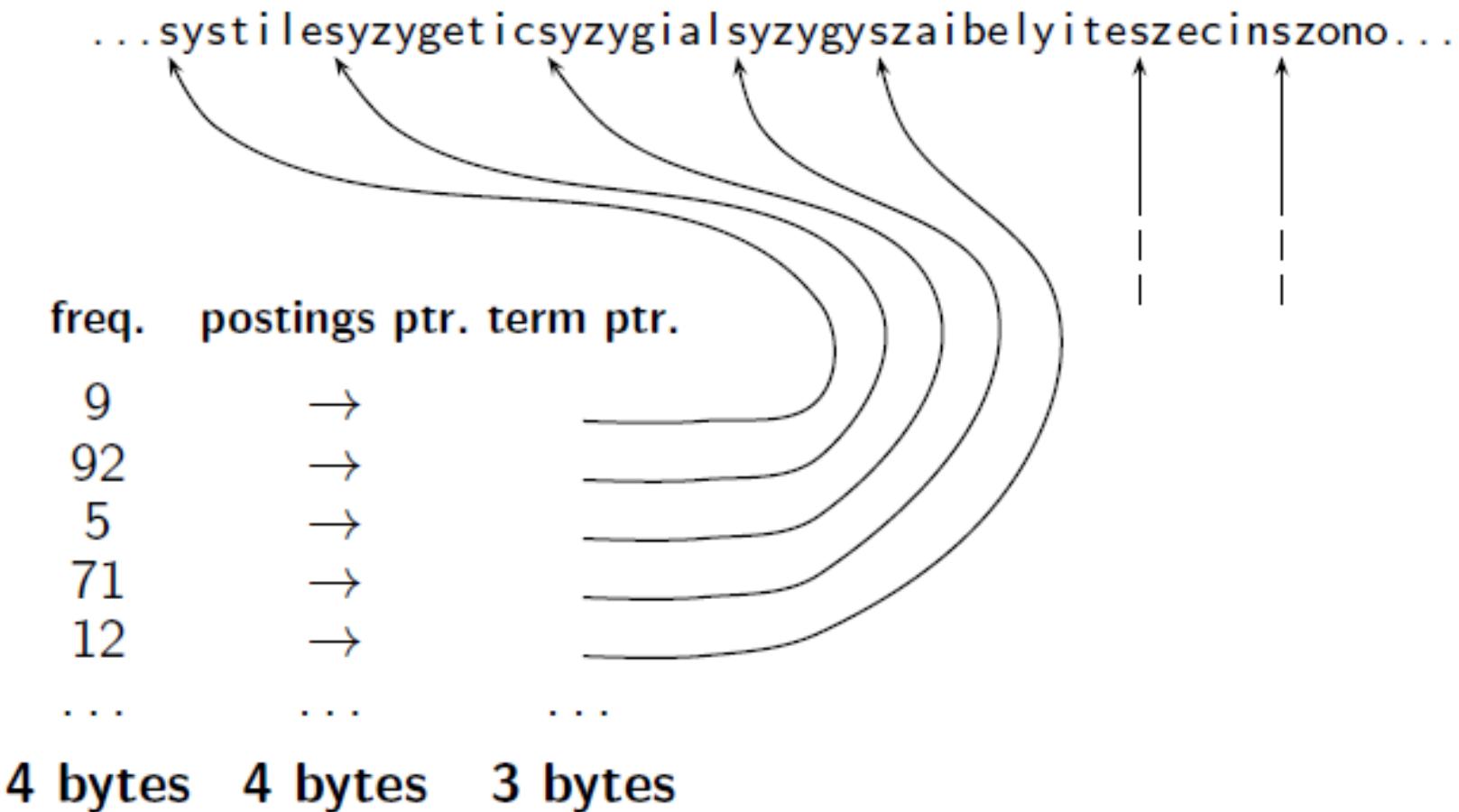
- Most of the bytes in the term column are wasted.
  - We allot 20 bytes for terms of length 1.
- We can't handle hydrochlorofluorocarbons and supercalifragilisticexpialidocious
- Average length of a term in English: 8 characters (or a little bit less)
- How can we use on average 8 characters per term?

# Dictionary as a string

- It stores the dictionary terms as one long string of characters.
- Term pointers mark the end of the preceding term and the beginning of the next.

For example, the first three terms in this example are systile, syzygetic, and syzygial

# Dictionary as a string



# Space for dictionary as a string

- 4 bytes per term for frequency
- 4 bytes per term for pointer to postings list
- 8 bytes (on average) for term in string
- 3 bytes per pointer into string  
(need  $\log_2 8 \times 400,000 < 24$  bits to resolve 8 · 400,000 positions)
- Space:  $400,000 \times (4 + 4 + 3 + 8) = 7.6\text{MB}$   
(compared to 11.2 MB for fixed-width array)

Thank You

# Information Retrieval

## Topic: Index Compression (Part-2)

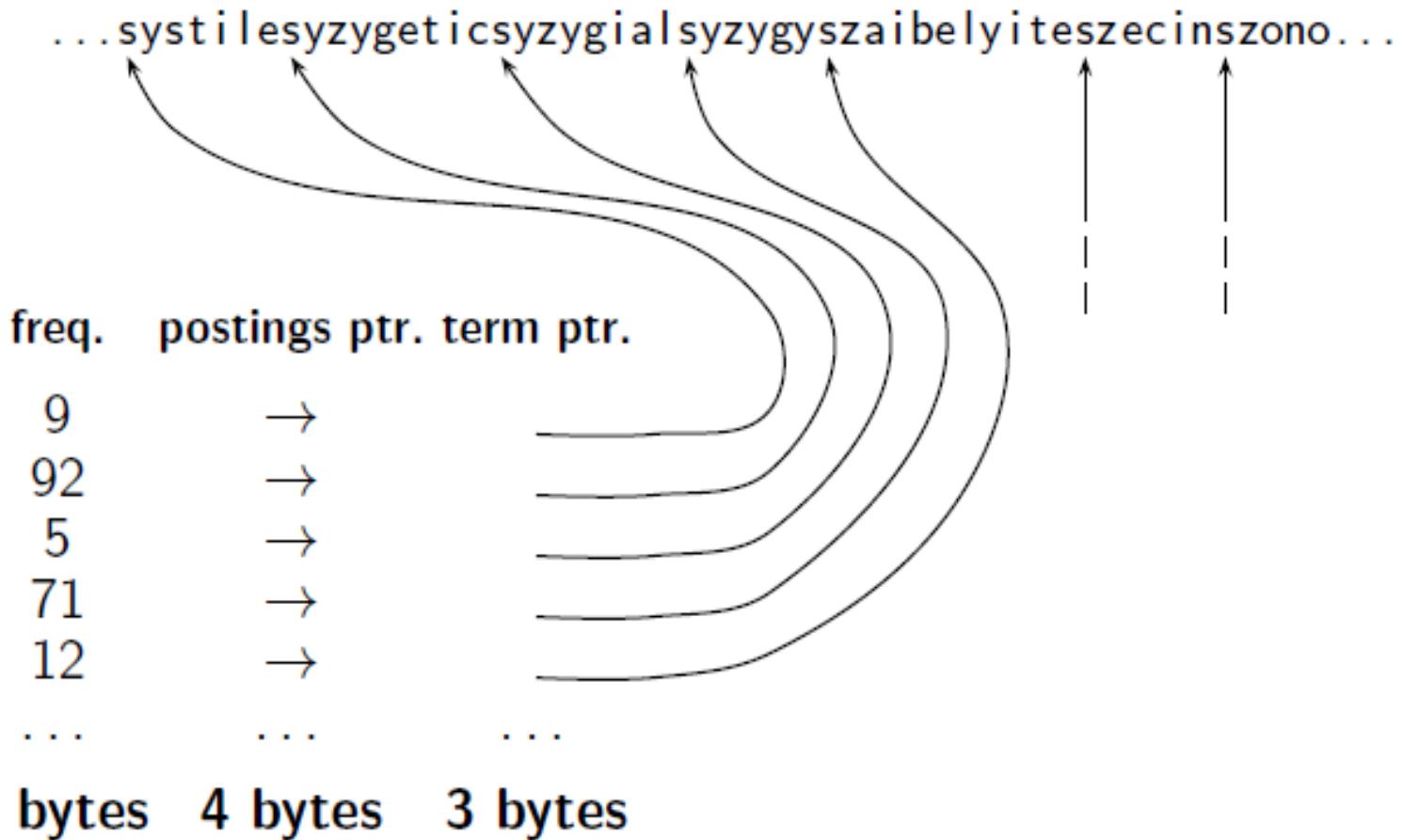
### Lecture-22

**Prepared By**

Dr. Rasmita Rautray & Dr. Rasmita Dash  
Associate Professor  
Dept. of CSE

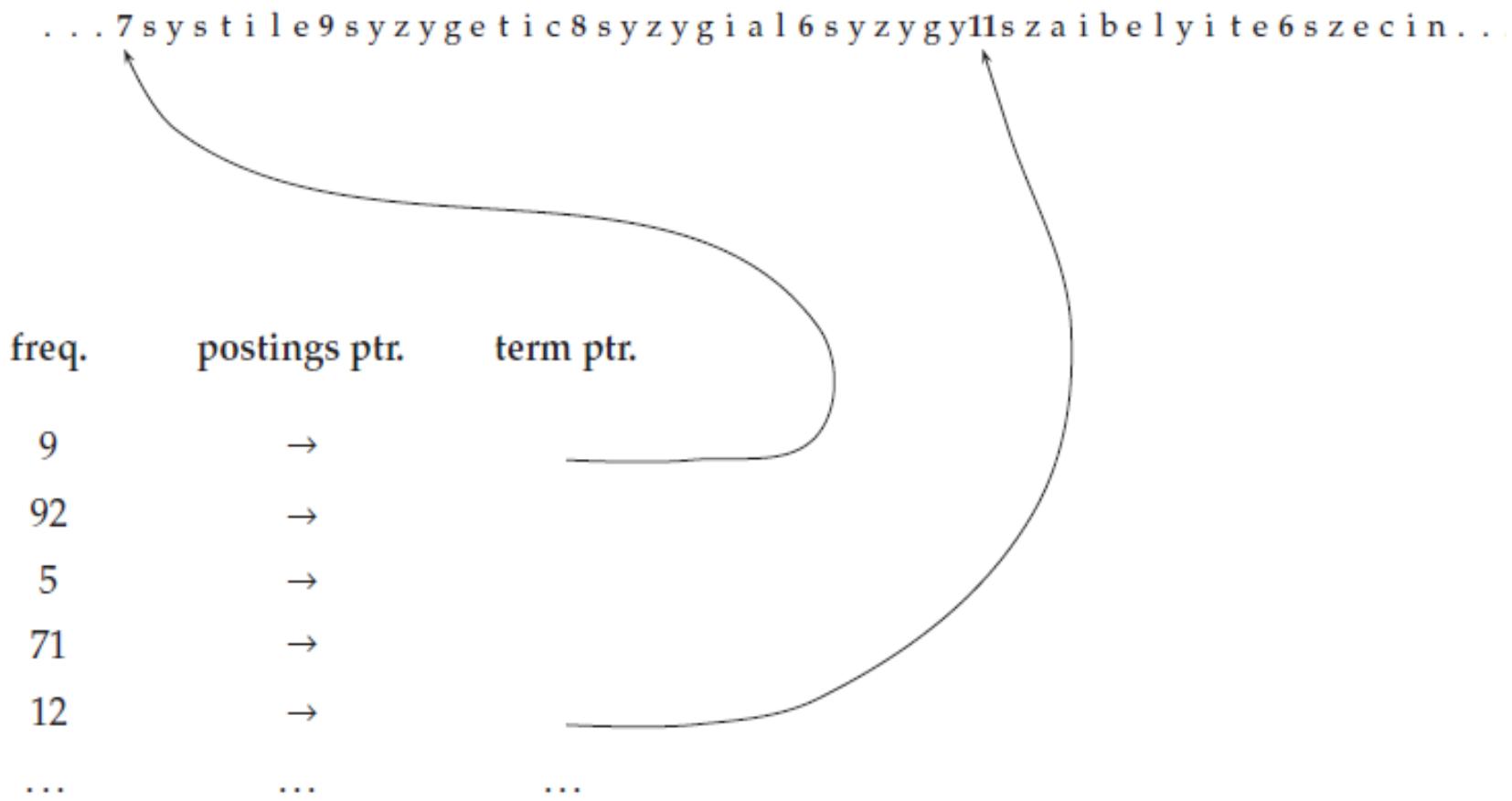


# Dictionary as a string



we need  $400,000 \times (4 + 4 + 3 + 8) = 7.6$  MB for the Reuters-RCV1 dictionary: 4 bytes each for frequency and postings pointer, 3 bytes for the term pointer, and 8 bytes on average for the term. So we have reduced the space requirements by one third from 11.2 to 7.6 MB

# Dictionary compression as Blocked storage



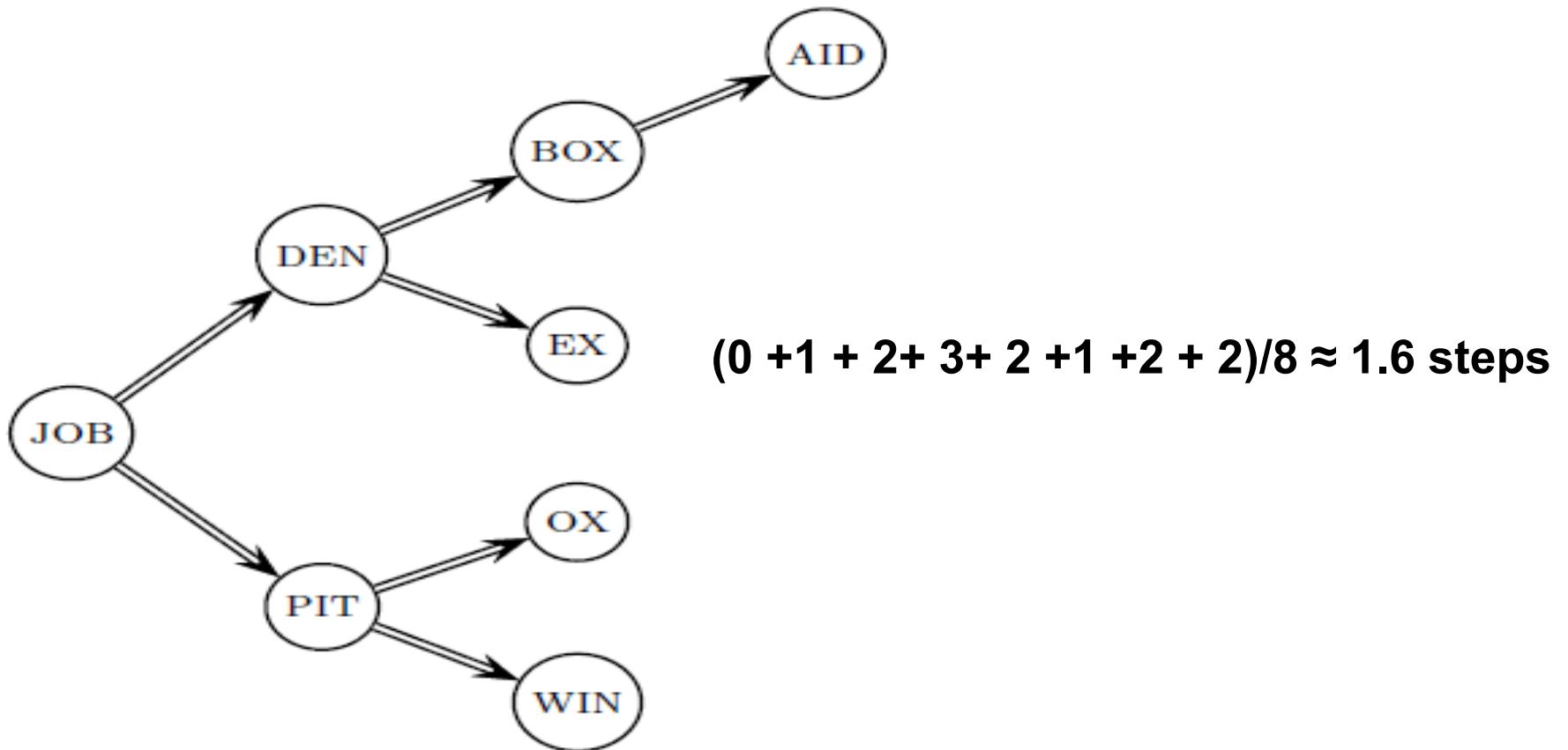
# Space for dictionary as a string with blocking

Example: block size  $k = 4$

- Where we used  $4 \times 3$  bytes for term pointers without blocking . . .
- . . .we now use 3 bytes for one pointer plus 4 bytes for indicating the length of each term.
- We save  $12 - (3 + 4) = 5$  bytes per block.
- Total savings:  $400,000/4 * 5 = 0.5$  MB
- This reduces the size of the dictionary from 7.6 MB to 7.1 MB.

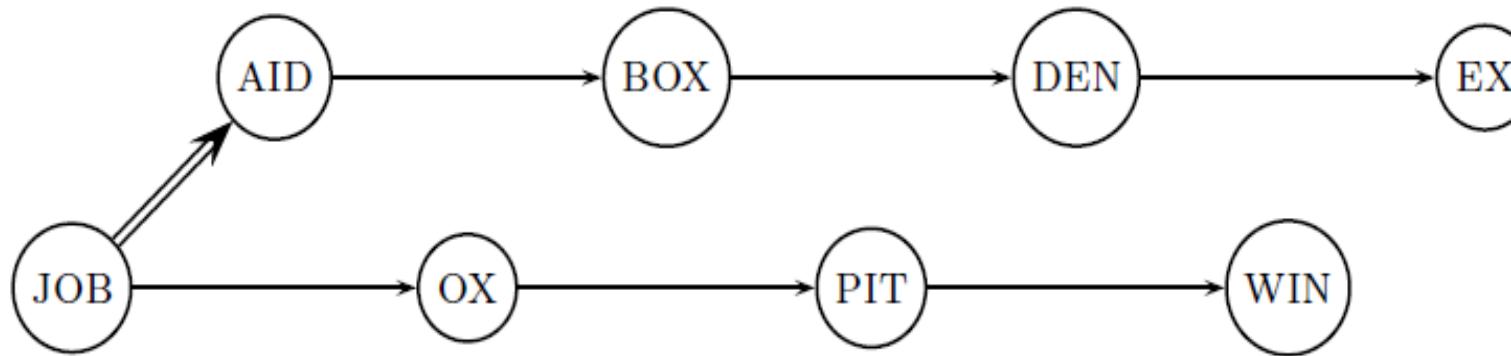
- By increasing the block size  $k$ , we get better compression.
- There is a tradeoff between compression and the speed of term lookup.

# Lookup of a term without blocking



Search of the uncompressed dictionary

# Lookup of a term with blocking: (slightly) slower



$$(0+1+2+3+4+1+2+3)/8 = 2 \text{ steps on average, } \approx 25\% \text{ more}$$

Dictionary compressed by blocking with  $k = 4$

# Front coding

- One block in blocked compression ( $k = 4$ ) . . .

8 automata 8 automate 9 automatic 10 automation



- . . . further compressed with front coding.

8 automat\*a 1 ◊ e 2 ◊ ic 3 ◊ ion

# Dictionary compression for Reuters: Summary

data structure	size in MB
dictionary, fixed-width	11.2
dictionary, term pointers into string	7.6
~, with blocking, $k = 4$	7.1
~, with blocking & front coding	5.9

Thank You

# Information Retrieval

## Topic- Scoring, Term Weighting, The Vector Space Model (Term frequency and weighting)

### Lecture-23

**Prepared By**

Dr. Rasmita Rautray & Dr. Rasmita Dash  
Associate Professor  
Dept. of CSE

# Content

- Ranked retrieval
- Jaccard coefficient
- Term frequency

# Ranked retrieval

- Ranking search results: why it is important (as opposed to just presenting a set of unordered Boolean results)
- Thus far, our queries have been Boolean.
  - Documents either match or don't.
- Good for expert users with precise understanding of their needs and of the collection.
- Also good for applications: Applications can easily consume 1000s of results.
- Not good for the majority of users
- Most users are not capable of writing Boolean queries . . .
  - . . . or they are, but they think it's too much work.
- Most users don't want to wade through 1000s of results.
- This is particularly true of web search.

# Problem with Boolean search: Feast or famine

- Boolean queries often result in either too few ( $=0$ ) or too many (1000s) results.
- Query 1 (boolean conjunction):  
→ 200,000 hits – feast
- Query 2 (boolean conjunction): [no card found]  
→ 0 hits – famine
- In Boolean retrieval, it takes a lot of skill to come up with a query that produces a manageable number of hits.

# Feast or famine: No problem in ranked retrieval

- With ranking, large result sets are not an issue.
- Just show the top 10 results
- The ranking algorithm works: More relevant results are ranked higher than less relevant results.

# Scoring as the basis of ranked retrieval

- How can we accomplish a relevance ranking of the documents with respect to a query?
- Assign a score to each query-document pair, say in  $[0, 1]$ .
- This score measures how well document and query “match”.
- Sort documents according to scores

# Query-document matching scores

- How do we compute the score of a query-document pair?
- If no query term occurs in the document: score should be 0.
- The more frequent a query term in the document, the higher the score
- The more query terms occur in the document, the higher the score
- We will look at a number of alternatives for doing this.

# Jaccard coefficient

- A commonly used measure of overlap of two sets
- Let A and B be two sets
- Jaccard coefficient:  $\text{jaccard}(A,B) = |A \cap B| / |A \cup B|$   
( $A \neq \emptyset$  or  $B \neq \emptyset$ )
- $\text{jaccard}(A,A) = 1$
- $\text{jaccard}(A,B) = 0$  if  $A \cap B = 0$
- A and B don't have to be the same size.
- Always assigns a number between 0 and 1.

# Jaccard coefficient: Example

- What is the query-document match score that the Jaccard coefficient computes for:
  - Query: “ides of March”
  - Document “Caesar died in March”
  - $\text{jaccard}(q, d) = 1/6$

# What's wrong with Jaccard?

- It doesn't consider term frequency (how many occurrences a term has).
- Rare terms are more informative than frequent terms. Jaccard does not consider this information.
- We need a more sophisticated way of normalizing for the length of a document.

# Term frequency

- This is a key ingredient for ranking

# Binary incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	1	1	0	0	0	1	
BRUTUS	1	1	0	1	0	0	
CAESAR	1	1	0	1	1	1	
CALPURNIA	0	1	0	0	0	0	
CLEOPATRA	1	0	0	0	0	0	
MERCY	1	0	1	1	1	1	
WORSER	1	0	1	1	1	0	
...							

Each document is represented as a binary vector  $\in \{0, 1\}^{|V|}$ .

# Count matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	157	73	0	0	0	1	
BRUTUS	4	157	0	2	0	0	
CAESAR	232	227	0	2	1	0	
CALPURNIA	0	10	0	0	0	0	
CLEOPATRA	57	0	0	0	0	0	
MERCY	2	0	3	8	5	8	
WORSER	2	0	1	1	1	5	
...							

Each document is now represented as a count vector  $\in \mathbb{N}^{|V|}$ .

# Bag of words model

- The exact ordering of the terms in a document is ignored but the number of occurrences of each term is material.
- Only retain information on the number of occurrences of each term
  - John is quicker than Mary and Mary is quicker than John are represented the same way.
- This is called a bag of words model.
- In a sense, this is a step back: The positional index was able to distinguish these two documents.

# Term frequency (tf)

- The term frequency  $tf_{t,d}$  of term t in document d is defined as the number of times that t occurs in d.
- We want to use  $tf$  when computing query-document match scores.
  - But how?
- Raw term frequency is not what we want because:
- A document with  $tf = 10$  occurrences of the term is more relevant than a document with  $tf = 1$  occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

# Instead of raw frequency: Log frequency weighting

- Score for a document-query pair: sum over terms t in both q

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

- $tf_{t,d} \rightarrow w_{t,d}$  :
- $0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4$ , etc.
- Score for a document-query pair: sum over terms t in both q and d:
- $\text{tf-matching-score}(q, d) = \sum_{t \in q \cap d} (1 + \log tf_{t,d})$
- The score is 0 if none of the query terms is present in the document.

# Example

Query: “best car insurance”

Document: “car insurance auto insurance”

words	Query		Document	
	tf-raw	tf-wt	tf-raw	tf-wt
auto	0	0	1	1
best	1	1	0	0
car	1	1	1	1
insurance	1	1	2	1.3

Thank You

# Information Retrieval

## Topic- Scoring, Term Weighting, The Vector Space Model (tf-idf weighting)

### Lecture-24

**Prepared By**

Dr. Rasmita Rautray & Dr. Rasmita Dash  
Associate Professor  
Dept. of CSE

# Content

- Collection frequency
- Document frequency
- tf-idf weighting

# Frequency in document vs. frequency in collection

- In addition, to term frequency (the frequency of the term in the document) . . .
- . . . we also want to use the frequency of the term in the collection for weighting and ranking.

# Desired weight for rare terms

- Rare terms are more informative than frequent terms.
- Consider a term in the query that is rare in the collection (e.g., arachnocentric).
- A document containing this term is very likely to be relevant.  
→ We want high weights for rare terms like arachnocentric.

# Desired weight for frequent terms

- Frequent terms are less informative than rare terms.
- Consider a term in the query that is frequent in the collection (e.g., GOOD, INCREASE, LINE).
- A document containing this term is more likely to be relevant than a document that doesn't . . .
- . . . but words like GOOD, INCREASE and LINE are not sure indicators of relevance.
- → For frequent terms like GOOD, INCREASE, and LINE, we want positive weights . . .
- . . . but lower weights than for rare terms.

# Document frequency

- We want high weights for rare terms like ARACHNOCENTRIC.
- We want low (positive) weights for frequent words like GOOD, INCREASE, and LINE.
- We will use document frequency to factor this into computing the matching score.
- The document frequency is the number of documents in the collection that the term occurs in.

# idf weight

- $\text{df}_t$  is the document frequency, the number of documents that  $t$  occurs in.
- $\text{df}_t$  is an inverse measure of the informativeness of term  $t$ .
- We define the idf weight of term  $t$  as follows:
- $\text{idf}_t = \log_{10} (N/\text{df}_t)$   
( $N$  is the number of documents in the collection.)
- $\text{idf}_t$  is a measure of the informativeness of the term.
- $[\log N/\text{df}_t]$  instead of  $[N/\text{df}_t]$  to “dampen” the effect of idf
- Note that we use the log transformation for both term frequency and document frequency.

# Examples for idf

- Compute  $\text{idf}_t$  using the formula:  $\text{idf}_t = \log_{10} (1,000,000 / \text{df}_t)$

term	$\text{df}_t$	$\text{idf}_t$
calpurnia	1	6
animal	100	4
sunday	1000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

# Effect of idf on ranking

- idf affects the ranking of documents for queries with at least two terms.
- For example, in the query “arachnocentric line”, idf weighting increases the relative weight of ARACHNOCENTRIC and decreases the relative weight of LINE.
- idf has little effect on ranking for one-term queries.

# Collection frequency vs. Document frequency

- Collection frequency of  $t$ : number of tokens of  $t$  in the collection
- Document frequency of  $t$ : number of documents  $t$  occurs in

Word	cf	df
try	10422	8760
Insurance	10440	3997

# tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$w_{t,d} = (1 + \log \text{tf}_{t,d}) \cdot \log (N / \text{df}_t)$$

- tf-weight
- idf-weight
- Best known weighting scheme in information retrieval
- Alternative names: tf.idf is tf x idf

# tf-idf

- Assign a tf-idf weight for each term  $t$  in each document  $d$ :

$$w_{t,d} = (1 + \log \text{tf}_{t,d}) \cdot \log (N / \text{df}_t)$$

- The tf-idf weight . . .
  - . . . increases with the number of occurrences within a document. (term frequency)
  - . . . increases with the rarity of the term in the collection. (inverse document frequency)

# Term, collection and document frequency

Quantity	Symbol	Definition
term frequency	$\text{tf}_{t,d}$	number of occurrences of $t$ in $d$
document frequency	$\text{df}_t$	number of documents in the collection that $t$ occurs in
collection frequency	$\text{Cf}_t$	total number of occurrences of $t$ in the collection

Consider the 1<sup>st</sup> table of term frequencies for 3 documents denoted Doc1, Doc2, Doc3. Compute the tf-idf weights for the terms car, auto, insurance, best, for each document, using the idf values given below.

	Doc1	Doc2	Doc3
car	27	4	24
auto	3	33	0
insurance	0	33	29
best	14	0	17

Term frequencies for 3 documents

term	df <sub>t</sub>	idf <sub>t</sub>
car	18,165	1.65
auto	6723	2.08
insurance	19,241	1.62
best	25,235	1.5

idf's of terms with various frequencies in the Reuters collection of 806,791 documents

Thank You

# Information Retrieval

## Topic- Scoring, Term Weighting, The Vector Space Model (Vector Space Model)

### Lecture-25

**Prepared By**

Dr. Rasmita Rautray & Dr. Rasmita Dash  
Associate Professor  
Dept. of CSE

# Content

- Vector space model
- Similarity measure
- Cosine similarity

# Binary incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	1	1	0	0	0	1	
BRUTUS	1	1	0	1	0	0	
CAESAR	1	1	0	1	1	1	
CALPURNIA	0	1	0	0	0	0	
CLEOPATRA	1	0	0	0	0	0	
MERCY	1	0	1	1	1	1	
WORSER	1	0	1	1	1	0	
...							

Each document is represented as a binary vector  $\in \{0, 1\}^{|V|}$ .

# Count matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	157	73	0	0	0	1	
BRUTUS	4	157	0	2	0	0	
CAESAR	232	227	0	2	1	0	
CALPURNIA	0	10	0	0	0	0	
CLEOPATRA	57	0	0	0	0	0	
MERCY	2	0	3	8	5	8	
WORSER	2	0	1	1	1	5	
...							

Each document is now represented as a count vector  $\in \mathbb{N}^{|V|}$ .

# Binary → count → weight matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	5.25	3.18	0.0	0.0	0.0	0.35	
BRUTUS	1.21	6.10	0.0	1.0	0.0	0.0	
CAESAR	8.59	2.54	0.0	1.51	0.25	0.0	
CALPURNIA	0.0	1.54	0.0	0.0	0.0	0.0	
CLEOPATRA	2.85	0.0	0.0	0.0	0.0	0.0	
MERCY	1.51	0.0	1.90	0.12	5.25	0.88	
WORSER	1.37	0.0	0.11	4.15	0.25	1.95	
...							

- Each document is now represented as a real-valued vector of tf-idf weights  $\in \mathbb{R}^{|V|}$ .

# Documents as vectors

- Each document is now represented as a real-valued vector of tf-idf weights  $\in \mathbb{R}^{|V|}$ .
- So we have a  $|V|$ -dimensional real-valued vector space.
- Terms are axes of the space.
- Documents are points or vectors in this space.
- Very high-dimensional: tens of millions of dimensions when you apply this to web search engines
- Each vector is very sparse - most entries are zero.

# Queries as vectors

- Key idea 1: do the same for queries: represent them as vectors in the high-dimensional space
- Key idea 2: Rank documents according to their proximity to the query
- proximity = similarity
- proximity  $\approx$  negative distance
- Recall: We're doing this because we want to get away from the you're-either-in-or-out, feast-or-famine Boolean model.
- Instead: rank relevant documents higher than nonrelevant documents

# How do we formalize vector space similarity?

- First cut: (negative) distance between two points
- ( $=$  distance between the end points of the two vectors)
- Euclidean distance?
- Euclidean distance is a bad idea . . .
- . . . because Euclidean distance is large for vectors of different lengths.

# Use angle instead of distance

- Rank documents according to angle with query
- Thought experiment: take a document  $d$  and append it to itself. Call this document  $d'$ .  $d'$  is twice as long as  $d$ .
- “Semantically”  $d$  and  $d'$  have the same content.
- The angle between the two documents is 0, corresponding to maximal similarity . . .
- . . . even though the Euclidean distance between the two documents can be quite large.

# From angles to cosines

- The following two notions are equivalent.
  - Rank documents according to the angle between query and document in decreasing order
  - Rank documents according to  $\cos(\text{query}, \text{document})$  in increasing order
- Cosine is a monotonically decreasing function of the angle for the interval  $[0^\circ, 180^\circ]$

# Cosine similarity between query and document

$$\cos(\vec{q}, \vec{d}) = \text{SIM}(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}| |\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

- $q_i$  is the tf-idf weight of term  $i$  in the query.
- $d_i$  is the tf-idf weight of term  $i$  in the document.
- $|\vec{q}|$  and  $|\vec{d}|$  are the lengths of  $\vec{q}$  and  $\vec{d}$ .
- This is the cosine similarity of  $\vec{q}$  and  $\vec{d}$ . . . . or, equivalently, the cosine of the angle between  $\vec{q}$  and  $\vec{d}$

# Cosine for normalized vectors

- For normalized vectors, the cosine is equivalent to the dot product or scalar product.

$$\cos(\vec{q}, \vec{d}) = \vec{q} \cdot \vec{d} = \sum_i q_i d_i$$

– (if  $\vec{q}$  and  $\vec{d}$  are length-normalized).

# Cosine: Example

How similar are these novels?

SaS: Sense and Sensibility

PaP: Pride and Prejudice

WH: Wuthering Heights

Term frequencies (counts)

term	SaS	PaP	WH
AFFECTION	115	58	20
JEALOUS	10	7	11
GOSSIP	2	0	6
WUTHERING	0	0	38

# Cosine: Example

Term frequencies (counts)

term	SaS	PaP	WH
AFFECTION	115	58	20
JEALOUS	10	7	11
GOSSIP	2	0	6
WUTHERING	0	0	38

log frequency weighting

term	SaS	PaP	WH
AFFECTION	3.06	2.76	2.30
JEALOUS	2.0	1.85	2.04
GOSSIP	1.30	0	1.78
WUTHERING	0	0	2.58

(To simplify this example, we don't do idf weighting.)

# Cosine: Example

log frequency weighting

term	SaS	PaP	WH
AFFECTION	3.06	2.76	2.30
JEALOUS	2.0	1.85	2.04
GOSSIP	1.30	0	1.78
WUTHERING	0	0	2.58

log frequency weighting  
& cosine normalization

term	SaS	PaP	WH
AFFECTION	0.789	0.832	0.524
JEALOUS	0.515	0.555	0.465
GOSSIP	0.335	0.0	0.405
WUTHERING	0.0	0.0	0.588

$$\cos(\text{SaS}, \text{PaP}) \approx$$

$$0.789 * 0.832 + 0.515 * 0.555 + 0.335 * 0.0 + 0.0 * 0.0 \approx 0.94.$$

$$\cos(\text{SaS}, \text{WH}) \approx 0.79$$

$$\cos(\text{PaP}, \text{WH}) \approx 0.69$$

# Computing the cosine score

CosineScore( $q$ )

- 1 float Scores[ $N$ ] = 0
- 2 float Length[ $N$ ]
- 3 for each query term  $t$
- 4 do calculate  $w_{t,q}$  and fetch postings list for  $t$
- 5 for each pair( $d$ ,  $tf_{t,d}$ ) in postings list
- 6 do Scores[ $d$ ]+ =  $w_{t,d} \times w_{t,q}$
- 7 Read the array Length
- 8 for each  $d$
- 9 do Scores[ $d$ ] = Scores[ $d$ ]/Length[ $d$ ]
- 10 return Top  $K$  components of Scores[]

# Summary: Ranked retrieval in the vector space model

- Represent the query as a weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Compute the cosine similarity between the query vector and each document vector
- Rank documents with respect to the query
- Return the top K (e.g.,  $K = 10$ ) to the user

Thank You

# Information Retrieval

## Topic- The complete search system

### Lecture-26

**Prepared By**

**Dr. Rasmita Rautray & Dr. Rasmita Dash**  
Associate Professor  
Dept. of CSE

# Content

- Vector space model
- Similarity measure
- Cosine similarity

- Term frequency weight
- Inverse document frequency weight
- Tf-idf weight
- Cosine similarity

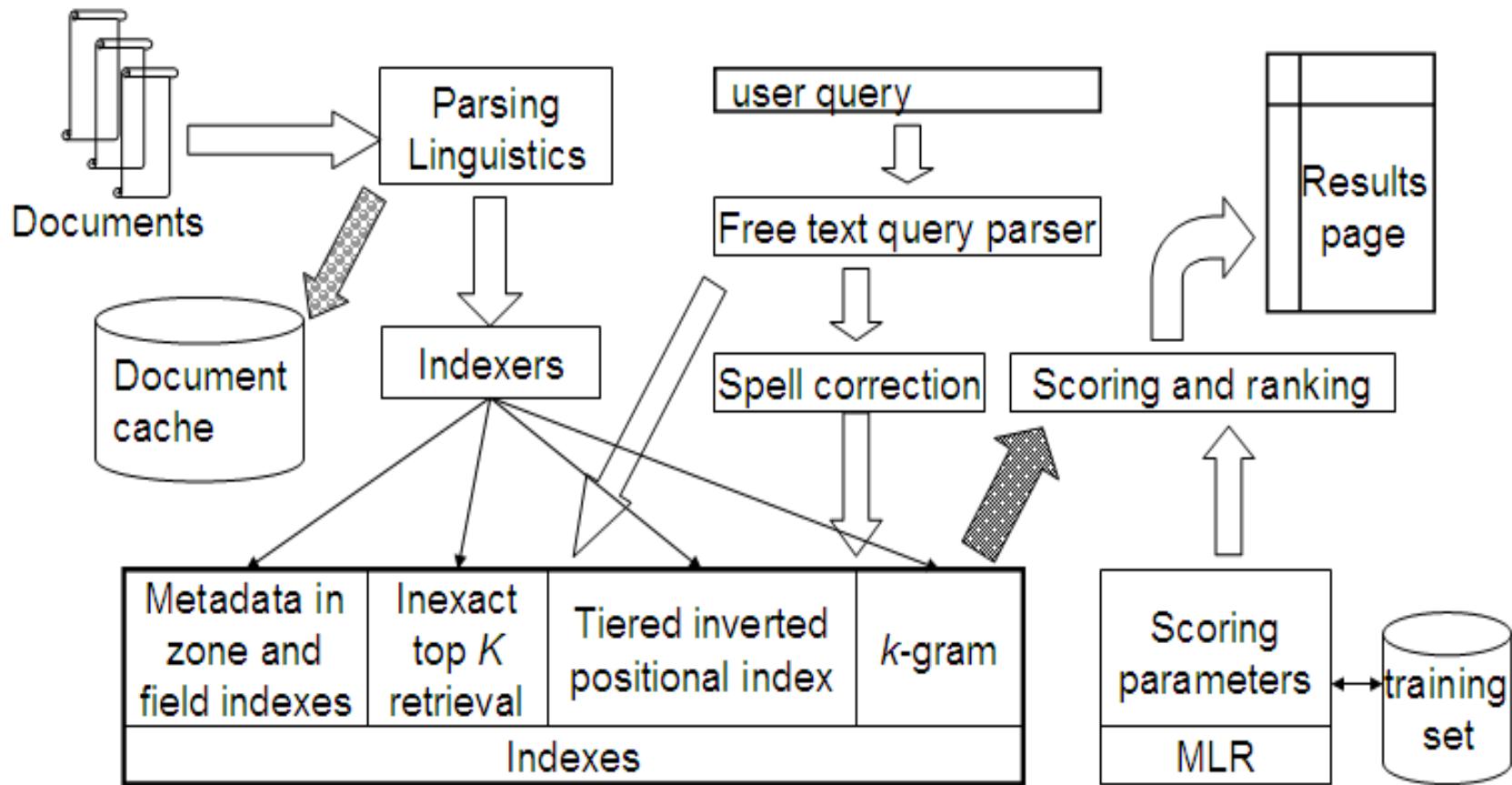
# Why is ranking so important?

- Last lecture: Problems with unranked retrieval
  - Users want to look at a few results – not thousands.
  - It's very hard to write queries that produce a few results.
  - Even for expert searchers
    - Ranking is important because it effectively reduces a large set of results to a very small one.
- Next: More data on “users only look at a few results”

# Importance of ranking: Summary

- Viewing abstracts: Users are a lot more likely to read the abstracts of the top-ranked pages (1, 2, 3, 4) than the abstracts of the lower ranked pages (7, 8, 9, 10).
- Clicking: Distribution is even more skewed for clicking
- In 1 out of 2 cases, users click on the top-ranked page.
- Even if the top-ranked page is not relevant, 30% of users will click on it.
  - Getting the ranking right is very important.
  - Getting the top-ranked page right is most important.

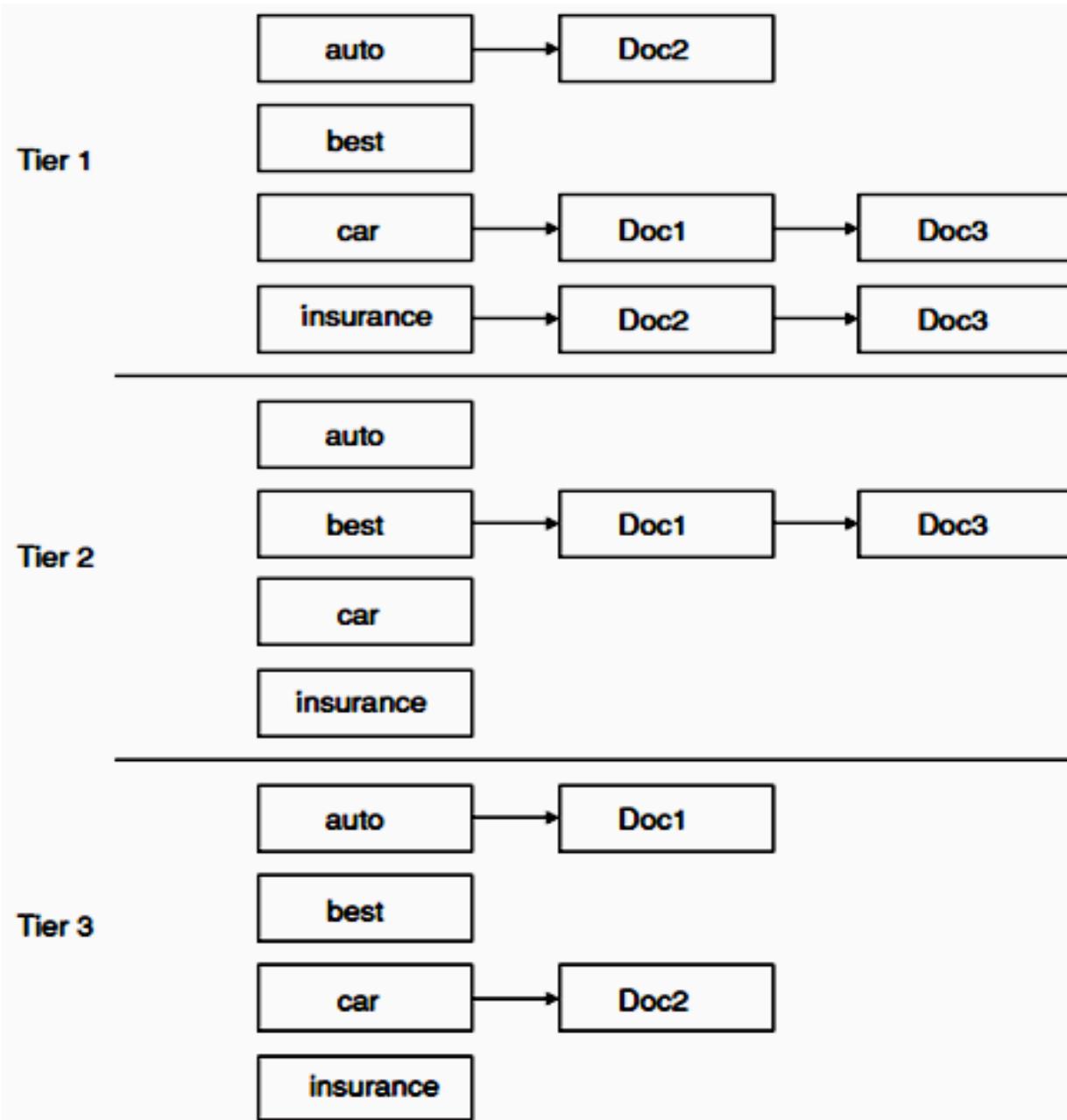
# Complete search system



# Tiered indexes

- Basic idea:
  - Create several tiers of indexes, corresponding to importance of indexing terms
  - During query processing, start with highest-tier index
  - If highest-tier index returns at least  $k$  (e.g.,  $k = 100$ ) results: stop and return results to user
  - If we've only found  $< k$  hits: repeat for next index in tier cascade
- Example: two-tier system
  - Tier 1: Index of all titles
  - Tier 2: Index of the rest of documents
  - Pages containing the search words in the title are better hits than pages containing the search words in the body of the text.

# Tiered index



# Components we have introduced thus far

- Document preprocessing (linguistic and otherwise)
- Positional indexes
- Tiered indexes
- Spelling correction
- k-gram indexes for wildcard queries and spelling correction
- Query processing
- Document scoring

# Components we haven't covered yet

- Document cache: we need this for generating snippets (= dynamic summaries)
- Zone indexes: They separate the indexes for different zones: the body of the document, all highlighted text in the document, anchor text, text in metadata fields etc
- Machine-learned ranking functions
- Proximity ranking (e.g., rank documents in which the query terms occur in the same local window higher than documents in which the query terms occur far from each other)
- Query parser

Thank You

# Information Retrieval

## Topic- Evaluation in information Retrieval (unranked evaluation)

### Lecture-27

**Prepared By**

Dr. Rasmrita Rautray & Dr. Rasmrita Dash  
Associate Professor  
Dept. of CSE

# Evaluation in information Retrieval

# Content

- Introduction to evaluation: Measures of an IR system
- Evaluation of unranked and ranked retrieval
- Evaluation benchmarks
- Result summaries

# Measures for a search engine

- How fast does it index
  - e.g., number of bytes per hour
- How fast does it search
  - e.g., latency as a function of queries per second
- What is the cost per query?
  - in dollars

# Measures for a search engine

- All of the preceding criteria are measurable: we can quantify
- speed / size / money
- However, the key measure for a search engine is user
- happiness.
- What is user happiness?
- How can we quantify user happiness?

# Who is the user?

- Who is the user we are trying to make happy?
- Web search engine: **searcher**. Success: Searcher finds what she was looking for. Measure: rate of return to this search engine
- Web search engine: **advertiser**. Success: Searcher clicks on ad. Measure: click through rate
- Ecommerce: **buyer**. Success: Buyer buys something. Measures: time to purchase, fraction of “conversions” of searchers to buyers
- Ecommerce: **seller**. Success: Seller sells something. Measure: profit per item sold
- Enterprise: **CEO**. Success: Employees are more productive  
• (because of effective search). Measure: profit of the

# Most Common definition of user happiness: Relevance

- User happiness is equated with the relevance of search results to the query.
- But how do you measure relevance?
- Standard methodology in information retrieval consists of three elements.
  - A benchmark document collection
  - A benchmark suite of queries
  - An assessment of the relevance of each query-document pair

# Relevance: query vs. information need

- Relevance to what?
- First take: relevance to the query
- “Relevance to the query” is very problematic.
- Information need i : “I am looking for information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine.”
- This is an information need, not a query.
- Query q: [red wine white wine heart attack]
- Consider document d': At the heart of his speech was an attack on the wine industry lobby for downplaying the role of red and white wine in drunk driving.
- d' is an excellent match for query q . . .
- d' is not relevant to the information need i

# Relevance: query vs. information need

- User happiness can only be measured by relevance to an information need, not by relevance to queries.
- Our terminology is sloppy in these slides and in IIR: we talk about query-document relevance judgments even though we mean information-need-document relevance judgments.

# Unranked evaluation

# Precision and recall

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

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

# Precision and recall

	Relevant	Nonrelevant
Retrieved	true positives (TP)	false positives (FP)
Not retrieved	false negatives (FN)	true negatives (TN)

$$P = TP / (TP + FP)$$

$$R = TP / (TP + FN)$$

# Precision/recall tradeoff

- You can increase recall by returning more docs.
- Recall is a non-decreasing function of the number of docs retrieved.
- A system that returns all docs has 100% recall!
- The converse is also true (usually): It's easy to get high precision for very low recall.
- Suppose the document with the largest score is relevant. How can we maximize precision?

# Example for precision, recall, F1

	relevant	not relevant	
retrieved	20	40	60
not retrieved	60	1,000,000	1,000,06
	80	1,000,040	1,000,12

$$P = 20/(20 + 40) = 1/3$$

$$R = 20/(20 + 60) = 1/4$$

$$F_1 = 2/7$$

# Thank You

# Information Retrieval

## Topic- Evaluation in information Retrieval (ranked evaluation)

### Lecture-28

**Prepared By**

Dr. Rasmita Rautray & Dr. Rasmita Dash  
Associate Professor  
Dept. of CSE

# Evaluation in information Retrieval

# Content

- Ranked retrieval
- ROC curve

# Precision and recall

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

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

# Precision and recall

	Relevant	Nonrelevant
Retrieved	true positives (TP)	false positives (FP)
Not retrieved	false negatives (FN)	true negatives (TN)

$$P = TP / (TP + FP)$$

$$R = TP / (TP + FN)$$

# Precision/recall tradeoff

- Recall can be increased by returning more docs.
- Recall is a non-decreasing function of the number of docs retrieved.
- A system that returns all docs has 100% recall!
- The converse is also true (usually): It's easy to get high precision for very low recall.
- Suppose the document with the largest score is relevant. How can we maximize precision?

# Example for precision, recall, F1

	relevant	not relevant	
retrieved	20	40	60
not retrieved	60	1,000,000	1,000,06
	80	1,000,040	1,000,12

$$P = 20/(20 + 40) = 1/3$$

$$R = 20/(20 + 60) = 1/4$$

$$F_1 = 2/7$$

# Accuracy

- Why do we use complex measures like precision, recall, and  $F$ ?
- Why not something simple like accuracy?
- Accuracy is the fraction of decisions (relevant/nonrelevant) that are correct.
- In terms of the contingency table above,  
**accuracy =  $(TP + TN)/(TP + FP + FN + TN)$ .**

# Why accuracy is a useless measure in IR

- Simple trick to maximize accuracy in IR: always say no and return nothing
- We then get 99.99% accuracy on most queries.
- Searchers on the web (and in IR in general) **want to find something** and have a certain tolerance for junk.
- It's better to return some bad hits as long as we return something.
  - We use precision, recall, and  $F$  for evaluation, not accuracy.

# F: Why harmonic mean?

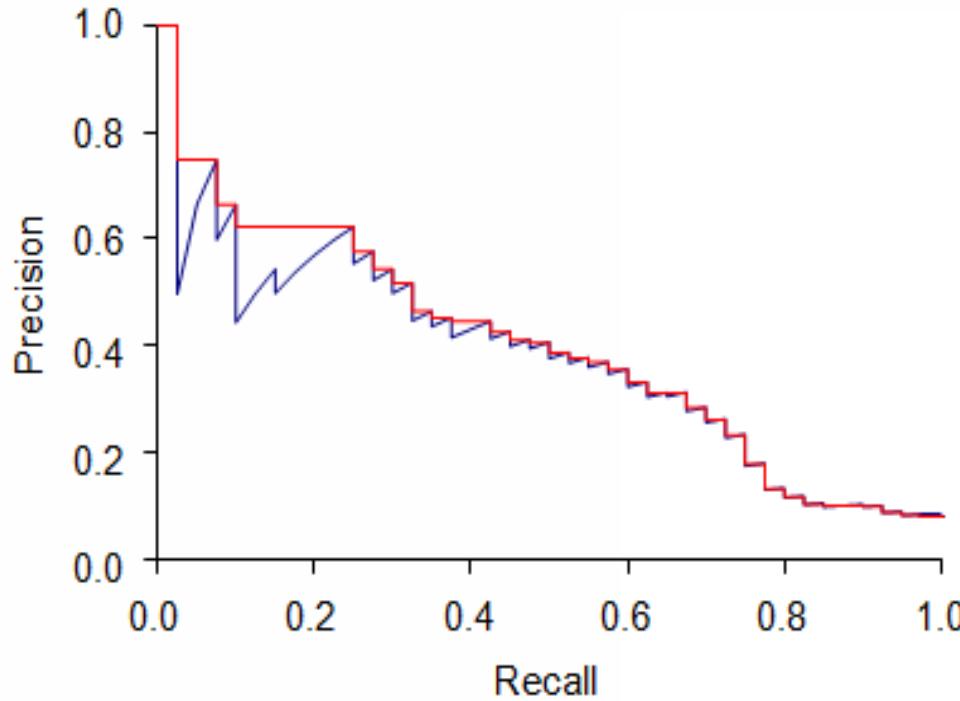
- Why don't we use a different mean of  $P$  and  $R$  as a measure?
  - e.g., the arithmetic mean
- Desideratum: Punish really bad performance on either precision or recall.
- Taking the minimum achieves this.
- But minimum is not smooth and hard to weight.
- $F$  (harmonic mean) is a kind of smooth minimum.

# Ranked evaluation

# Precision-recall curve

- Precision/recall/F are measures for **unranked sets**.
- We can easily turn set measures into measures of **ranked lists**.
- Just compute the set measure for each “prefix”: the top 1, top 2, top 3, top 4 etc results
- Doing this for precision and recall gives you a **precision-recall curve**.

# A precision-recall curve

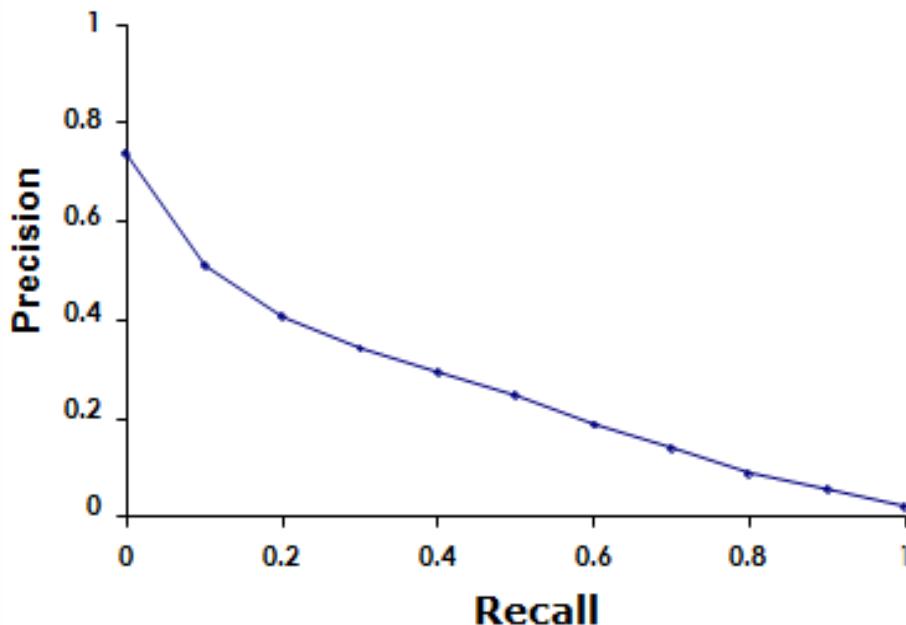


- Each point corresponds to a result for the top  $k$  ranked hits ( $k = 1, 2, 3, 4, \dots$ ).
  - **Interpolation (in red): Take maximum of all future points**
  - Rationale for interpolation: The user is willing to look at more stuff if both precision and recall get better.

# 11-point interpolated average precision

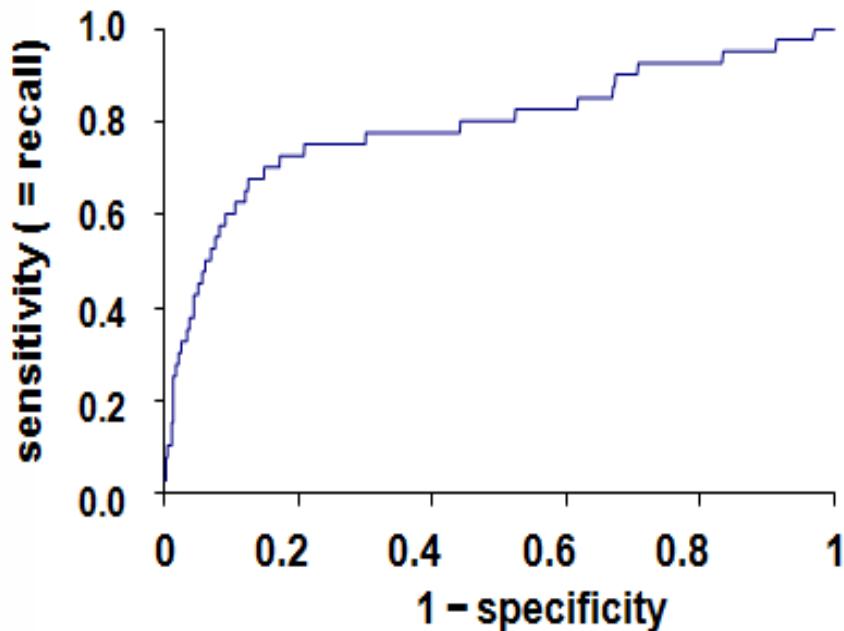
Recall	Interpolated Precision	
0.0	1.00	
0.1	0.67	• 11-point average: $\approx 0.425$
0.2	0.63	
0.3	0.55	
0.4	0.45	• How can precision at 0.0 be > 0?
0.5	0.41	
0.6	0.36	
0.7	0.29	
0.8	0.13	
0.9	0.10	
1.0	0.08	

# Averaged 11-point precision/recall graph



- Compute interpolated precision at recall levels 0.0, 0.1, 0.2,
  - ...
  - Do this for each of the queries in the evaluation benchmark
- Average over queries
- This measure measures performance **at all recall levels**.
  - The curve is typical of performance levels at TREC.
  - Note that performance is not very good!

# ROC curve



- Similar to precision-recall graph
- But we are only interested in the small area in the lower left corner.
- Precision-recall graph “blows up” this area.

# Variance of measures like precision/recall

- For a test collection, it is usual that a system does badly on some information needs (e.g.,  $P = 0.2$  at  $R = 0.1$ ) and really well on others (e.g.,  $P = 0.95$  at  $R = 0.1$ ).
- Indeed, it is usually the case that the variance of the same system across queries is much greater than the variance of different systems on the same query.
- That is, there are easy information needs and hard ones.

# Thank You

# Information Retrieval

## Topic- Relevance feedback and query expansion

### (Relevance feedback )

## Lecture-29

**Prepared By**

Dr. Rasmita Rautray & Dr. Rasmita Dash  
Associate Professor  
Dept. of CSE

# Relevance feedback

# Content

- Relevance feedback
- Rocchio algorithm

# Relevance

- We will evaluate the quality of an information retrieval system and, in particular, its ranking algorithm with respect to relevance.
- A document is relevant if it gives the user the information she was looking for.
- To evaluate relevance, we need an evaluation benchmark with three elements:
  - A benchmark document collection
  - A benchmark suite of queries
  - An assessment of the relevance of each query-document pair

# Relevance: query vs. information need

- The notion of “relevance to the query” is very problematic.
- ***Information need i***: “I am looking for information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine.”
- This is an information need, not a query.
- ***Query q***: [RED AND WINE AND WHITE AND WINE AND HEART AND ATTACK]
- Consider document d': He then launched into the heart of his speech and attacked the wine industry lobby for downplaying the role of red and white wine in drunk driving.
- d' is relevant to the query q, but d' is not relevant to the information need i .
- User happiness/satisfaction (i.e., how well our ranking algorithm works) can only be measured by relevance to information needs, not by relevance to queries

# Precision and recall

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

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

# A combined measure: F

- F allows us to trade off precision against recall.
- Balanced F:  
$$F = 2PR / (P + R)$$
- This is a kind of soft minimum of precision and recall.

- **Interactive relevance feedback:** improve initial retrieval results by telling the IR system which docs are relevant / non-relevant.
  - Best known relevance feedback method: Rocchio feedback
- **Query expansion:** improve retrieval results by adding synonyms / related terms to the query
  - Sources for related terms: Manual thesauri, automatic thesauri, query logs

# How can we improve recall in search?

- Main topic today: two ways of improving recall: relevance feedback and query expansion
- As an example consider query q: [aircraft] . . .
- . . . and document d containing “plane”, but not containing “aircraft”
- A simple IR system will not return d for q.
- Even if d is the most relevant document for q!
- We want to change this:
  - Return relevant documents even if there is no term match with the (original) query

# Recall

- Loose definition of recall in this lecture:  
“increasing the number of relevant documents returned to user”
- This may actually decrease recall on some measures, e.g., when expanding “jaguar” to “jaguar AND panthera”
  - . . . which eliminates some relevant documents, but increases relevant documents returned on top pages

# Options for improving recall

- Local: Do a “local”, on-demand analysis for a user query
  - Main local method: **relevance feedback**
- Global: Do a global analysis once (e.g., of collection) to produce thesaurus
  - Use thesaurus for **query expansion**

# Relevance feedback: Basic idea

- The user issues a (short, simple) query.
- The search engine returns a set of documents.
- User marks some docs as relevant, some as non-relevant.
- Search engine computes a new representation of the information need. Hope: better than the initial query.
- Search engine runs new query and returns new results.
- New results have (hopefully) better recall.
- We will use the term ad hoc retrieval to refer to regular retrieval without relevance feedback.

# Relevance feedback: Examples

New Page 1 - Netscape

File Edit View Go Bookmarks Tools Window Help

http://nayana.ece.ucsb.edu/ji

Home Browsing and ...

Shopping related 607,000 images are indexed and classified in the database  
Only One keyword is allowed!!!

bike

Search

Designed by [Baris Sumengen](#) and [Shawn Newsam](#)

Powered by JLAMP2000 (Java, Linux, Apache, Mysql, Perl, Windows2000)

# Results for initial query

						Browse	Search	Prev	Next	Random	
						(144473, 16458) 0.0 0.0 0.0	(144457, 252140) 0.0 0.0 0.0	(144456, 262857) 0.0 0.0 0.0	(144456, 262863) 0.0 0.0 0.0	(144457, 252134) 0.0 0.0 0.0	(144483, 265154) 0.0 0.0 0.0
						(144483, 264644) 0.0 0.0 0.0	(144483, 263153) 0.0 0.0 0.0	(144518, 257752) 0.0 0.0 0.0	(144538, 525937) 0.0 0.0 0.0	(144456, 249611) 0.0 0.0 0.0	(144456, 250064) 0.0 0.0 0.0

# User feedback: Select what is relevant

Browse   Search   Prev   Next   Random

(144473, 16459) 0.0 0.0 0.0	(144457, 252140) 0.0 0.0 0.0	(144456, 262857) 0.0 0.0 0.0	(144456, 262863) 0.0 0.0 0.0	(144457, 252134) 0.0 0.0 0.0	(144493, 265154) 0.0 0.0 0.0
(144493, 264644) 0.0 0.0 0.0	(144483, 265153) 0.0 0.0 0.0	(144518, 257752) 0.0 0.0 0.0	(144539, 525937) 0.0 0.0 0.0	(144456, 249611) 0.0 0.0 0.0	(144456, 250064) 0.0 0.0 0.0

# Results after relevance feedback

						Browse	Search	Prev	Next	Random	
						(144538, 523493) 0.34182 0.231944 0.309876	(144538, 523835) 0.56319296 0.267304 0.295889	(144538, 523529) 0.384279 0.280881 0.303398	(144456, 253569) 0.64501 0.351395 0.293615	(144456, 253568) 0.650275 0.411745 0.23853	(144538, 523799) 0.66709197 0.358033 0.309059
						(144473, 16249) 0.6721 0.393922 0.278178	(144456, 249634) 0.675018 0.4639 0.211118	(144456, 253693) 0.676901 0.47645 0.200451	(144473, 16328) 0.700339 0.309002 0.391337	(144483, 265264) 0.70170296 0.36176 0.339948	(144478, 512410) 0.70297 0.469111 0.233859

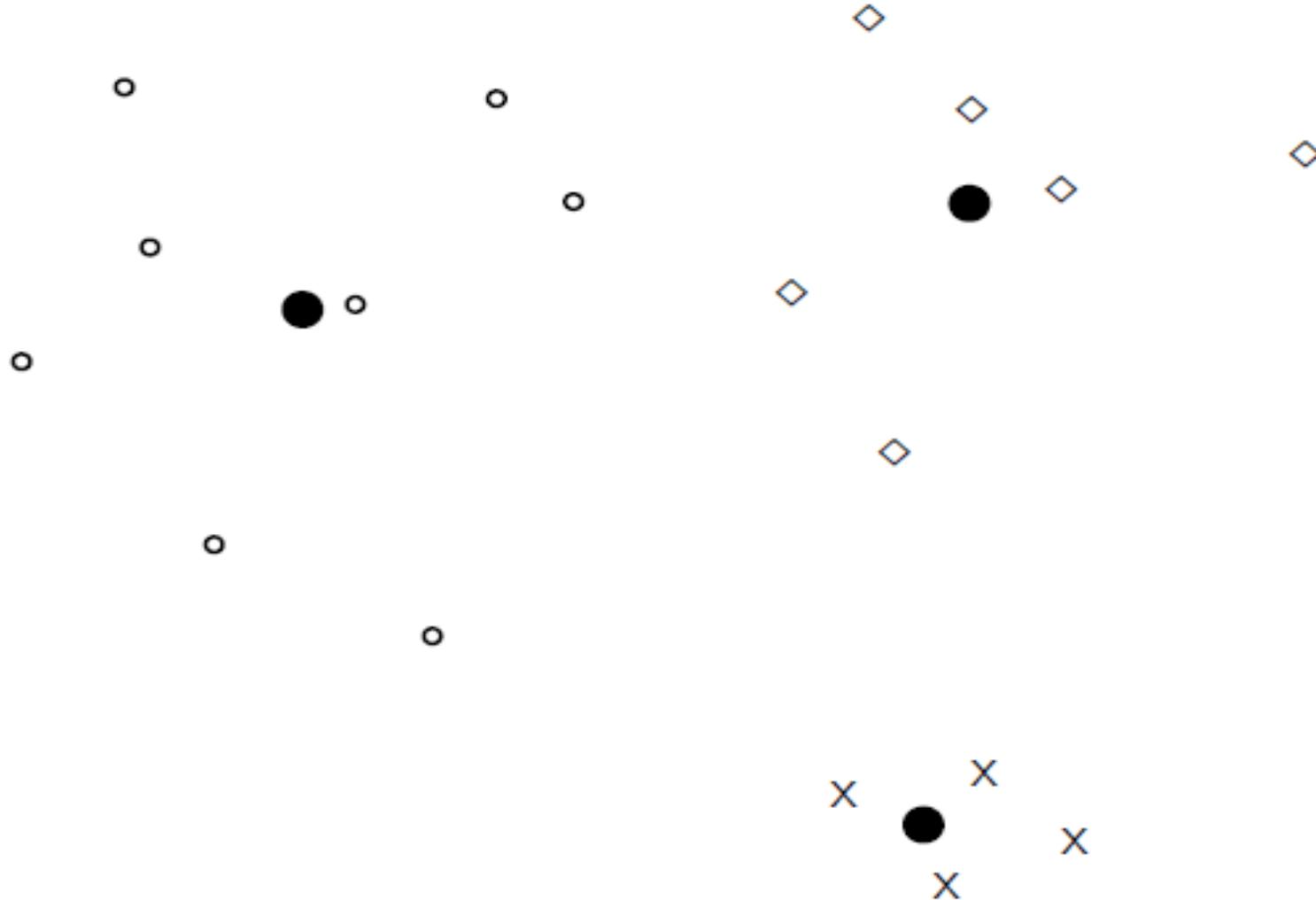
# Key concept for relevance feedback: Centroid

- The centroid is the center of mass of a set of points.
- Recall that we represent documents as points in a high-dimensional space.
- Thus: we can compute centroids of documents.
- Definition:

$$\text{Centroid} = \frac{1}{|D|} \sum_{d \in D} \vec{v}_d$$

- where  $D$  is a set of documents and  $\vec{v}_d$  is the vector we use to represent document  $d$ .

# Centroid: Examples



# Rocchio algorithm

- The Rocchio algorithm implements relevance feedback in the vector space model.
- Rocchio chooses the query that maximizes



- $D_r$  : set of relevant docs;  $D_{nr}$  : set of nonrelevant docs  $\vec{q}_{opt}$
- Intent:  $\vec{q}_{opt}$  is the vector that separates relevant and non-relevant docs maximally.
- Making some additional assumptions, we can rewrite  $\vec{q}_{opt}$  as:

# Rocchio algorithm

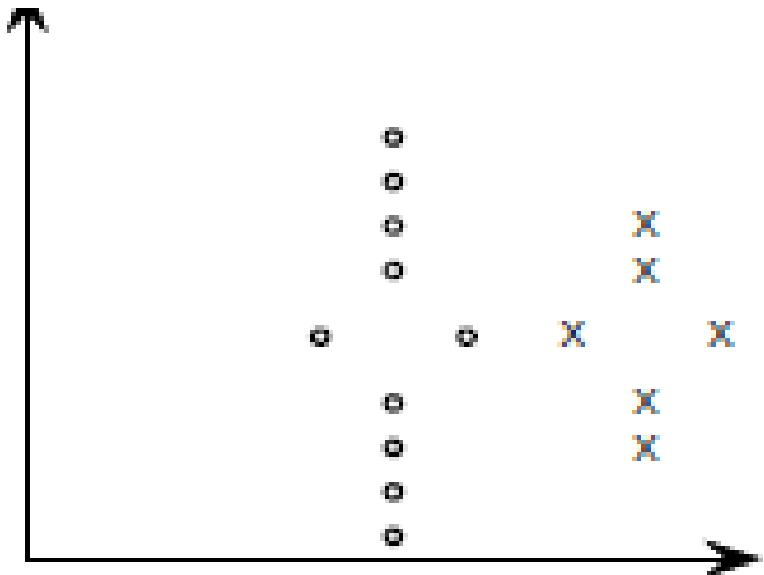
- The optimal query vector is:

$$\vec{q}_{opt} = \vec{q}_0 + \alpha_1 \vec{q}_r + \alpha_2 \vec{q}_n$$

The diagram illustrates the Rocchio formula for generating an optimal query vector. It shows a query vector  $\vec{q}$  as a weighted sum of three vectors:  $\vec{q}_0$ ,  $\vec{q}_r$ , and  $\vec{q}_n$ . The vector  $\vec{q}_0$  has a magnitude of 1. The vectors  $\vec{q}_r$  and  $\vec{q}_n$  also have magnitudes of 1. The tips of all three vectors meet at a single point on the right side of the diagram.

- We move the centroid of the relevant documents by the difference between the two centroids.

# Exercise: Compute Rocchio vector



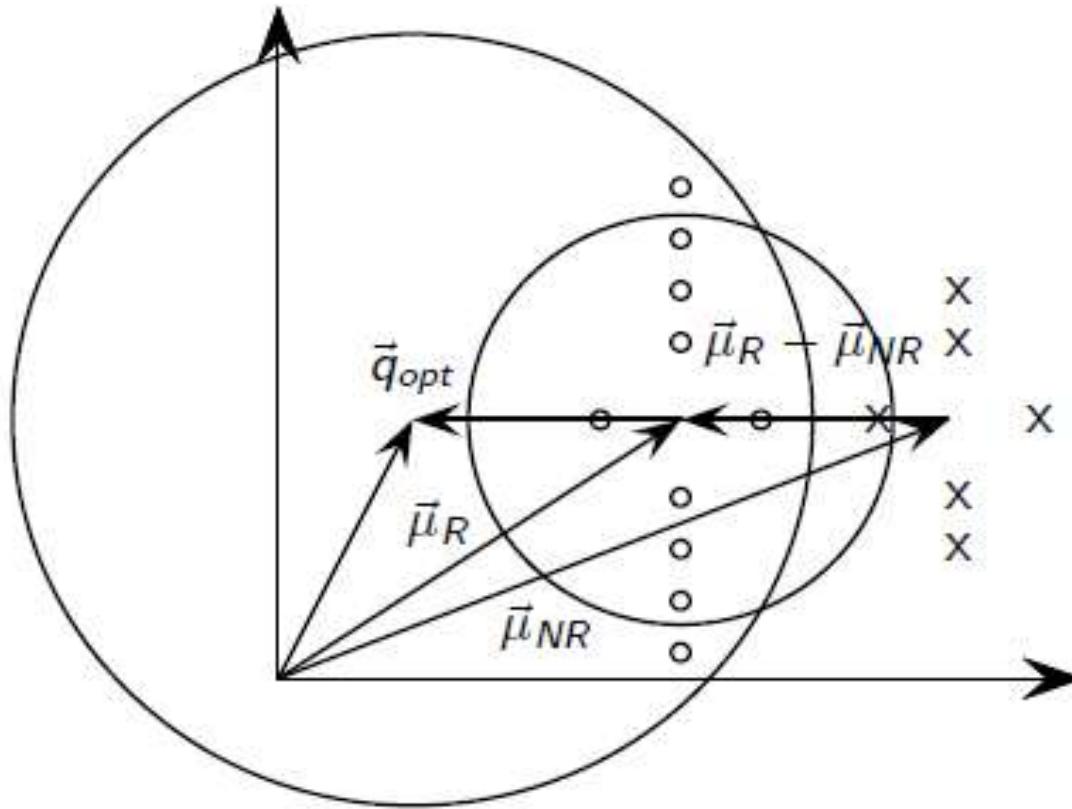
circles: relevant documents,

Xs: nonrelevant documents

compute:

$$q_{\text{opt}} = \mu(D_r) + [\mu(D_r) - \mu(D_{nr})]$$

# Rocchio illustrated



circles: relevant documents, Xs: nonrelevant documents  $\vec{\mu}_R$ : centroid of relevant documents  $\vec{\mu}_R$  does not separate relevant/nonrelevant.  $\vec{\mu}_{NR}$ : centroid of nonrelevant documents  $\vec{\mu}_R - \vec{\mu}_{NR}$ : difference vector Add difference vector to  $\vec{\mu}_R$  . . . . to get  $\vec{q}_{opt}$   $\vec{q}_{opt}$  separates relevant/nonrelevant perfectly.

# Thank You

# Information Retrieval

## Topic- Relevance feedback and query expansion

### (Relevance feedback contd...)

## Lecture-30

**Prepared By**

Dr. Rasmita Rautray & Dr. Rasmita Dash  
Associate Professor  
Dept. of CSE

# Relevance feedback

# Content

- Relevance feedback
- Rocchio algorithm
- Relevance feedback: Assumptions
- Relevance feedback: Evaluation

# Relevance

- A document is relevant if it gives the user the information she was looking for.
- To evaluate relevance, we need an evaluation benchmark with three elements:
  - A benchmark document collection
  - A benchmark suite of queries
  - An assessment of the relevance of each query-document pair

- **Interactive relevance feedback:** improve initial retrieval results by telling the IR system which docs are relevant / non-relevant.
  - Best known relevance feedback method: Rocchio feedback
- **Query expansion:** improve retrieval results by adding synonyms / related terms to the query
  - Sources for related terms: Manual thesauri, automatic thesauri, query logs

# Options for improving recall

- Local: Do a “local”, on-demand analysis for a user query
  - Main local method: **relevance feedback**
- Global: Do a global analysis once (e.g., of collection) to produce thesaurus
  - Use thesaurus for **query expansion**

# Relevance feedback: Basic idea

- The user issues a (short, simple) query.
- The search engine returns a set of documents.
- User marks some docs as relevant, some as non-relevant.
- Search engine computes a new representation of the information need. Hope: better than the initial query.
- Search engine runs new query and returns new results.
- New results have (hopefully) better recall.
- We will use the term ad hoc retrieval to refer to regular retrieval without relevance feedback.

# Relevance feedback: Examples

New Page 1 - Netscape

File Edit View Go Bookmarks Tools Window Help

http://nayana.ece.ucsb.edu/ji

Home Browsing and ...

Shopping related 607,000 images are indexed and classified in the database  
Only One keyword is allowed!!!

bike

Search

Designed by [Baris Sumengen](#) and [Shawn Newsam](#)

Powered by JLAMP2000 (Java, Linux, Apache, Mysql, Perl, Windows2000)

# Results for initial query

Browse   Search   Prev   Next   Random

(144473, 16458) (144457, 252140) (144456, 262857) (144456, 262863) (144457, 252134) (144483, 265154)  
0.0  
0.0  
0.0  
0.0  
0.0  
0.0

(144483, 264644) (144483, 263153) (144518, 257752) (144538, 525937) (144456, 249611) (144456, 250064)  
0.0  
0.0  
0.0  
0.0  
0.0  
0.0

# User feedback: Select what is relevant

Browse   Search   Prev   Next   Random

(144473, 16459) 0.0 0.0 0.0	(144457, 252140) 0.0 0.0 0.0	(144456, 262857) 0.0 0.0 0.0	(144456, 262863) 0.0 0.0 0.0	(144457, 252134) 0.0 0.0 0.0	(144493, 265154) 0.0 0.0 0.0
(144493, 264644) 0.0 0.0 0.0	(144483, 265153) 0.0 0.0 0.0	(144518, 257752) 0.0 0.0 0.0	(144539, 525937) 0.0 0.0 0.0	(144456, 249611) 0.0 0.0 0.0	(144456, 250064) 0.0 0.0 0.0

# Results after relevance feedback

						Browse	Search	Prev	Next	Random	
						(144538, 523493) 0.34182 0.231944 0.309876	(144538, 523835) 0.56319296 0.267304 0.295889	(144538, 523529) 0.384279 0.280881 0.303398	(144456, 253569) 0.64501 0.351395 0.293615	(144456, 253568) 0.650275 0.411745 0.23853	(144538, 523799) 0.66709197 0.358033 0.309059
						(144473, 16249) 0.6721 0.393922 0.278178	(144456, 249634) 0.675018 0.4639 0.211118	(144456, 253693) 0.676901 0.47645 0.200451	(144473, 16328) 0.700339 0.309002 0.391337	(144483, 265264) 0.70170296 0.36176 0.339948	(144478, 512410) 0.70297 0.469111 0.233859

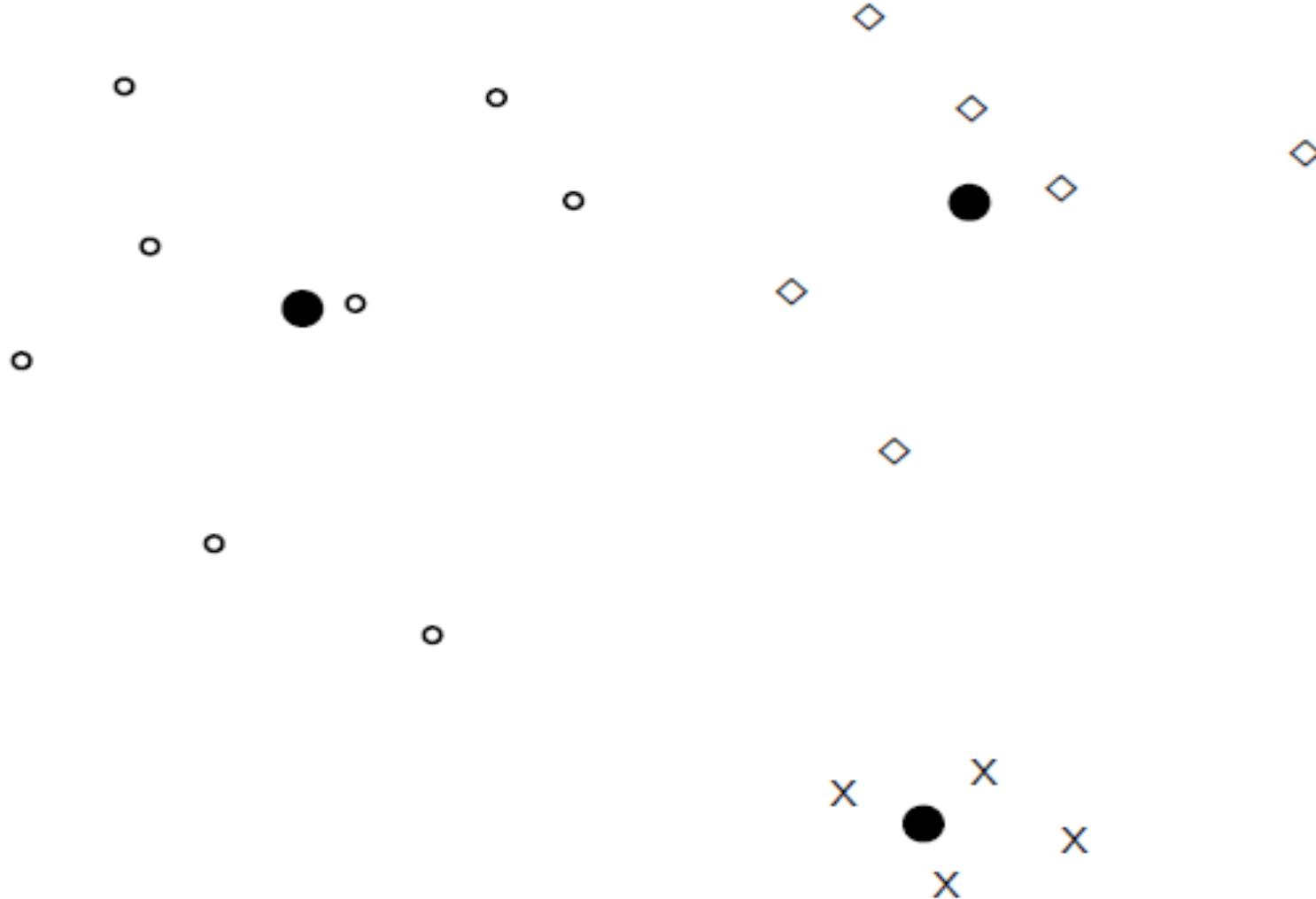
# Key concept for relevance feedback: Centroid

- The centroid is the center of mass of a set of points.
- Recall that we represent documents as points in a high-dimensional space.
- Thus: we can compute centroids of documents.
- Definition:

$$\text{Centroid} = \frac{1}{|D|} \sum_{d \in D} \vec{v}_d$$

- where  $D$  is a set of documents and  $\vec{v}_d$  is the vector we use to represent document  $d$ .

# Centroid: Examples



# Rocchio algorithm

- The Rocchio algorithm implements relevance feedback in the vector space model.
- Rocchio chooses the query that maximizes



- $D_r$ : set of relevant docs;  $D_{nr}$ : set of nonrelevant docs  $\vec{q}_{opt}$
- Intent:  $\vec{q}_{opt}$  is the vector that separates relevant and non-relevant docs maximally.
- Making some additional assumptions, we can rewrite as:

# Rocchio algorithm

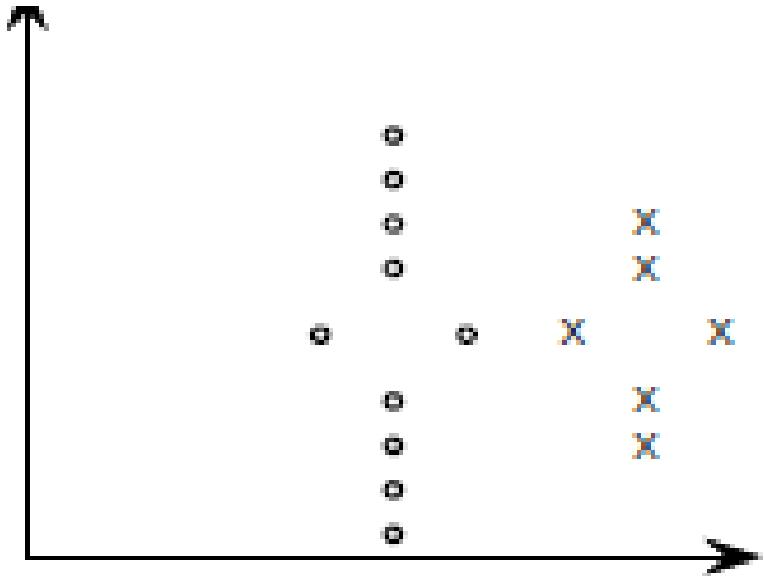
- The optimal query vector is:

$$\vec{q}_P = \vec{D}_{Relevant} - \vec{D}_{Irrelevant}$$

$$= \frac{1}{n_r} \sum_{i=1}^{n_r} \vec{D}_{Relevant} - \frac{1}{n_i} \sum_{j=1}^{n_i} \vec{D}_{Irrelevant}$$

- We move the centroid of the relevant documents by the difference between the two centroids.

# Exercise: Compute Rocchio vector



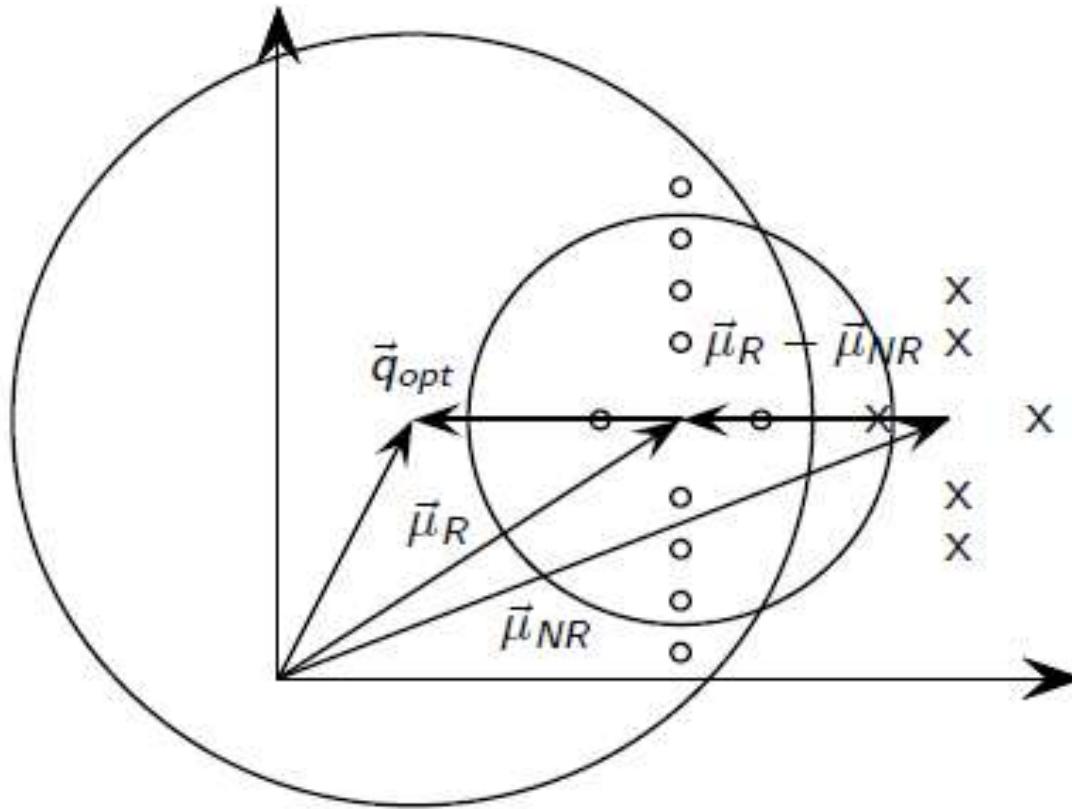
circles: relevant documents,

Xs: nonrelevant documents

compute:

$$q_{\text{opt}} = \mu(D_r) + [\mu(D_r) - \mu(D_{nr})]$$

# Rocchio illustrated



circles: relevant documents, Xs: nonrelevant documents  $\vec{\mu}_R$ : centroid of relevant documents  $\vec{\mu}_R$  does not separate relevant/nonrelevant.  $\vec{\mu}_{NR}$ : centroid of nonrelevant documents  $\vec{\mu}_R - \vec{\mu}_{NR}$ : difference vector Add difference vector to  $\vec{\mu}_R$  . . . . to get  $\vec{q}_{opt}$   $\vec{q}_{opt}$  separates relevant/nonrelevant perfectly.

# Positive vs. negative relevance feedback

- Positive feedback is more valuable than negative feedback.
- For example, set  $\beta = 0.75$ ,  $\gamma = 0.25$  to give higher weight to positive feedback.
- Many systems only allow positive feedback.

# Relevance feedback: Assumptions

- When can relevance feedback enhance recall?
- Assumption A1: The user knows the terms in the collection well enough for an initial query.
- Assumption A2: Relevant documents contain similar terms

# Violation of A1

- Assumption A1: The user knows the terms in the collection well enough for an initial query.
- Violation: Mismatch of searcher's vocabulary and collection vocabulary
- Example: cosmonaut / astronaut

# Violation of A2

- Assumption A2: Relevant documents are similar.
- Example for violation: [contradictory government policies]
- Several unrelated “prototypes”
  - Subsidies for tobacco farmers vs. anti-smoking campaigns
  - Aid for developing countries vs. high tariffs on imports from developing countries
- Relevance feedback on tobacco docs will not help with finding docs on developing countries.

# Relevance feedback: Assumptions

- When can relevance feedback enhance recall?
- Assumption A1: The user knows the terms in the collection well enough for an initial query.
- Assumption A2: Relevant documents contain similar terms (so I can “hop” from one relevant document to a different one when giving relevance feedback).

# Relevance feedback: Evaluation

- Pick an evaluation measure, e.g., precision in top 10: P@10
- Compute P@10 for original query  $q_0$
- Compute P@10 for modified relevance feedback query  $q_1$
- In most cases:  $q_1$  is spectacularly better than  $q_0$ !
- Is this a fair evaluation?

# Relevance feedback: Evaluation

- Fair evaluation must be on “residual” collection: docs not yet judged by user.
- Studies have shown that relevance feedback is successful when evaluated this way.
- Empirically, one round of relevance feedback is often very useful. Two rounds are marginally useful.

# Evaluation: Caveat

- True evaluation of usefulness must compare to other methods taking the same amount of time.
- Alternative to relevance feedback: User revises and resubmits query.
- Users may prefer revision/resubmission to having to judge relevance of documents.
- There is no clear evidence that relevance feedback is the “best use” of the user’s time.

# 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.
- The search engine Excite had full relevance feedback at one point, but abandoned it later.

# Pseudo-relevance feedback

- Pseudo-relevance feedback automates the “manual” part of true relevance feedback.
- Pseudo-relevance feedback algorithm:
  - Retrieve a ranked list of hits for the user’s query
  - Assume that the top  $k$  documents are relevant.
  - Do relevance feedback (e.g., Rocchio)
- Works very well on average
- But can go horribly wrong for some queries.
- Because of query drift
- If you do several iterations of pseudo-relevance feedback, then you will get query drift for a large proportion of queries.

# Thank You

# Information Retrieval

## Topic- Relevance feedback and query expansion (Query expansion)

### Lecture-31

**Prepared By**

Dr. Rasmita Rautray & Dr. Rasmita Dash  
Associate Professor  
Dept. of CSE

# Query expansion

# Content

- Relevance feedback: Assumption
- Relevance feedback: Evaluation
- Query expansion

# Relevance

- A document is relevant if it gives the user the information she was looking for.

- **Interactive relevance feedback:** improve initial retrieval results by telling the IR system which docs are relevant / non-relevant.
  - Best known relevance feedback method: Rocchio feedback
- **Query expansion:** improve retrieval results by adding synonyms / related terms to the query
  - Sources for related terms: Manual thesauri, automatic thesauri, query logs

# Relevance feedback: Assumptions

- When can relevance feedback enhance recall?
- Assumption A1: The user knows the terms in the collection well enough for an initial query.
- Assumption A2: Relevant documents contain similar terms

# Violation of A1

- Assumption A1: The user knows the terms in the collection well enough for an initial query.
- Violation: Mismatch of searcher's vocabulary and collection vocabulary
- Example: cosmonaut / astronaut

# Violation of A2

- Assumption A2: Relevant documents are similar.
- Example for violation: [contradictory government policies]
- Several unrelated “prototypes”
  - Subsidies for tobacco farmers vs. anti-smoking campaigns
  - Aid for developing countries vs. high tariffs on imports from developing countries
- Relevance feedback on tobacco docs will not help with finding docs on developing countries.

# Relevance feedback: Assumptions

- When can relevance feedback enhance recall?
- Assumption A1: The user knows the terms in the collection well enough for an initial query.
- Assumption A2: Relevant documents contain similar terms (so I can “hop” from one relevant document to a different one when giving relevance feedback).

# Relevance feedback: Evaluation

- Pick an evaluation measure, e.g., precision in top 10: P@10
- Compute P@10 for original query  $q_0$
- Compute P@10 for modified relevance feedback query  $q_1$
- In most cases:  $q_1$  is spectacularly better than  $q_0$ !
- Is this a fair evaluation?

# 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.
- The search engine Excite had full relevance feedback at one point, but abandoned it later.

# Pseudo-relevance feedback

- Pseudo-relevance feedback automates the “manual” part of true relevance feedback.
- Pseudo-relevance feedback algorithm:
  - Retrieve a ranked list of hits for the user’s query
  - Assume that the top  $k$  documents are relevant.
  - Do relevance feedback (e.g., Rocchio)
- Works very well on average
- But can go horribly wrong for some queries.
- Because of query drift
- If you do several iterations of pseudo-relevance feedback, then you will get query drift for a large proportion of queries.

# Types of user feedback

- User gives feedback on documents.
  - More common in relevance feedback
- User gives feedback on words or phrases.
  - More common in query expansion

# Query expansion

- Query expansion is another method for increasing recall.
- We use “global query expansion” to refer to “global methods for query reformulation”.
- In global query expansion, the query is modified based on some global resource, i.e. a resource that is not query-dependent.
- Main information we use: (near-)synonymy

# “Global” resources used for query expansion

- A publication or database that collects (near-)synonyms is called a thesaurus.
- Manual thesaurus (maintained by editors, e.g., PubMed)
- Automatically derived thesaurus (e.g., based on co-occurrence statistics)
- Query-equivalence based on query log mining (common on the web as in the “palm” example)

# Thesaurus-based query expansion

- For each term  $t$  in the query, expand the query with words the thesaurus lists as semantically related with  $t$ .
- Example from earlier: HOSPITAL → MEDICAL
- Generally increases recall
- May significantly decrease precision, particularly with ambiguous terms
- INTEREST RATE → INTEREST RATE  
FASCINATE
- Widely used in specialized search engines for science and engineering
- It's very expensive to create a manual thesaurus

# Example for manual thesaurus: PubMed

The screenshot shows the PubMed search interface. The top navigation bar includes links for PubMed, Nucleotide, Protein, Genome, Structure, PopSet, and Taxonomy. Below the navigation bar is a search bar containing the text "Search PubMed for cancer". To the right of the search bar are "Go" and "Clear" buttons. Below the search bar are buttons for "Limits", "Preview/Index", "History", "Clipboard", and "Details". On the left side of the page is a sidebar with links for "About Entrez", "Text Version", "Entrez PubMed Overview", "Help | FAQ", "Tutorial", "New/Noteworthy", "E-Utilities", "PubMed Services", "Journals Database", "MeSH Browser", and "Single Citation". At the bottom of the page are "Search" and "URL" buttons.

PubMed Query:

```
("neoplasms"[MeSH Terms] OR cancer[Text Word])
```

Search URL

# Automatic thesaurus generation

- Attempt to generate a thesaurus automatically by analyzing the distribution of words in documents
- Fundamental notion: similarity between two words
- Definition 1: Two words are similar if they co-occur with similar words.
  - “car” ≈ “motorcycle” because both occur with “road”, “gas” and “license”, so they must be similar.
- Definition 2: Two words are similar if they occur in a given grammatical relation with the same words.
  - You can harvest, peel, eat, prepare, etc. apples and oranges, so they are similar.

# Query expansion at search engines

- Main source of query expansion at search engines: query logs
- Example 1: After issuing the query [herbs], users frequently search for [herbal remedies].
  - → “herbal remedies” is potential expansion of “herb”.
- Example 2: Users searching for [flower pix] frequently click on the URL photobucket.com/flower. Users searching for [flower clipart] frequently click on the same URL.
  - → “flower clipart” and “flower pix” are potential expansions of each other.

# Take-away today Interactive

- Interactive relevance feedback: improve initial retrieval results by telling the IR system which docs are relevant / non-relevant
- Best known relevance feedback method: Rocchio feedback
- Query expansion: improve retrieval results by adding synonyms / related terms to the query
  - Sources for related terms: Manual thesauri, automatic thesauri, query logs

# Thank You

# Information Retrieval

## Topic- XML Retrieval

## Lecture-32

**Prepared By**

Dr. Rasmita Rautray & Dr. Rasmita Dash  
Associate Professor  
Dept. of CSE

# XML Retrieval

# Content

- XML Retrieval
- XML Retrieval : Challenges

# IR and relational databases

- IR systems are often contrasted with relational databases (RDB).
- Traditionally, IR systems retrieve information from *unstructured text* (“raw” text without markup).
- RDB systems are used for querying *relational data*: sets of records that have values for predefined attributes such as employee number, title and salary.

	RDB search	unstructured
<b>IR</b>		
<b>objects</b>	records	unstructured docs
<b>main data structure</b>	table	inverted index
<b>model</b>	relational model	vector space &
<b>others</b>		
<b>queries</b>	SQL	free text queries

- Some structured data sources containing text are best

# **Structured retrieval**

- Basic setting: queries are structured or unstructured; documents are structured.

## **Applications of structured retrieval**

- Digital libraries, patent databases, blogs, tagged text with entities like persons and locations (named entity tagging).

# Why RDB is not suitable in this case

- Three main **problems** :
- An unranked system (DB) would return a potentially large number of articles that mention the Vatican, the Coliseum and sightseeing tours without ranking them by relevance to the query.
- Difficult for users to precisely state structural constraints - may not know which structured elements are supported by the system.

*tours AND(COUNTRY : Vatican OR*

*LANDMARK : Coliseum) ?*

*tours AND (STATE : Vatican OR BUILDING : Coliseum) ?*

- Users may be completely unfamiliar with structured search and advanced search interfaces or unwilling to use them.
- **Solution:** adapt ranked retrieval to structured documents to address these problems

# Structured Retrieval

	RDB search	unstructured retrieval	structured retrieval
objects	records	unstructured documents	trees with text at leaves
model	relational model	vector space & others	?
main data structure	table	inverted index	?
queries	SQL	free text queries	?

- Standard for encoding structured documents: Extensible Markup Language ( XML )
- structured IR → XML IR
- also applicable to other types of markup (HTML, SGML, ...)

# XML document

- Ordered, labeled tree

- Each node of the tree is an XML element, written with an opening and closing XML tag (e.g.

<title...>, </title...>)

- An element can have one or more XML attributes (e.g. number)

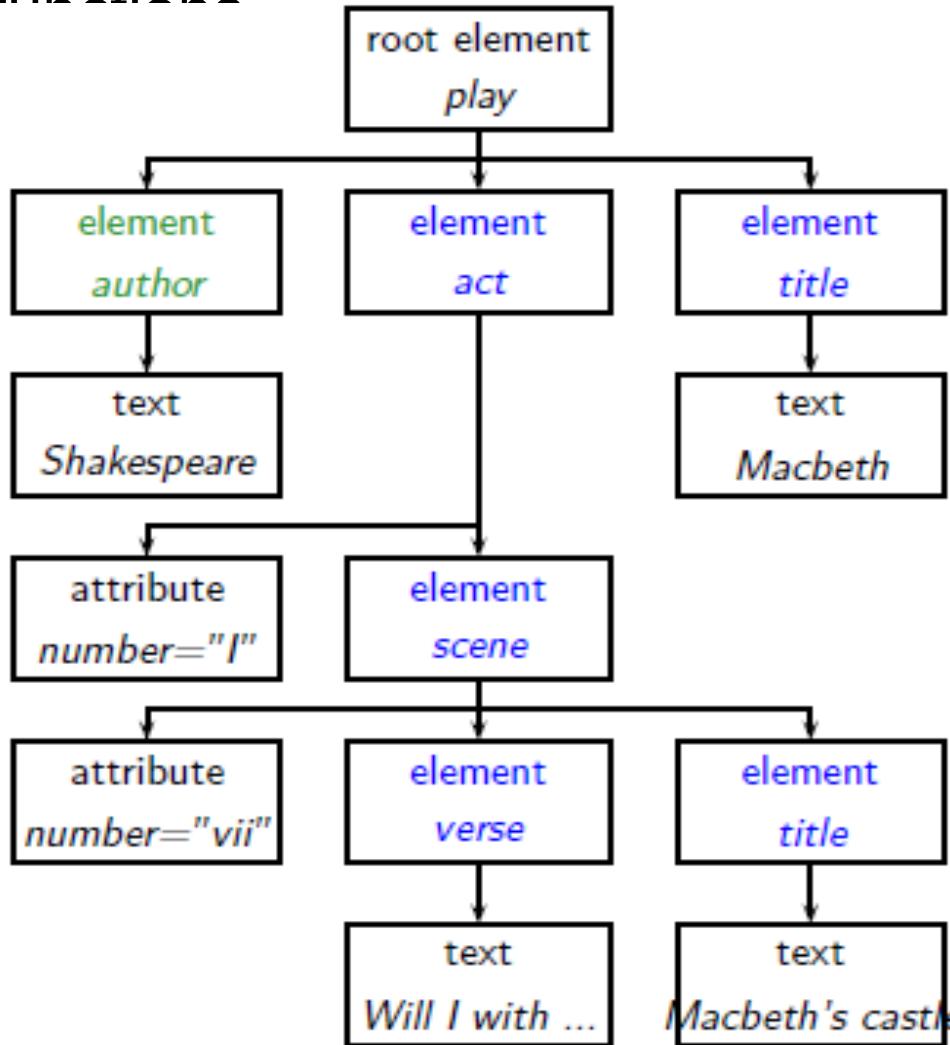
- Attributes can have values (e.g. vii)

- Attributes can have child elements (e.g.

```
<play>
<author>Shakespeare</author>
<title>Macbeth</title>
<act number=" I">
<scene number=" vii">
<title>Macbeth's castle</title>
<verse>Will I with wine
...
</verse>
</scene>
</act>
</play>
```

# XML document

- The *internal nodes* encode **document structure or metadata**



# XML basics

- **XML Document Object Model (XML DOM):** standard for accessing and processing XML documents
  - The DOM represents elements, attributes and text within elements as nodes in a tree.
  - With a DOM API, we can process an XML document by starting at the root element and then descending down the tree from parents to children.
- **XPath:** standard for enumerating paths in an XML document collection.
  - We will also refer to paths as XML contexts or simply contexts
- **Schema:** puts constraints on the structure of allowable XML documents. E.g. a schema for Shakespeare's plays: scenes can only occur as children of acts.
  - Two standards for schemas for XML documents are: XML DTD (document type definition) and XML Schema.

# Challenges in XML IR

# First challenge: document parts to retrieve

- Structured or XML retrieval: users want us to return parts of documents (i.e., XML elements), not entire documents as IR systems usually do in unstructured retrieval.

## Example

- If we query Shakespeare's plays for *Macbeth's castle*, should we return the scene, the act or the entire play?
  - In this case, the user is probably looking for the scene.
  - However, an otherwise unspecified search for *Macbeth* should return the play of this name, not a subunit.

**Solution:** structured document retrieval principle

# Structured document retrieval principle

Structured document retrieval principle

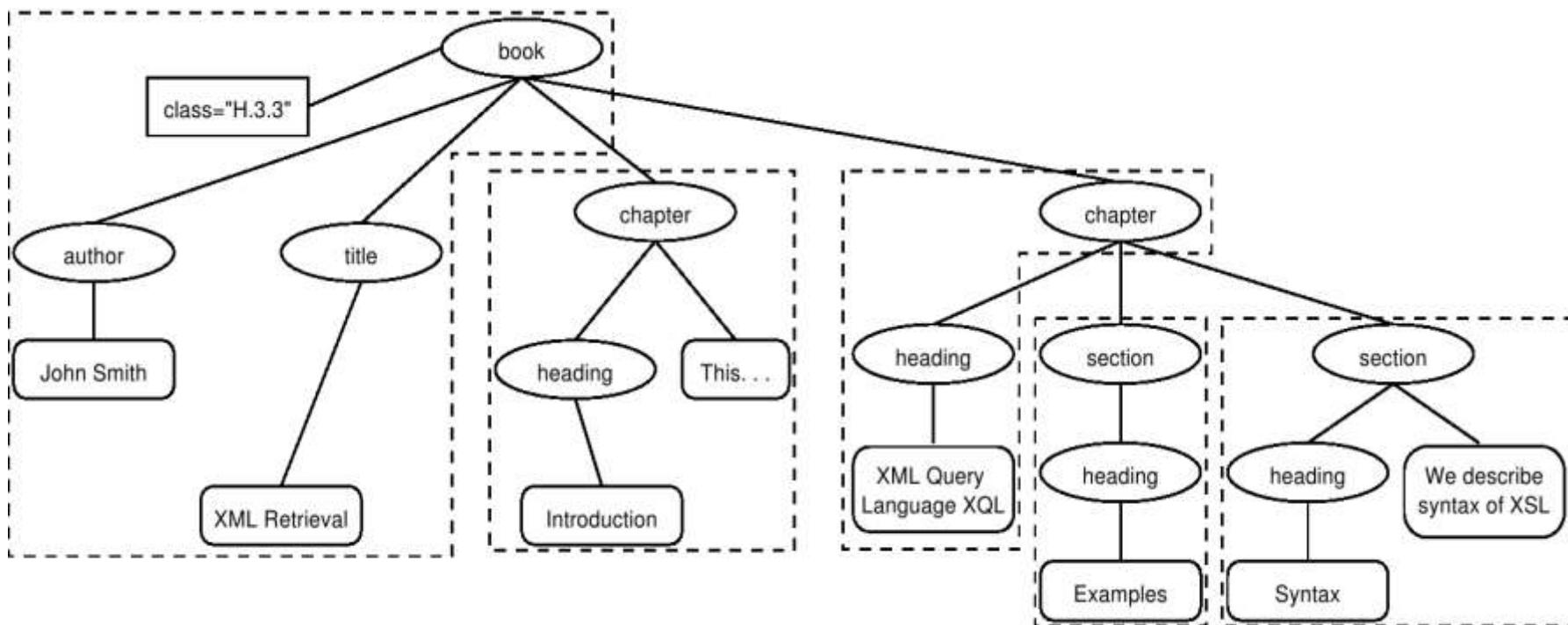
- One criterion for selecting the most appropriate part of a document:
  - A system *should always retrieve the most specific part of a document answering the query.*
- Motivates a retrieval strategy that returns the smallest unit that contains the information sought, but does not go below this level.
- **Hard to implement** this principle algorithmically. E.g. query: *title: Macbeth* can match both the title of the tragedy, *Macbeth*, and the title of Act I, Scene vii, *Macbeth's castle*.
  - But in this case, the title of the tragedy (higher node) is preferred.
  - Difficult to decide which level of the tree satisfies the query.

# Second challenge: document parts to index

- Central notion for indexing and ranking in IR: **document unit** or **indexing unit**.
- In unstructured retrieval, usually straightforward: files on your desktop, email messages, web pages on the web etc.
- In structured retrieval, there are four main different approaches to defining the indexing unit.
  - non-overlapping pseudodocuments
  - top down
  - bottom up
  - all

# XML indexing unit: approach 1

- Group nodes into non-overlapping pseudo-documents.



**Indexing units:** books, chapters, sections, but without overlap.

**Disadvantage:** pseudo-documents may not make sense to the user because they are not coherent units.

# XML indexing unit: approach 2

- Top down (2-stage process):
  - start with one of the largest elements as the indexing unit,  
e.g. the *book element in a collection of books*
  - then, post process search results to find for each book the sub-element that is the best hit.
- This two-stage retrieval process often fails to return the best sub-element because the relevance of a whole book is often not a good predictor of the relevance of small sub-elements within it.

# XML indexing unit: approach 3

Bottom up:

- Instead of retrieving large units and identifying sub-elements (top down), we can search all leaves, select the most relevant ones and then extend them to larger units in post processing.
- Similar problem as top down: the relevance of a leaf element is often not a good predictor of the relevance of elements it is contained in.

# XML indexing unit: approach 4

- Index all elements: the least restrictive approach. Also problematic:
  - many XML elements are not meaningful search results, e.g., an ISBN number.
  - indexing all elements means that search results will be highly redundant.

## Example

For the query Macbeth's castle we would return all of the play, act, scene and title elements on the path between the root node and Macbeth's castle. The leaf node would then occur 4 times in the result set: 1 directly and 3 as part of other elements.

- We call elements that are contained within each other nested elements. Returning redundant nested elements in a list of returned hits is not very user-friendly.

# Third challenge: nested elements

- Because of the redundancy caused by nested elements it is common to restrict the set of elements eligible for retrieval. Restriction strategies include:
  - discard all small elements
  - discard all element types that users do not look at (working XML retrieval system logs)
  - discard all element types that assessors generally do not judge to be relevant (if relevance assessments are available)
  - only keep element types that a system designer or librarian has deemed to be useful search results
- In most of these approaches, result sets will still contain nested elements.

# Third challenge: nested elements

Further techniques:

- remove nested elements in a post processing step to reduce redundancy.
- collapse several nested elements in the results list and use highlighting of query terms to draw the user's attention to the relevant passages.

## Highlighting

- Gain 1: enables users to scan medium-sized elements (e.g., a section); thus, if the section and the paragraph both occur in the results list, it is sufficient to show the section.
- Gain 2: paragraphs are presented in-context (i.e., their embedding section). This context may be helpful in interpreting the paragraph.

# Nested elements and term statistics

- Further challenge related to nesting: we may need to distinguish different contexts of a term when we compute term statistics for ranking, in particular inverse document frequency (idf).

## Example

- The term *Gates under the node author is unrelated to an occurrence under a content node like section if used to refer to the plural of gate. It makes little sense to compute a single document frequency for Gates in this example.*

**Solution:** compute idf for XML-context term pairs.

- sparse data problems (many XML-context pairs occur too rarely to reliably estimate df)

**compromise:** consider the parent node *x of the term and not the rest of the path from the root to x to distinguish contexts*

# Thank You

# Information Retrieval

## Topic- XML Retrieval

### (Vector space model for XML IR)

## Lecture-33

**Prepared By**

Dr. Rasmita Rautray & Dr. Rasmita Dash  
Associate Professor  
Dept. of CSE

# XML Retrieval

# Content

- XML Retrieval
- XML Retrieval : Vector space model
- XML Retrieval : Assessment

# IR and relational databases

- IR systems are often contrasted with relational databases (RDB).
- Traditionally, IR systems retrieve information from *unstructured text* (“raw” text without markup).
- RDB systems are used for querying *relational data*: sets of records that have values for predefined attributes such as employee number, title and salary.

	RDB search	unstructured
<b>IR objects</b>	records	unstructured docs
<b>main data structure</b>	table	inverted index
<b>model</b>	relational model	vector space &
<b>others</b>		
<b>queries</b>	SQL	free text queries

- Some structured data sources containing text are best

# Structured retrieval

- Basic setting: queries are structured or unstructured; documents are structured.

## Applications of structured retrieval

- Digital libraries, patent databases, blogs, tagged text with entities like persons and locations (named entity tagging).

# Why RDB is not suitable in this case

- Three main **problems** :
- An unranked system (DB) would return a potentially large number of articles that mention the Vatican, the Coliseum and sightseeing tours without ranking them by relevance to the query.
- Difficult for users to precisely state structural constraints - may not know which structured elements are supported by the system.

*tours AND(COUNTRY : Vatican OR*

*LANDMARK : Coliseum) ?*

*tours AND (STATE : Vatican OR BUILDING : Coliseum) ?*

- Users may be completely unfamiliar with structured search and advanced search interfaces or unwilling to use them.
- **Solution:** adapt ranked retrieval to structured documents to address these problems

# Structured Retrieval

	RDB search	unstructured retrieval	structured retrieval
objects	records	unstructured documents	trees with text at leaves
model	relational model	vector space & others	?
main data structure	table	inverted index	?
queries	SQL	free text queries	?

- Standard for encoding structured documents: Extensible Markup Language ( XML )
- structured IR → XML IR
- also applicable to other types of markup (HTML, SGML, ...)

# First challenge: document parts to retrieve

- Structured or XML retrieval: users want us to return parts of documents (i.e., XML elements), not entire documents as IR systems usually do in unstructured retrieval.

## Example

- If we query Shakespeare's plays for *Macbeth's castle*, should we return the scene, the act or the entire play?
  - In this case, the user is probably looking for the scene.
  - However, an otherwise unspecified search for *Macbeth* should return the play of this name, not a subunit.

**Solution:** structured document retrieval principle

# Second challenge: document parts to index

- Central notion for indexing and ranking in IR: **document unit** or **indexing unit**.
- In unstructured retrieval, usually straightforward: files on your desktop, email messages, web pages on the web etc.
- In structured retrieval, there are four main different approaches to defining the indexing unit.
  - non-overlapping pseudodocuments
  - top down
  - bottom up
  - all

# Third challenge: nested elements

- Because of the redundancy caused by nested elements it is common to restrict the set of elements eligible for retrieval. Restriction strategies include:
  - discard all small elements
  - discard all element types that users do not look at (working XML retrieval system logs)
  - discard all element types that assessors generally do not judge to be relevant (if relevance assessments are available)
  - only keep element types that a system designer or librarian has deemed to be useful search results
- In most of these approaches, result sets will still contain nested elements.

# Third challenge: nested elements

Further techniques:

- remove nested elements in a post processing step to reduce redundancy.
- collapse several nested elements in the results list and use highlighting of query terms to draw the user's attention to the relevant passages.

# Nested elements and term statistics

- Further challenge related to nesting: we may need to distinguish different contexts of a term when we compute term statistics for ranking, in particular inverse document frequency (idf).

## Example

- The term *Gates under the node author is unrelated to an occurrence under a content node like section if used to refer to the plural of gate. It makes little sense to compute a single document frequency for Gates in this example.*

**Solution:** compute idf for XML-context term pairs.

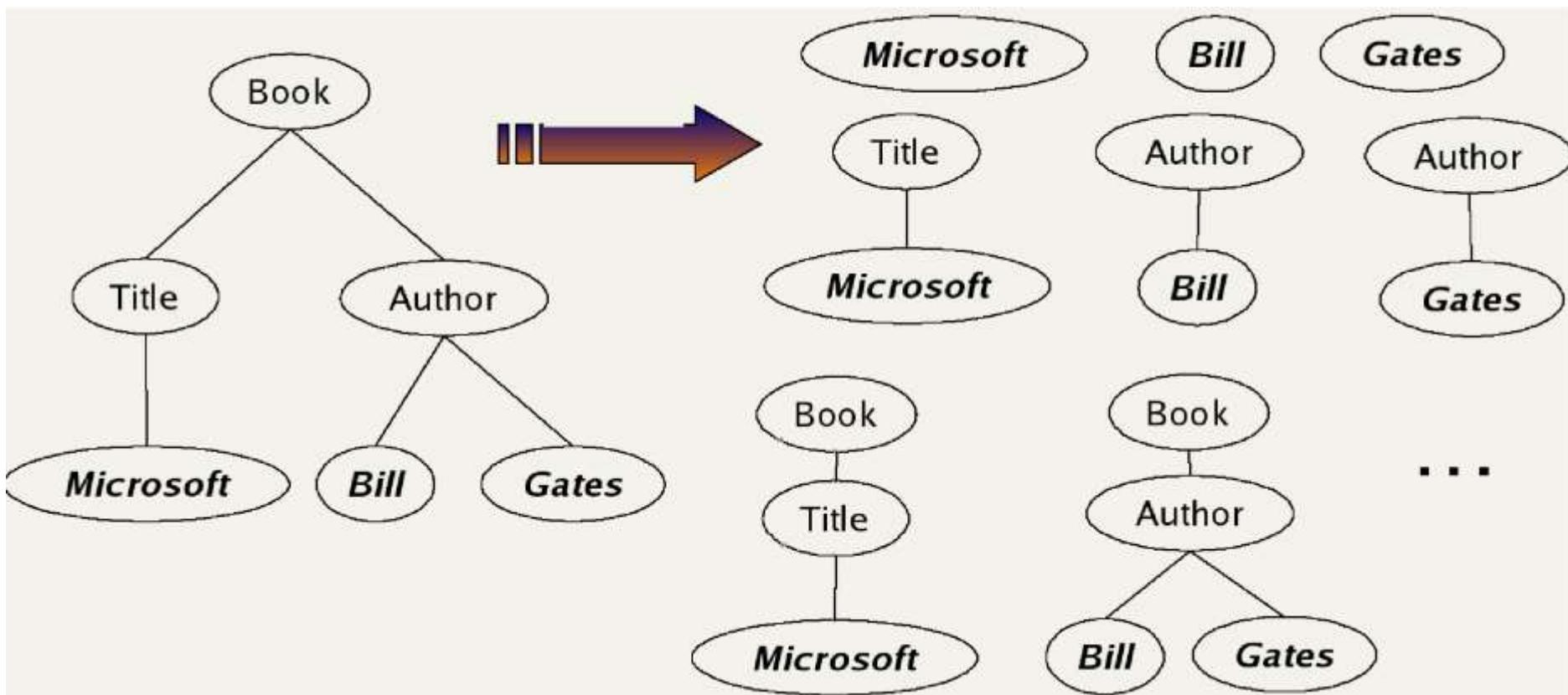
- sparse data problems (many XML-context pairs occur too rarely to reliably estimate df)

**compromise:** consider the parent node *x of the term and not the rest of the path from the root to x to distinguish contexts*

# Vector space model for XML IR

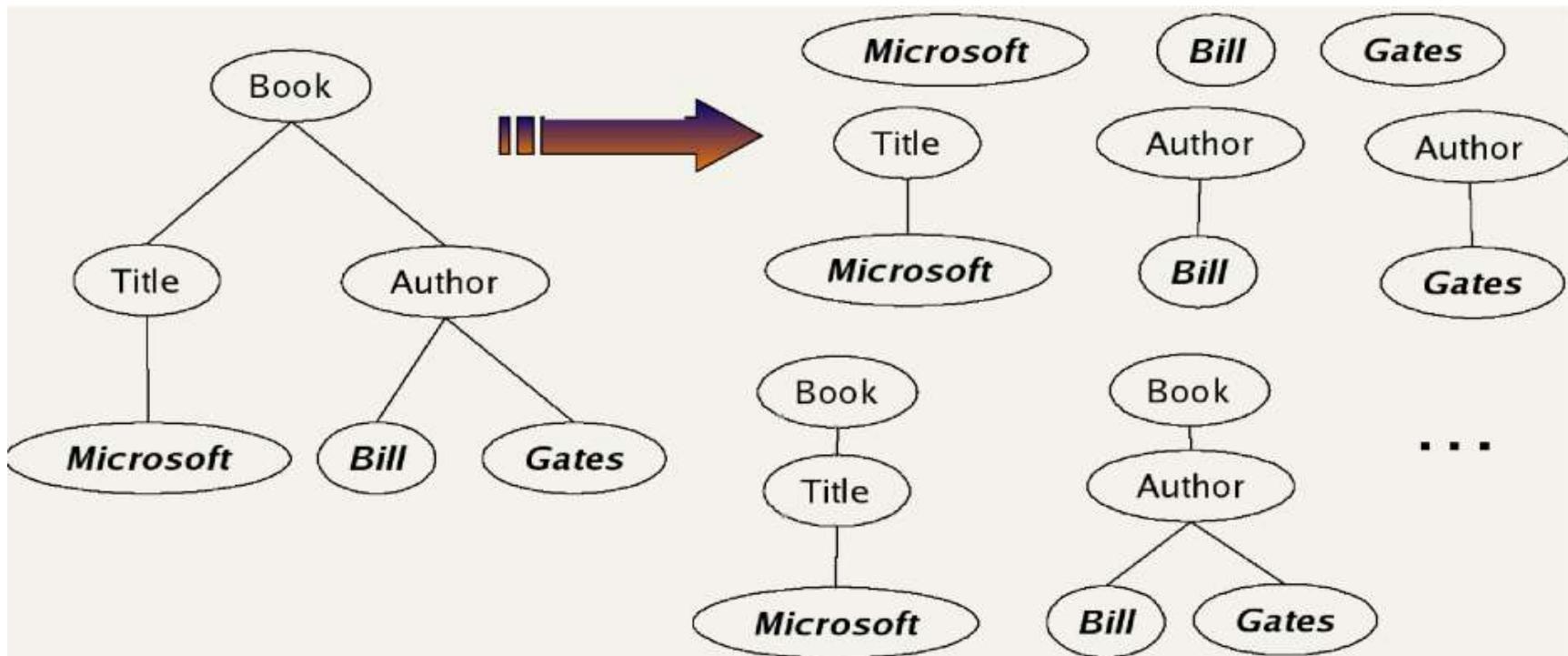
# Main idea: lexicalised subtrees

- Aim: to have each dimension of the vector space encode a word together with its position within the XML tree.
- How: Map XML documents to lexicalised subtrees.



# Main idea: lexicalised subtrees

- Take each text node (leaf) and break it into multiple nodes, one for each word. E.g. split *Bill Gates* into *Bill* and *Gates*.
- Define the dimensions of the vector space to be lexicalized subtrees of documents – subtrees that contain at least one vocabulary term.



# Lexicalised subtrees

- We can now represent queries and documents as vectors in this space of lexicalized subtrees and compute matches between them,  
e.g. using the vector space formalism.

Vector space formalism in **unstructured** **vs.**  
**structured IR**

- The main difference is that the dimensions of vector space in unstructured retrieval are vocabulary terms whereas they are lexicalized subtrees in XML retrieval.

# Structural term

- There is a tradeoff between the dimensionality of the space and accuracy of query results.
  - If we restrict dimensions to vocabulary terms, then we have a standard vector space retrieval system that will retrieve many documents that do not match the structure of the query (e.g., *Gates* in the title as opposed to the author element).
  - If we create a separate dimension for each lexicalized subtree occurring in the collection, the dimensionality of the space becomes too large.

**Compromise:** index all paths that end in a single vocabulary term, in other words, all XML-context term pairs. We call such an XML-context term pair a structural term and denote it by  $\langle c, t \rangle$ : a pair of XML-context  $c$  and vocabulary term  $t$ .

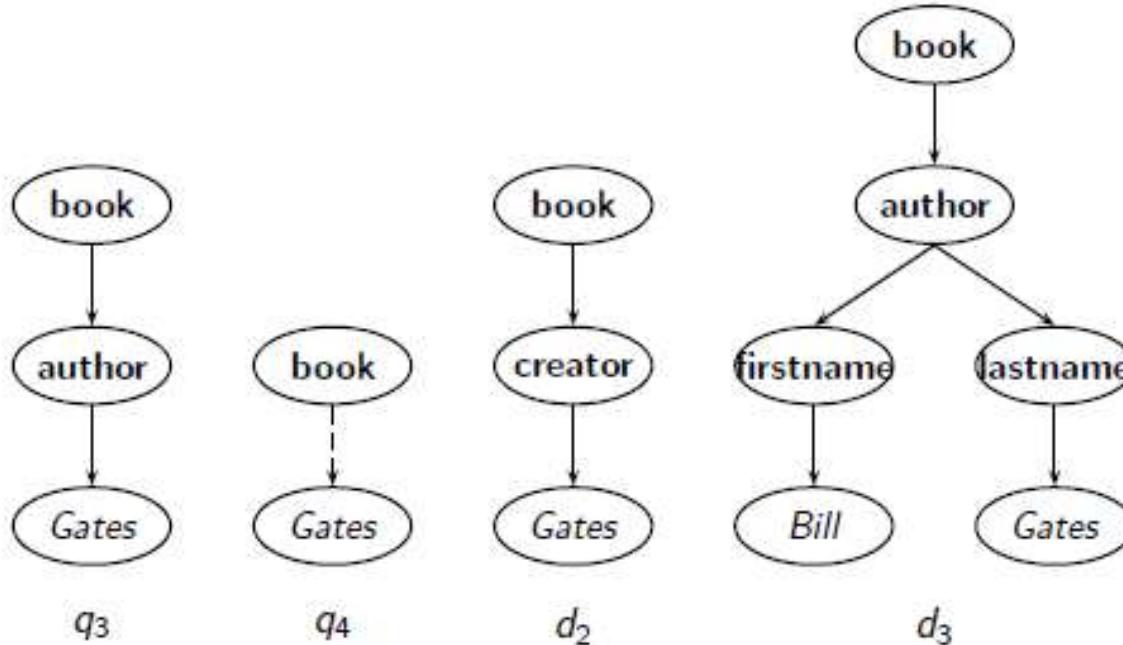
# Context resemblance

- A simple measure of the similarity of a path  $c_q$  in a query and a path  $c_d$  in a document is the following context resemblance function  $C_R$ :

$$C_R(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{if } c_q \text{ matches } c_d \\ 0 & \text{if } c_q \text{ does not match } c_d \end{cases} \quad (1)$$

- $|c_q|$  and  $|c_d|$  are the number of nodes in the query path and document path, resp.  
 $c_q$  matches  $c_d$  iff we can transform  $c_q$  into  $c_d$  by inserting additional nodes.

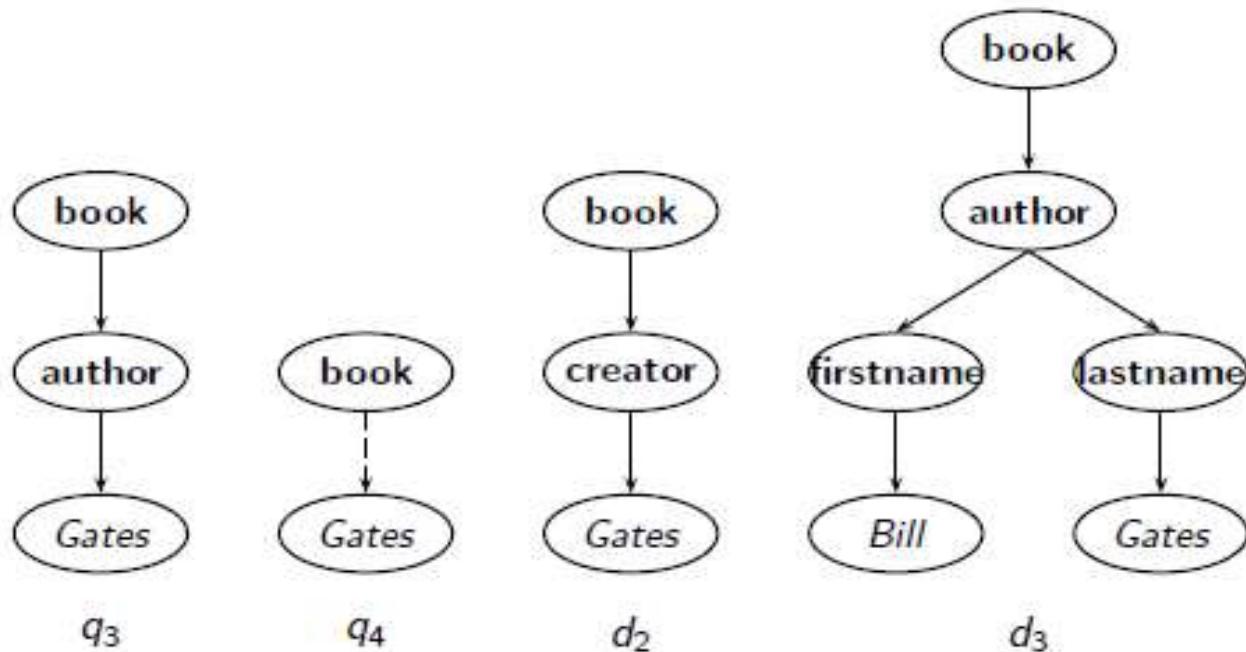
# Context resemblance example



$$\text{CR}(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{if } c_q \text{ matches } c_d \\ 0 & \text{if } c_q \text{ does not match } c_d \end{cases}$$

$\text{CR}(c_{q_4}, c_{d_2}) = 3/4 = 0.75$ . The value of  $\text{CR}(c_q, c_d)$  is 1.0 if  $q$  and  $d$  are identical.

# Context resemblance exercise



$$\text{CR}(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{if } c_q \text{ matches } c_d \\ 0 & \text{if } c_q \text{ does not match } c_d \end{cases}$$

$$\text{CR}(c_{q_4}, c_{d_3}) = ? \quad \text{CR}(c_{q_4}, c_{d_3}) = 3/5 = 0.6.$$

# Document similarity measure

The final score for a document is computed as a variant of the cosine measure, which we call SIMNoMERGE.

SIMNoMERGE( $q, d$ ) =

$$\sum_{c_k \in B} \sum_{c_l \in B} \text{CR}(c_k, c_l) \sum_{t \in V} \text{weight}(q, t, c_k) \frac{\text{weight}(d, t, c_l)}{\sqrt{\sum_{c \in B, t \in V} \text{weight}^2(d, t, c)}}$$

- $V$  is the vocabulary of non-structural terms
- $B$  is the set of all XML contexts
- $\text{weight}(q, t, c)$ ,  $\text{weight}(d, t, c)$  are the weights of term  $t$  in XML context  $c$  in query  $q$  and document  $d$ , resp. (standard weighting e.g.  $\text{idf}_t \cdot \text{wf}_{t,d}$ , where  $\text{idf}_t$  depends on which elements we use to compute  $\text{df}_t$ . )

SIMNoMERGE( $q, d$ ) is not a true cosine measure since its value can be larger than 1.0.

# Evaluation of XML Retrieval

# Initiative for the Evaluation of XML Retrieval (INEX)

- INEX: standard benchmark evaluation (yearly) that has produced test collections (documents, sets of queries, and relevance judgments).
- Based on IEEE journal collection (since 2006 INEX uses the much larger English Wikipedia as a test collection).
- The relevance of documents is judged by human assessors.

## INEX 2002 collection statistics

12,107	number of documents
494 MB	size
1995–2002	time of publication of articles
1,532	average number of XML nodes per document
6.9	average depth of a node
30	number of CAS topics
30	number of CO topics

# INEX topics

- Two types:
  - content-only or CO topics: regular keyword queries as in unstructured information retrieval
  - content-and-structure or CAS topics: have structural constraints in addition to keywords
- Since CAS queries have both structural and content criteria, relevance assessments are more complicated than in unstructured retrieval.

# INEX relevance assessments

- INEX 2002 defined component coverage and topical relevance as orthogonal dimensions of relevance.

## Component coverage

- Evaluates whether the element retrieved is “structurally” correct, i.e., neither too low nor too high in the tree.

We distinguish four cases:

- Exact coverage (E): The information sought is the main topic of the component and the component is a meaningful unit of information.
- Too small (S): The information sought is the main topic of the component, but the component is not a meaningful (self-contained) unit of information.
- Too large (L): The information sought is present in the component, but is not the main topic.
- No coverage (N): The information sought is not a topic of the Component.

# INEX relevance assessments

The topical relevance dimension also has four levels: highly relevant (3), fairly relevant (2), marginally relevant (1) and nonrelevant (0).

## Combining the relevance dimensions

Components are judged on both dimensions and the judgments are then combined into a digit-letter code, e.g. 2S is a fairly relevant component that is too small. In theory, there are 16 combinations of coverage and relevance, but many cannot occur. For example, a nonrelevant component cannot have exact coverage, so the combination 3N is not possible.

# INEX relevance assessments

- The relevance-coverage combinations are quantized as follow

$$Q(rel, cov) = \begin{cases} 1.00 & \text{if } (rel, cov) = 3E \\ 0.75 & \text{if } (rel, cov) \in \{2E, 3L\} \\ 0.50 & \text{if } (rel, cov) \in \{1E, 2L, 2S\} \\ 0.25 & \text{if } (rel, cov) \in \{1S, 1L\} \\ 0.00 & \text{if } (rel, cov) = 0N \end{cases}$$

- This evaluation scheme takes account of the fact that binary relevance judgments, which are standard in unstructured IR, are not appropriate for XML retrieval. The quantization function  $Q$  does not impose a binary choice relevant / non-relevant and instead allows us to grade the component as partially relevant. The number of relevant components  $\sum_{c \in A} Q(rel(c), cov(c))$  can then be computed.

# INEX evaluation measures

- Recent INEX focus: develop algorithms and evaluation measures that return non-redundant results lists and evaluate them properly.

# Thank You

# Information Retrieval

## Topic- Text Classification & Naive Bayes

### (Text Classification )

## Lecture-34

**Prepared By**

Dr. Rasmita Rautray & Dr. Rasmita Dash  
Associate Professor  
Dept. of CSE

# Text Classification

# Content

- Text Classification
- Classification methods
- Naive Bayes classifier

# Relevance feedback: Basic idea

- The user issues a (short, simple) query.
- The search engine returns a set of documents.
- User marks some docs as relevant, some as nonrelevant.
- Search engine computes a new representation of the information need – should be better than the initial query.
- Search engine runs new query and returns new results.
- New results have (hopefully) better recall.

# Types of query expansion

- Manual thesaurus (maintained by editors, e.g., PubMed)
- Automatically derived thesaurus (e.g., based on co-occurrence statistics)
- Query-equivalence based on query log mining (common on the web as in the “palm” example)

# Query expansion at search engines

- Main source of query expansion at search engines: query logs
- Example 1: After issuing the query [herbs], users frequently search for [herbal remedies].  
→ “herbal remedies” is potential expansion of “herb”.
- Example 2: Users searching for [flower pix] frequently click on the URL photobucket.com/flower. Users searching for [flower clipart] frequently click on the same URL.  
→ “flower clipart” and “flower pix” are potential expansions of each other.

# Text classification: definition & relevance to information retrieval

# A text classification task: Email spam filtering

From: "" <takworlld@hotmail.com>

Subject: real estate is the only way... gem oalvgkay

Anyone can buy real estate with no money down

Stop paying rent TODAY !

There is no need to spend hundreds or even thousands for similar courses

I am 22 years old and I have already purchased 6 properties using the

methods outlined in this truly INCREDIBLE ebook.

Change your life NOW !

---

---

=

Click Below to order:

<http://www.wholesaledaily.com/sales/nmd.htm>

---

---

# Formal definition of TC: Training

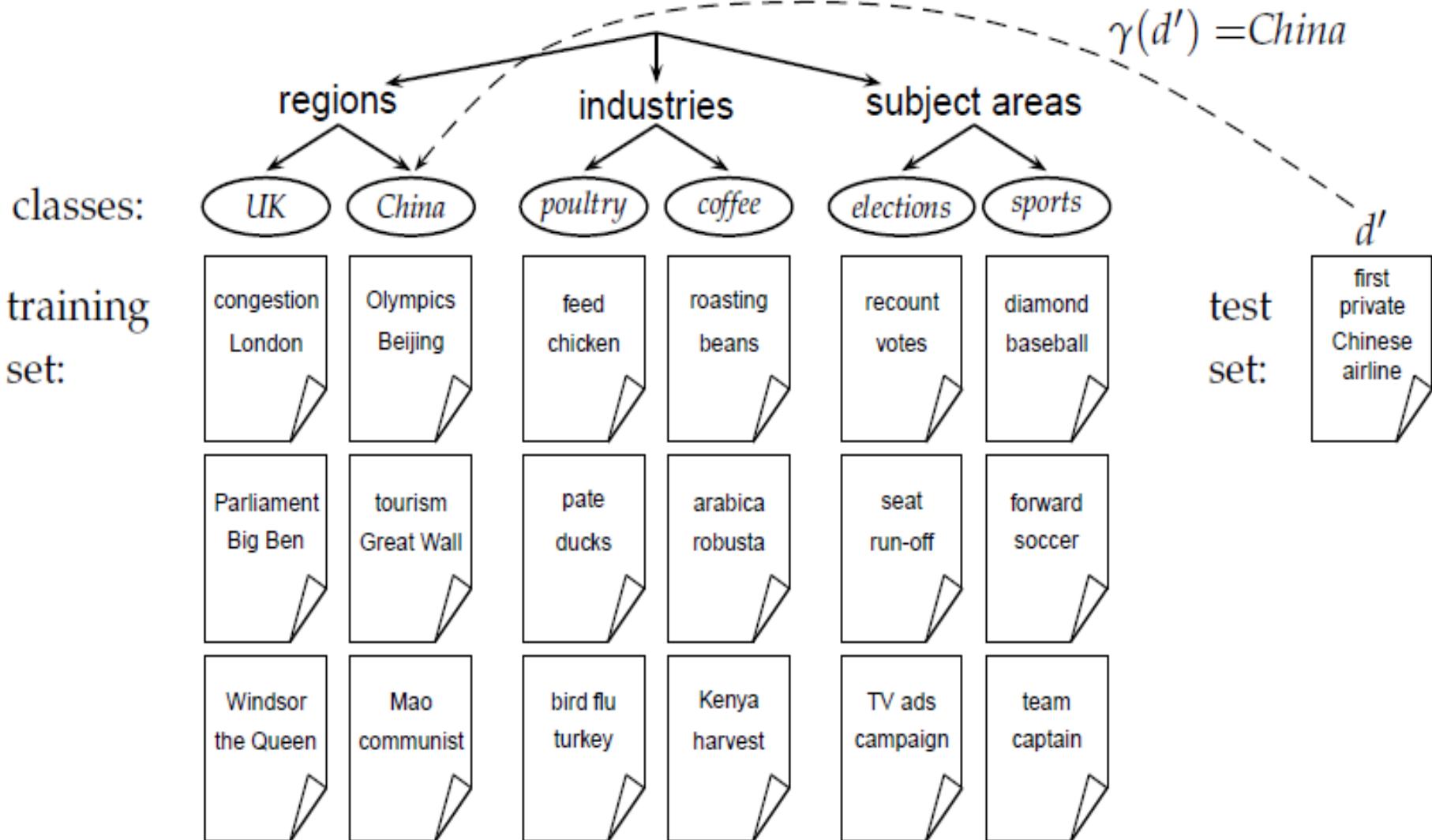
- Given:
- A document space  $X$ 
  - Documents are represented in this space – typically some type of high-dimensional space.
- A fixed set of classes  $C = \{c_1, c_2, \dots, c_J\}$ 
  - The classes are human-defined for the needs of an application (e.g., spam vs. nonspam).
- A training set  $D$  of labeled documents. Each labeled ~~document~~
- Using a learning method or learning algorithm, we then wish to learn a classifier  $\gamma$  that maps documents to classes:

$$\gamma : X \rightarrow C$$

# Formal definition of TC: Application / Testing

- Given: a description  $d \in X$  of a document
- Determine:  
 $\gamma(d) \in C$ ,  
that is, the class that is most appropriate for  $d$

# Topic classification



# Examples of how search engines use classification

- Language identification (classes: English vs. French etc.)
- The automatic detection of spam pages (spam vs. nonspam)
- Sentiment detection: is a movie or product review positive or negative (positive vs. negative)
- Topic-specific or vertical search – restrict search to a “vertical” like “related to health” (relevant to vertical vs. not)

# Classification methods: 1. Manual

- Manual classification was used by Yahoo in the beginning of the web. Also: ODP, PubMed
- Very accurate if job is done by experts
- Consistent when the problem size and team is small
- Scaling manual classification is difficult and expensive.  
→ We need automatic methods for classification.

# Classification methods: 2. Rule-based

- E.g., Google Alerts is rule-based classification.
- There are IDE-type development environments for writing very complex rules efficiently. (e.g., Verity)
- Often: Boolean combinations (as in Google Alerts)
- Accuracy is very high if a rule has been carefully refined over time by a subject expert.
- Building and maintaining rule-based classification systems is cumbersome and expensive.

# Classification methods: 3

## Statistical/Probabilistic

- This was our definition of the classification problem – text classification as a learning problem
  - (i) Supervised learning of a the classification function  $\gamma$  and
  - (ii) application of  $\gamma$  to classifying new documents
- We will look at two methods for doing this: Naive Bayes and SVMs
- No free lunch: requires hand-classified training data
- But this manual classification can be done by non-experts.

# The Naive Bayes classifier

- The Naive Bayes classifier is a probabilistic classifier.
- We compute the probability of a document  $d$  being in a class

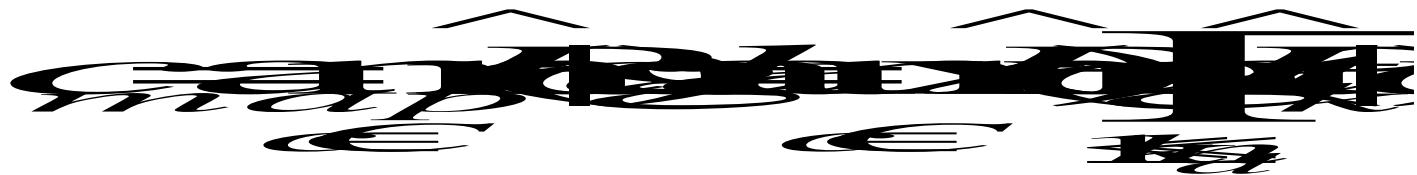
$c$  as follows:

$$\frac{P(c) \prod_{k=1}^{n_d} P(t_k | c)}{\prod_{k=1}^{n_d} P(t_k)}$$

- $n_d$  is the length of the document. (number of tokens)
- $P(t_k | c)$  is the conditional probability of term  $t_k$  occurring in a document of class  $c$
- $P(t_k | c)$  as a measure of how much evidence  $t_k$  contributes that  $c$  is the correct class.
- $P(c)$  is the prior probability of  $c$ .
- If a document's terms do not provide clear evidence for one class over another, we choose the one with highest  $P(c)$

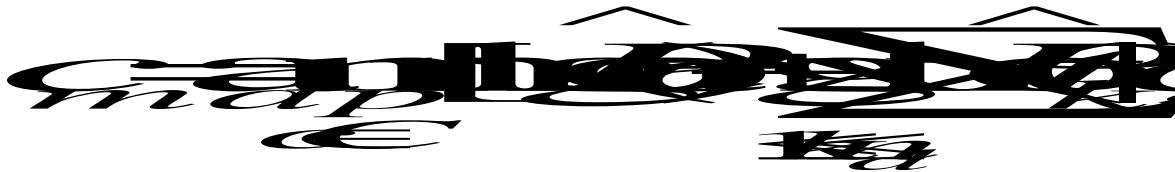
# Maximum a posteriori class

- Our goal in Naive Bayes classification is to find the “best” class.
- The best class is the most likely or maximum a posteriori (MAP) class  $c_{\text{map}}$ :



# Naive Bayes classifier

Classification rule:



- Simple interpretation:
  - Each conditional parameter  $\log \hat{P}(t_k | c)$  is a weight that indicates how good an indicator  $t_k$  is for  $c$ .
  - The prior  $\log \hat{P}(c)$  is a weight that indicates the relative frequency of  $c$ .
  - The sum of log prior and term weights is then a measure of how much evidence there is for the document being in the class.
  - We select the class with the most evidence.

# Thank You

# Information Retrieval

## Topic- Text Clustering (K-means clustering)

### Lecture-36

**Prepared By**

Dr. Rasmita Rautray & Dr. Rasmita Dash  
Associate Professor  
Dept. of CSE

# Text Clustering

# Content

- Clustering
- Classification Vs Clustering
- Applications of clustering in information retrieval
- K-mean Clustering

# Clustering: Definition

- (Document) clustering is the process of grouping a set of documents into clusters of similar documents.
- Documents within a cluster should be similar.
- Documents from different clusters should be dissimilar.
- Clustering is the most common form of unsupervised learning.
- Unsupervised = there are no labeled or annotated data.

# Classification vs. Clustering

- Classification: supervised learning
- Clustering: unsupervised learning
- Classification: Classes are human-defined and part of the input to the learning algorithm.
- Clustering: Clusters are inferred from the data without human input.
- However, there are many ways of influencing the outcome of clustering: number of clusters, similarity measure, representation of documents, . . .

# Clustering in IR

# The cluster hypothesis

- Cluster hypothesis. Documents in the same cluster behave similarly with respect to relevance to information needs.
- All applications of clustering in IR are based (directly or indirectly) on the cluster hypothesis.
- Van Rijsbergen's original wording (1979): "closely associated documents tend to be relevant to the same requests".

# Applications of clustering in IR

<b>Application</b>	<b>What is clustered?</b>	<b>Benefit</b>
search result clustering	search results	more effective information presentation to user
Scatter-Gather	(subsets of) collection	alternative user interface: “search without typing”
collection clustering	collection	effective information presentation for exploratory Browsing
cluster-based retrieval	collection	higher efficiency: faster search

# Search result clustering for better navigation



Vivísimo®

jaguar

the Web ▾

Search Advanced  
Search Help

**Clustered Results**

Top 208 results of at least 20,373,974 retrieved for the query **jaguar** (Details)

▶ [jaguar \(208\)](#)

+▶ [Cars \(74\)](#)

+▶ [Club \(34\)](#)

+▶ [Cat \(23\)](#)

+▶ [Animal \(13\)](#)

+▶ [Restoration \(10\)](#)

+▶ [Mac OS X \(8\)](#)

+▶ [Jaguar Model \(8\)](#)

+▶ [Request \(5\)](#)

+▶ [Mark Webber \(6\)](#)

+▶ [Maya \(5\)](#)

▼ [More](#)

Find in clusters:

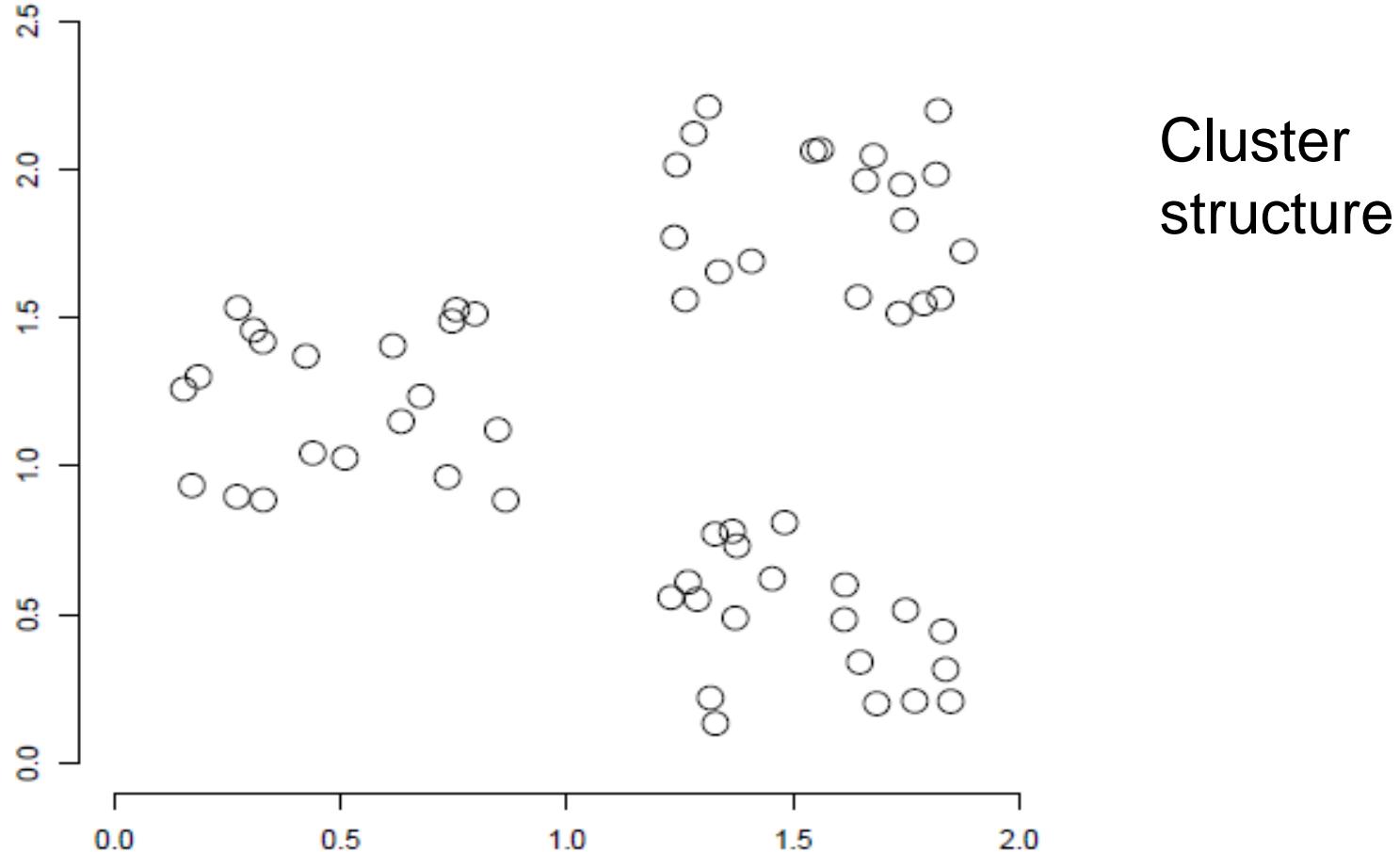
Enter Keywords  Go

1. [Jag-lovers - THE source for all Jaguar information](#) [new window] [frame] [cache] [preview] [clusters]  
... Internet! Serving Enthusiasts since 1993 The Jag-lovers Web Currently with 40661 members The Premier **Jaguar** Cars web resource for all enthusiasts Lists and Forums Jag-lovers originally evolved around its ...  
[www.jag-lovers.org](http://www.jag-lovers.org) - Open Directory 2, Wisenut 8, Ask Jeeves 8, MSN 9, Looksmart 12, MSN Search 18
2. [Jaguar Cars](#) [new window] [frame] [cache] [preview] [clusters]  
[...] redirected to [www.jaguar.com](http://www.jaguar.com)  
[www.jaguarcars.com](http://www.jaguarcars.com) - Looksmart 1, MSN 2, Lycos 3, Wisenut 6, MSN Search 9, MSN 29
3. <http://www.jaguar.com/> [new window] [frame] [preview] [clusters]  
[www.jaguar.com](http://www.jaguar.com) - MSN 1, Ask Jeeves 1, MSN Search 3, Lycos 9
4. [Apple - Mac OS X](#) [new window] [frame] [preview] [clusters]  
Learn about the new OS X Server, designed for the Internet, digital media and workgroup management.  
Download a technical factsheet.  
[www.apple.com/macosx](http://www.apple.com/macosx) - Wisenut 1, MSN 3, Looksmart 26

# Clustering for improving recall

- To improve search recall:
  - Cluster docs in collection a priori
  - When a query matches a doc  $d$ , also return other docs in the cluster containing  $d$
- Hope: if we do this: the query “car” will also return docs containing “automobile”
  - Because the clustering algorithm groups together docs containing “car” with those containing “automobile”.
  - Both types of documents contain words like “parts”, “dealer”, “mercedes”, “road trip”.

# Data set with clear cluster structure



# Desiderata for clustering

- General goal: put related docs in the same cluster, put unrelated docs in different clusters.
  - We'll see different ways of formalizing this.
- The number of clusters should be appropriate for the data set we are clustering.
  - Initially, we will assume the number of clusters  $K$  is given.
  - Later: Semiautomatic methods for determining  $K$
- Secondary goals in clustering
  - Avoid very small and very large clusters
  - Define clusters that are easy to explain to the user
  - Many others . . .

# Flat vs. Hierarchical clustering

- Flat algorithms
  - Usually start with a random (partial) partitioning of docs into groups
  - Refine iteratively
  - Main algorithm: K-means
- Hierarchical algorithms
  - Create a hierarchy
  - Bottom-up, agglomerative
  - Top-down, divisive

# Hard vs. Soft clustering

- Hard clustering: Each document belongs to exactly one cluster.
  - More common and easier to do
- Soft clustering: A document can belong to more than one cluster.
  - Makes more sense for applications like creating browsable hierarchies
  - You may want to put sneakers in two clusters:
    - sports apparel
    - shoes
  - You can only do that with a soft clustering approach.

# Flat algorithms

- Flat algorithms compute a partition of  $N$  documents into a set of  $K$  clusters.
- Given: a set of documents and the number  $K$
- Find: a partition into  $K$  clusters that optimizes the chosen partitioning criterion
- Global optimization: exhaustively enumerate partitions, pick optimal one
  - Not tractable
- Effective heuristic method: K-means algorithm

# K-means

- The best known clustering algorithm
- Simple, works well in many cases
- Use as default / baseline for clustering documents

# Document representations in clustering

- Vector space model
- As in vector space classification, we measure relatedness
- between vectors by Euclidean distance . . .
- . . . which is almost equivalent to cosine similarity.
- Almost: centroids are not length-normalized.

# K-means: Basic idea

- Each cluster in K-means is defined by a centroid.
- Objective/partitioning criterion: minimize the average squared difference from the centroid
- Recall definition of centroid:

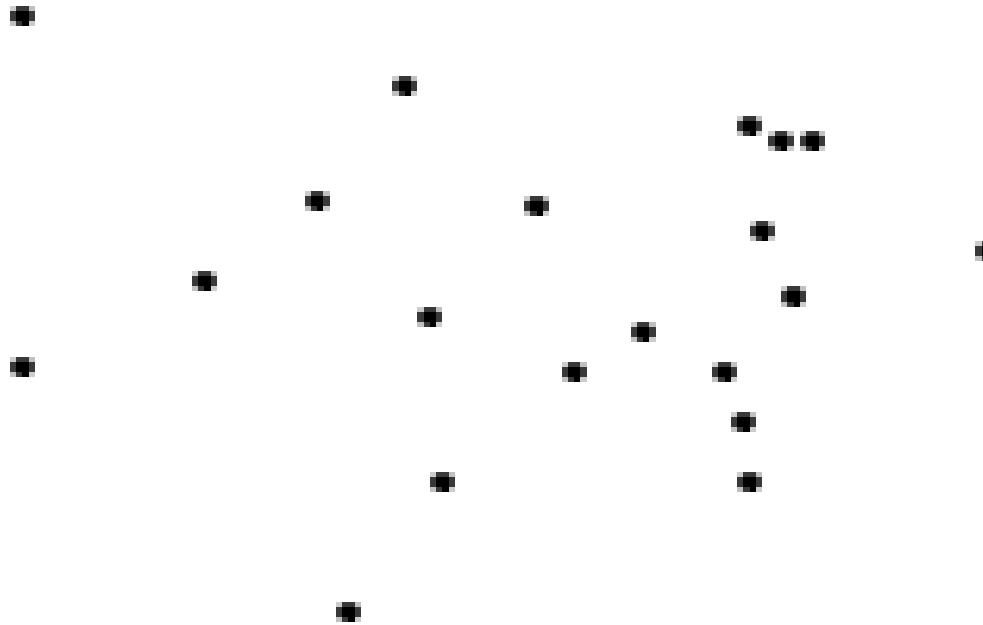
$$\mu_{\omega} = \frac{1}{|W_{\omega}|} \sum_{x \in W_{\omega}} x$$

where we use  $\omega$  to denote a cluster.

- We try to find the minimum average squared difference by iterating two steps:

- reassignment: assign each vector to its closest centroid
- recomputation: recompute each centroid as the average of the vectors that were assigned to it in reassignment

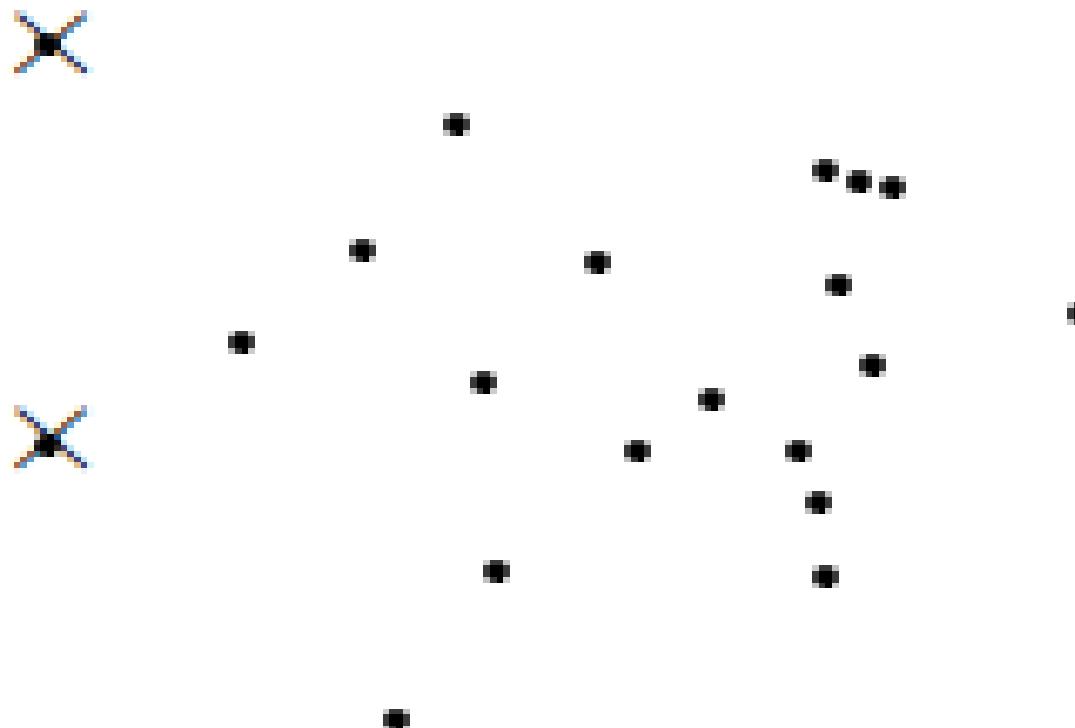
# Example: Set of points to be clustered



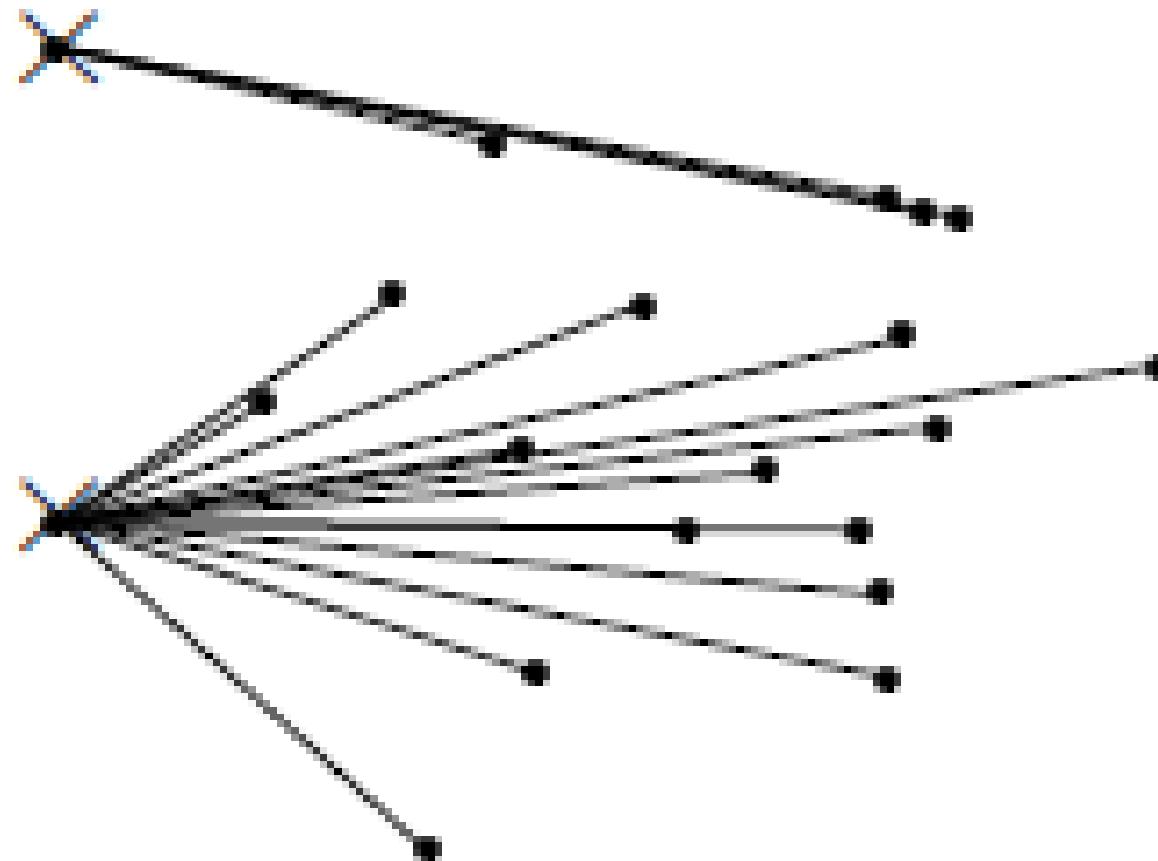
Exercise:

- (i) Guess what the optimal clustering into two clusters is in this case;
- (ii) compute the centroids of the clusters

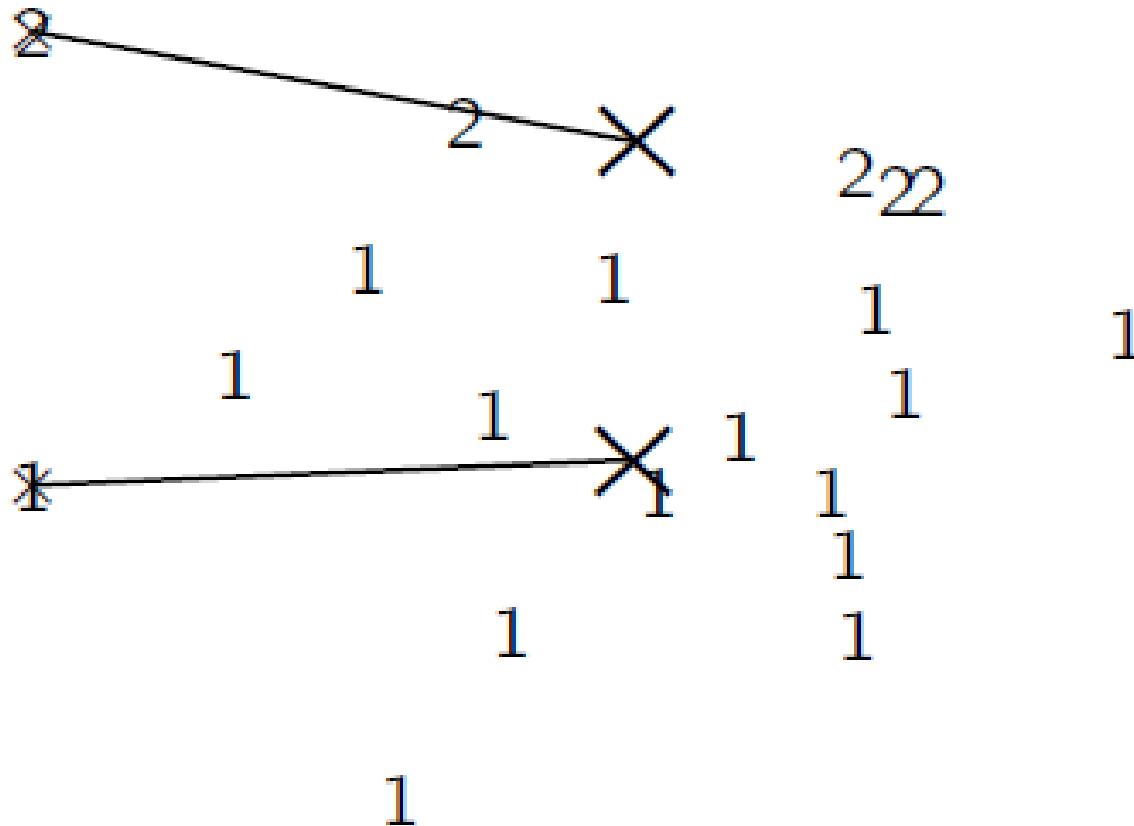
# Example: Random selection of initial centroids



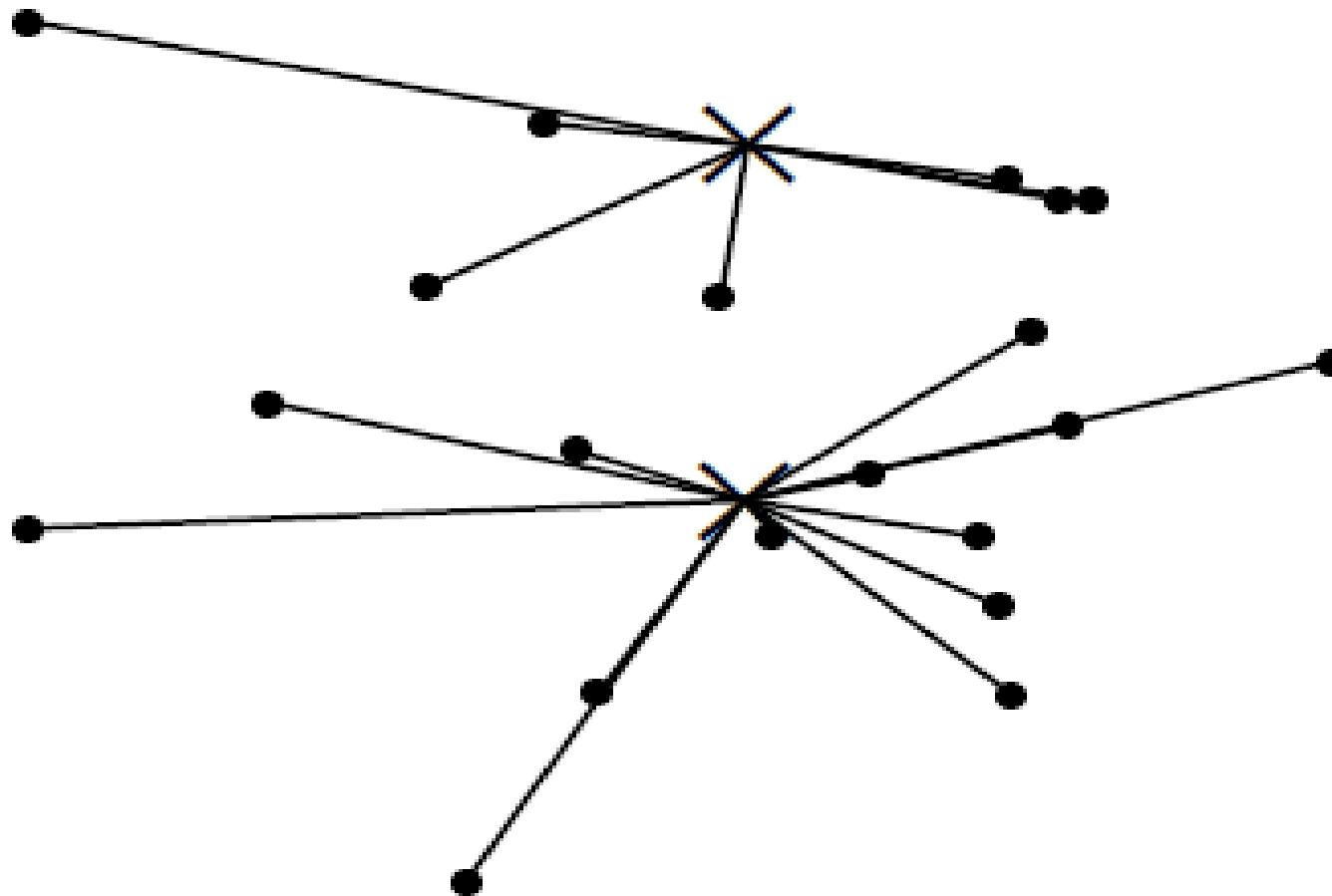
# Example: Assign points to closest center



# Example: Recompute cluster centroids



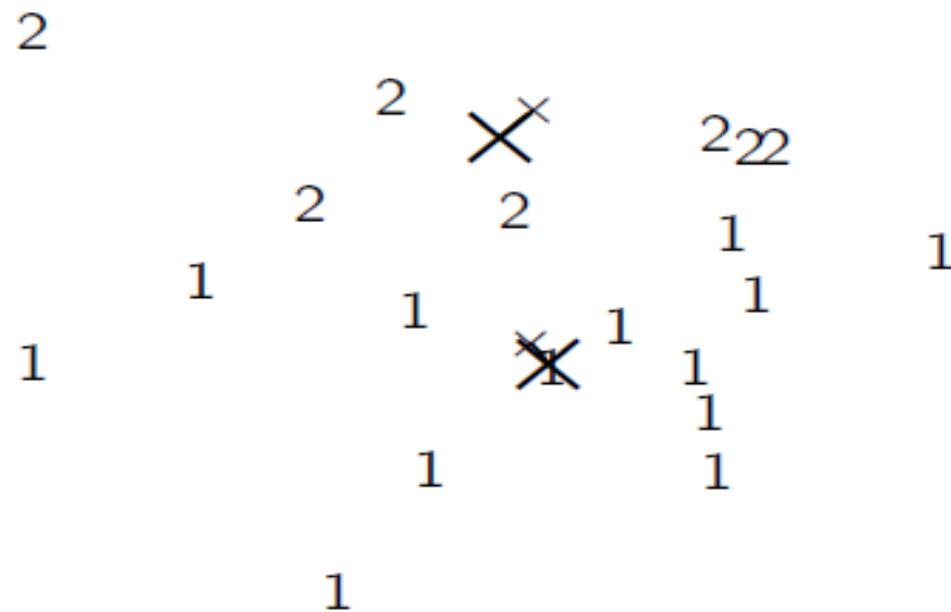
# Example: Assign points to closest centroid



# Example: Assignment

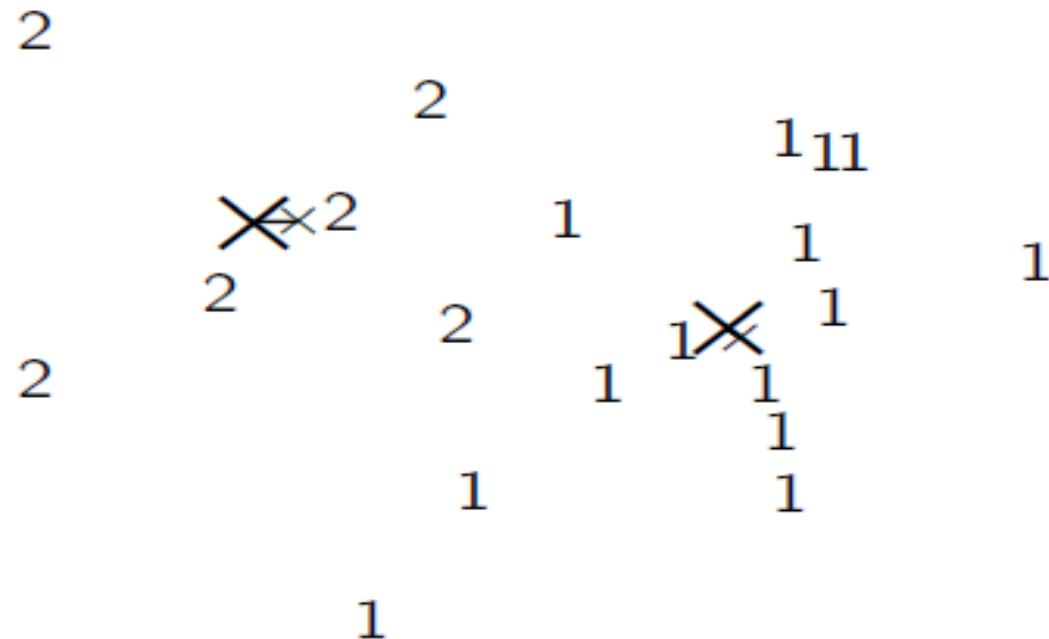
$$\begin{array}{r} 2 \\ \times 22 \\ \hline 11 \\ + 11 \\ \hline 44 \end{array}$$

# Example: Recompute cluster centroids

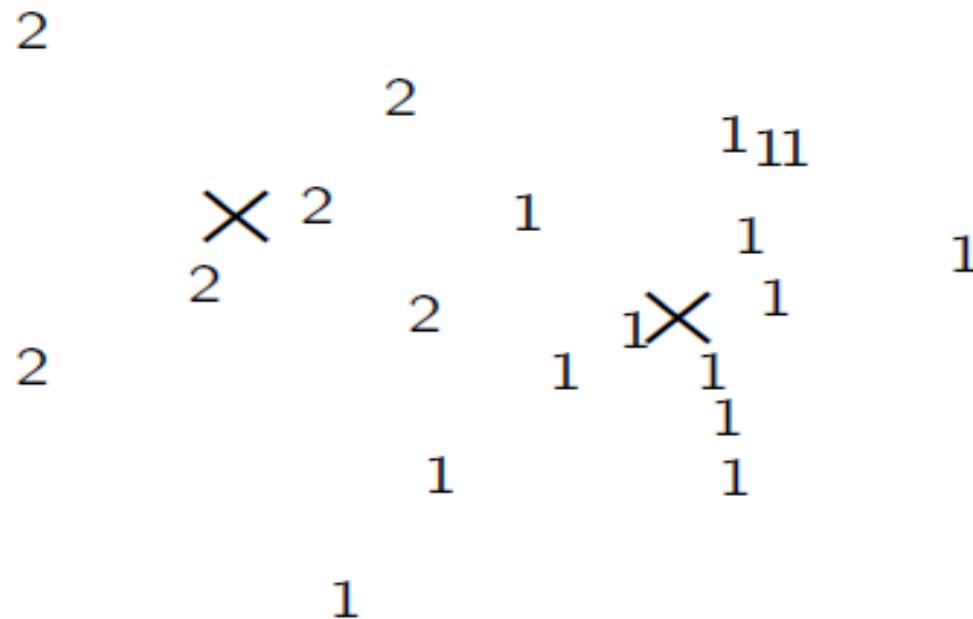




# Example: Recompute cluster centroids



# Example: Centroids and assignments after convergence



# Thank You

# Information Retrieval

## Topic- Text Clustering

### (Application of K-means clustering)

## Lecture-37

**Prepared By**

Dr. Rasmita Rautray & Dr. Rasmita Dash  
Associate Professor  
Dept. of CSE

# Text Clustering

# Content

- Clustering
- Classification Vs Clustering
- Applications of clustering in information retrieval
- K-mean Clustering

# Clustering: Definition

- (Document) clustering is the process of grouping a set of documents into clusters of similar documents.
- Documents within a cluster should be similar.
- Documents from different clusters should be dissimilar.
- Clustering is the most common form of unsupervised learning.
- Unsupervised = there are no labeled or annotated data.

# Classification vs. Clustering

- Classification: supervised learning
- Clustering: unsupervised learning
- Classification: Classes are human-defined and part of the input to the learning algorithm.
- Clustering: Clusters are inferred from the data without human input.
- However, there are many ways of influencing the outcome of clustering: number of clusters, similarity measure, representation of documents, . . .

# The cluster hypothesis

- Cluster hypothesis. Documents in the same cluster behave similarly with respect to relevance to information needs.
- All applications of clustering in IR are based (directly or indirectly) on the cluster hypothesis.

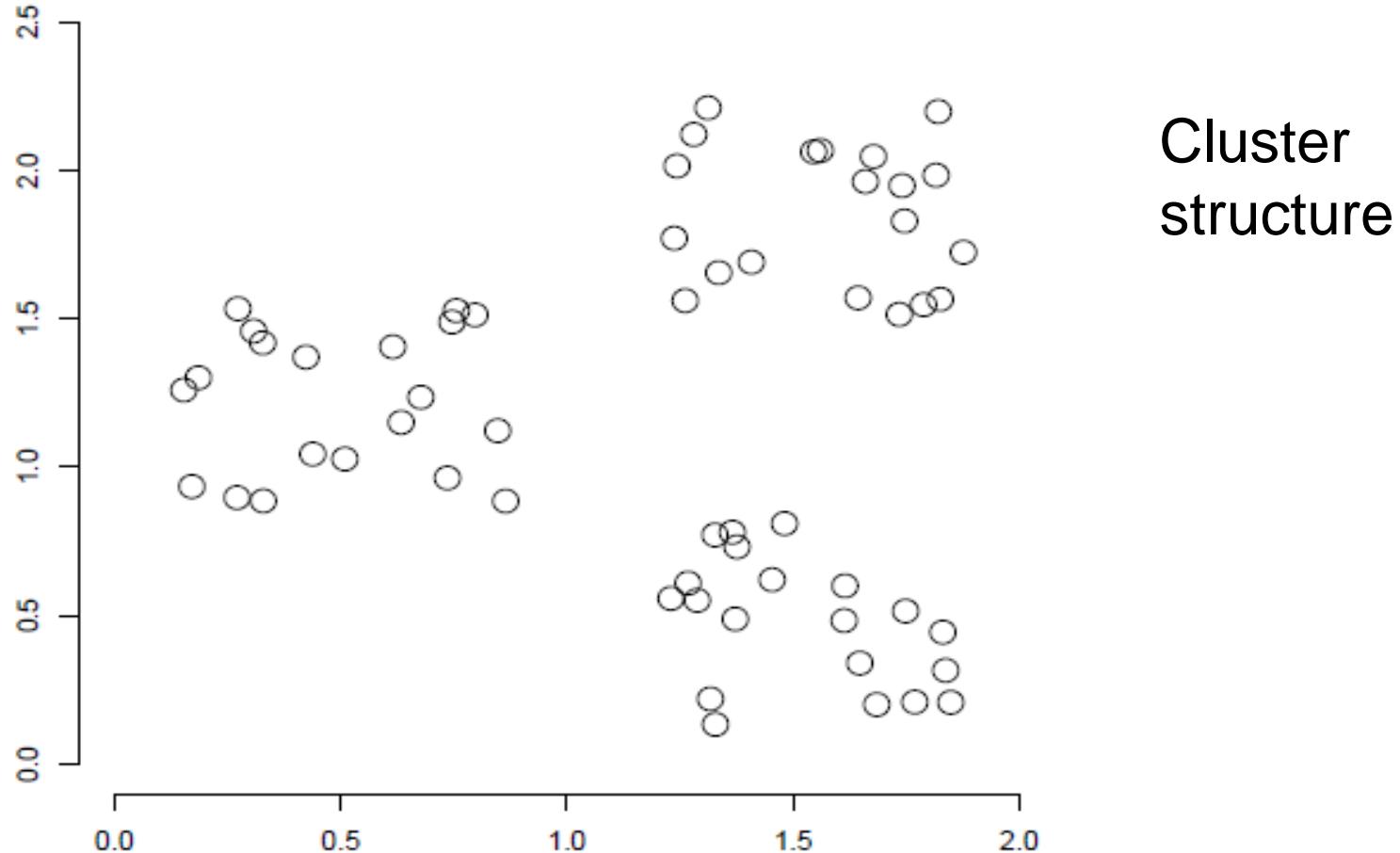
# Applications of clustering in IR

<b>Application</b>	<b>What is clustered?</b>	<b>Benefit</b>
search result clustering	search results	more effective information presentation to user
Scatter-Gather	(subsets of) collection	alternative user interface: “search without typing”
collection clustering	collection	effective information presentation for exploratory Browsing
cluster-based retrieval	collection	higher efficiency: faster search

# Clustering for improving recall

- To improve search recall:
  - Cluster docs in collection a priori
  - When a query matches a doc  $d$ , also return other docs in the cluster containing  $d$
- Hope: if we do this: the query “car” will also return docs containing “automobile”
  - Because the clustering algorithm groups together docs containing “car” with those containing “automobile”.
  - Both types of documents contain words like “parts”, “dealer”, “mercedes”, “road trip”.

# Data set with clear cluster structure



# Desiderata for clustering

- General goal: put related docs in the same cluster, put unrelated docs in different clusters.
  - We'll see different ways of formalizing this.
- The number of clusters should be appropriate for the data set we are clustering.
  - Initially, we will assume the number of clusters  $K$  is given.
  - Later: Semiautomatic methods for determining  $K$
- Secondary goals in clustering
  - Avoid very small and very large clusters
  - Define clusters that are easy to explain to the user
  - Many others . . .

# Flat algorithms

- Flat algorithms compute a partition of  $N$  documents into a set of  $K$  clusters.
- Given: a set of documents and the number  $K$
- Find: a partition into  $K$  clusters that optimizes the chosen partitioning criterion
- Global optimization: exhaustively enumerate partitions, pick optimal one
  - Not tractable
- Effective heuristic method: K-means algorithm

# K-means

- The best known clustering algorithm
- Simple, works well in many cases
- Use as default / baseline for clustering documents

# Document representations in clustering

- Vector space model
- As in vector space classification, we measure relatedness
- between vectors by Euclidean distance . . .
- . . . which is almost equivalent to cosine similarity.
- Almost: centroids are not length-normalized.

# K-means: Basic idea

- Each cluster in K-means is defined by a centroid.
- Objective/partitioning criterion: minimize the average squared difference from the centroid
- Recall definition of centroid:

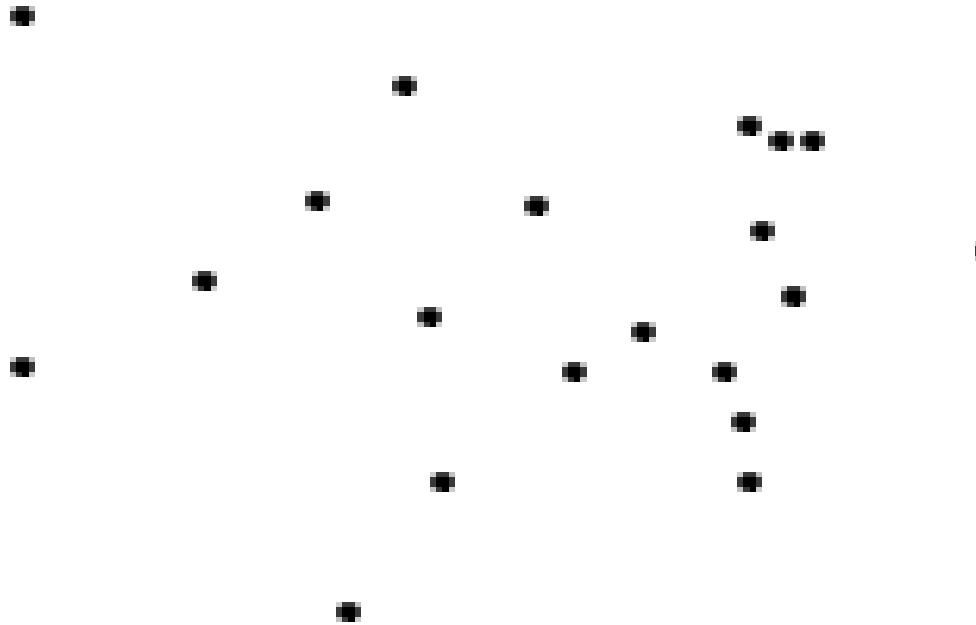
$$\mu_{\omega} = \frac{1}{|W_{\omega}|} \sum_{x \in W_{\omega}} x$$

where we use  $\omega$  to denote a cluster.

- We try to find the minimum average squared difference by iterating two steps:

- reassignment: assign each vector to its closest centroid
- recomputation: recompute each centroid as the average of the vectors that were assigned to it in reassignment

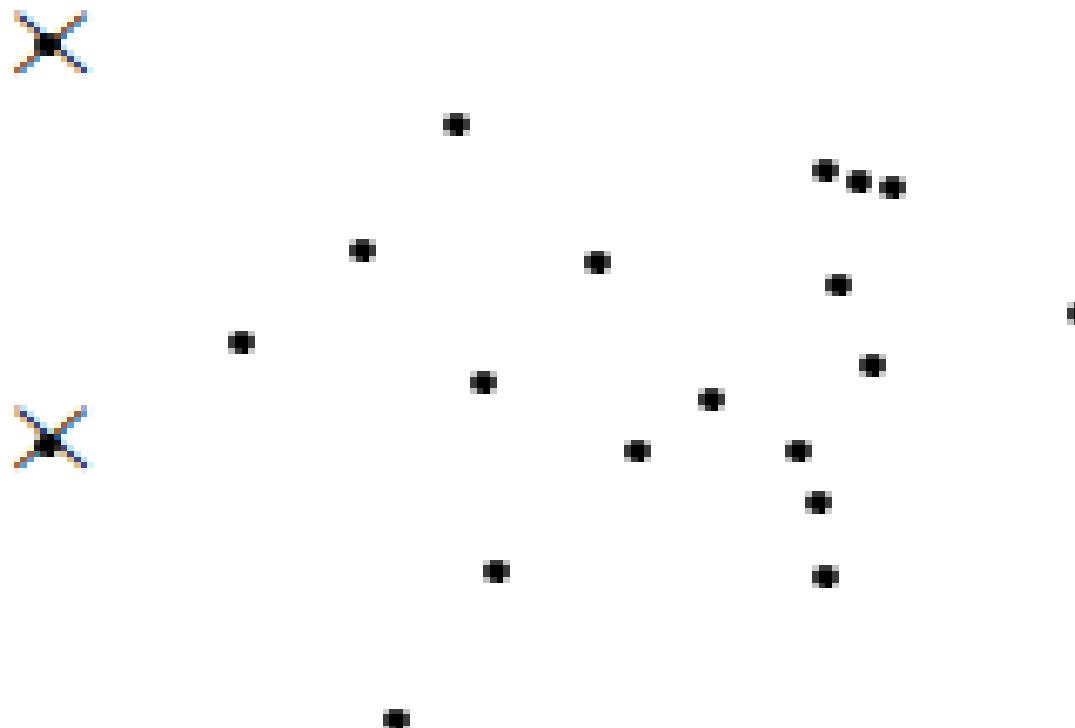
# Example: Set of points to be clustered



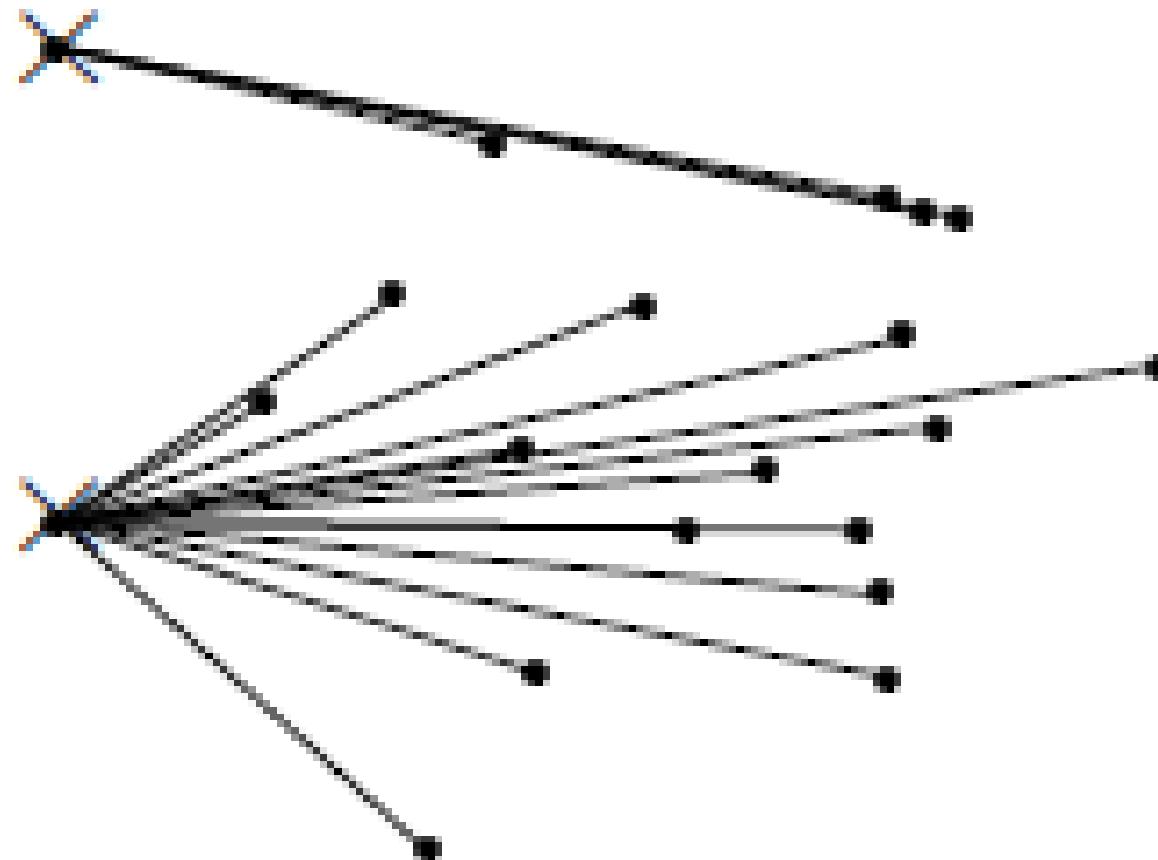
Exercise:

- (i) Guess what the optimal clustering into two clusters is in this case;
- (ii) compute the centroids of the clusters

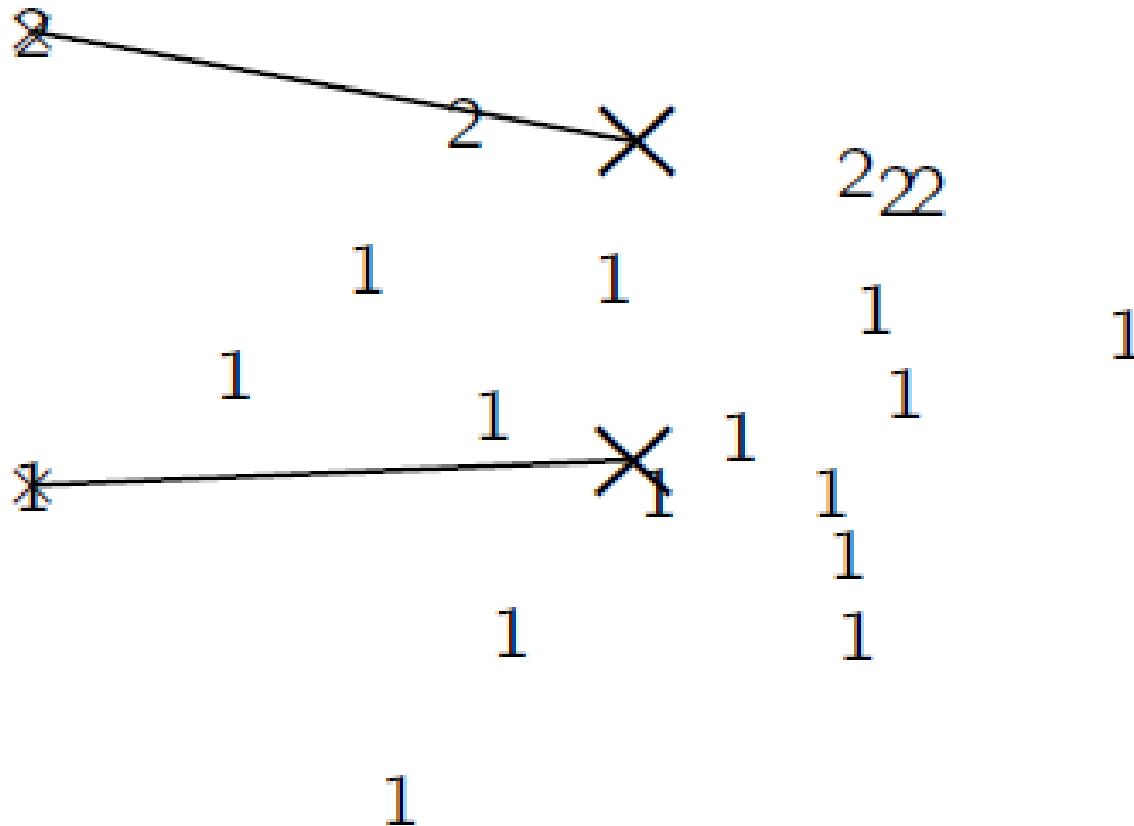
# Example: Random selection of initial centroids



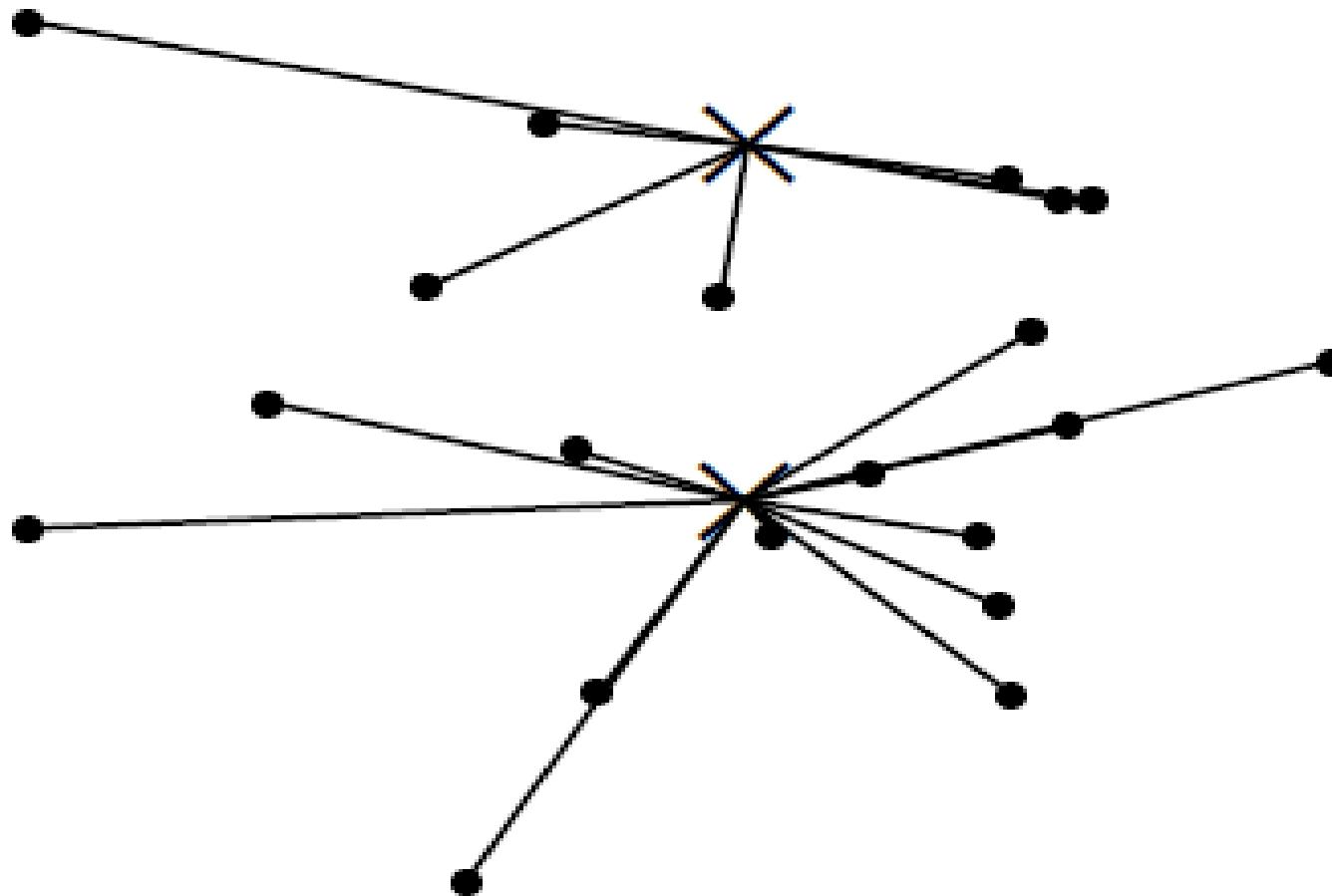
# Example: Assign points to closest center



# Example: Recompute cluster centroids



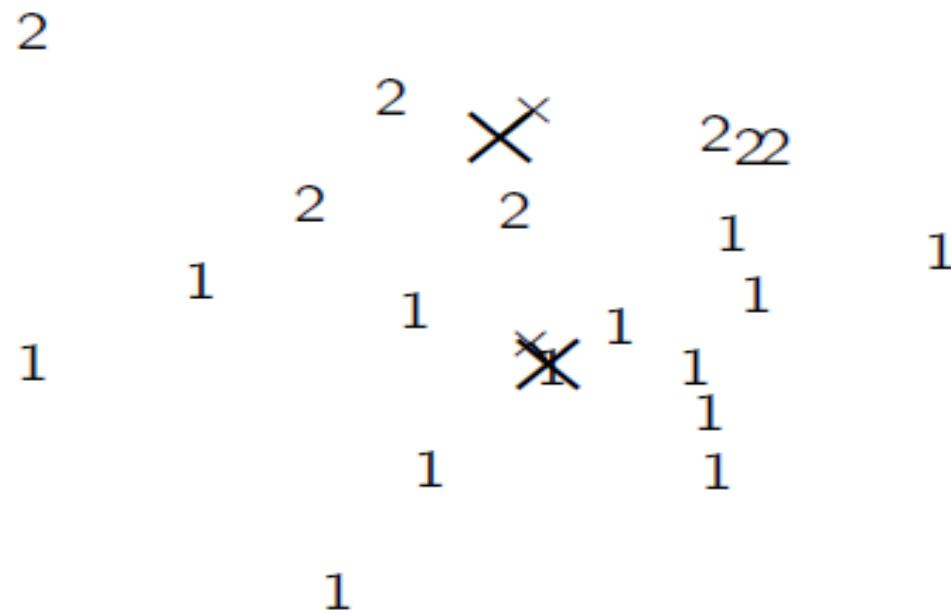
# Example: Assign points to closest centroid



# Example: Assignment

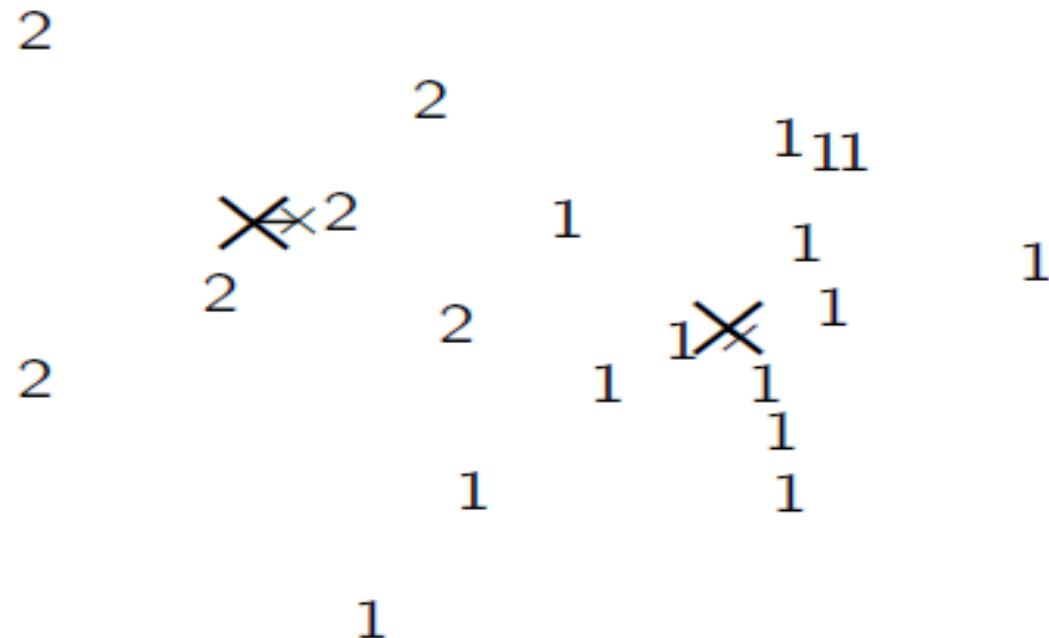
$$\begin{array}{r} 2 \\ \times 22 \\ \hline 11 \\ + 11 \\ \hline 44 \end{array}$$

# Example: Recompute cluster centroids

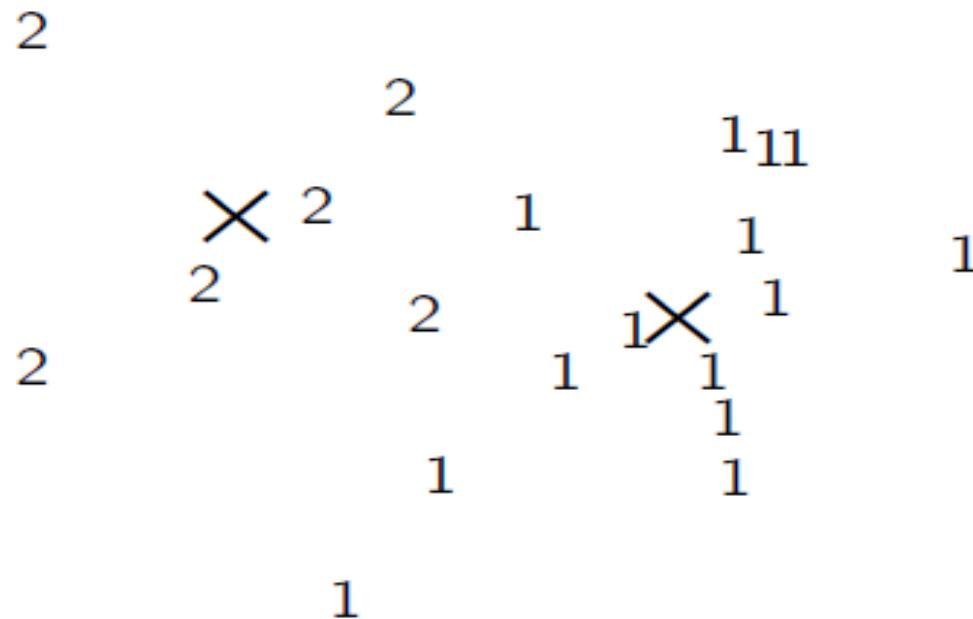




# Example: Recompute cluster centroids



# Example: Centroids and assignments after convergence



# K-means is guaranteed to converge: Proof

- RSS = sum of all squared distances between document vector and closest centroid
- RSS decreases during each reassignment step.
  - because each vector is moved to a closer centroid
- RSS decreases during each recomputation step.
- There is only a finite number of clusterings.
- Thus: We must reach a fixed point.
- Assumption: Ties are broken consistently.
- Finite set & monotonically decreasing → convergence

# Recomputation decreases average distance

$\text{RSS} = \sum_{k=1}^K \text{RSS}_k$  – the residual sum of squares (the “goodness” measure)

$$\begin{aligned}\text{RSS}_k(\vec{v}) &= \sum_{\vec{x} \in \omega_k} \|\vec{v} - \vec{x}\|^2 = \sum_{\vec{x} \in \omega_k} \sum_{m=1}^M (v_m - x_m)^2 \\ \frac{\partial \text{RSS}_k(\vec{v})}{\partial v_m} &= \sum_{\vec{x} \in \omega_k} 2(v_m - x_m) = 0\end{aligned}$$

$$v_m = \frac{1}{|\omega_k|} \sum_{\vec{x} \in \omega_k} x_m$$

The last line is the componentwise definition of the centroid! We minimize  $\text{RSS}_k$  when the old centroid is replaced with the new centroid. RSS, the sum of the  $\text{RSS}_k$ , must then also decrease during recomputation.

# K-means is guaranteed to converge

- But we don't know how long convergence will take!
- If we don't care about a few docs switching back and forth, then convergence is usually fast (< 10-20 iterations).
- However, complete convergence can take many more iterations.

# Optimality of K-means

- Convergence  $\neq$  optimality
- Convergence does not mean that we converge to the optimal clustering!
- This is the great weakness of K-means.
- If we start with a bad set of seeds, the resulting clustering can be horrible.

# Initialization of K-means

- Random seed selection is just one of many ways K-means can be initialized.
- Random seed selection is not very robust: It's easy to get a suboptimal clustering.
- Better ways of computing initial centroids:
  - Select seeds not randomly, but using some heuristic (e.g., filter out outliers or find a set of seeds that has “good coverage” of the document space)
  - Use hierarchical clustering to find good seeds
  - Select  $i$  (e.g.,  $i = 10$ ) different random sets of seeds, do a K-means clustering for each, select the clustering with lowest RSS

# Evaluation

# What is a good clustering?

- Internal criteria
  - Example of an internal criterion: RSS in K-means
- But an internal criterion often does not evaluate the actual utility of a clustering in the application.
- Alternative: External criteria
  - Evaluate with respect to a human-defined classification

# External criteria for clustering quality

- Based on a gold standard data set, e.g., the Reuters collection we also used for the evaluation of classification
- Goal: Clustering should reproduce the classes in the gold standard
- (But we only want to reproduce how documents are divided into groups, not the class labels.)
- First measure for how well we were able to reproduce the classes: purity

# External criterion: Purity

$$\text{purity}(\Omega, C) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j|$$

- $\Omega = \{\omega_1, \omega_2, \dots, \omega_K\}$  is the set of clusters and
- $C = \{c_1, c_2, \dots, c_J\}$  is the set of classes.
- For each cluster  $\omega_k$  : find class  $c_j$  with most members  $n_{kj}$  in  $\omega_k$
- Sum all  $n_{kj}$  and divide by total number of points

# Another external criterion: Rand index

Purity can be increased easily by increasing  $K$  – a measure that does not have this problem: Rand index.

Definition:  $RI = \frac{TP+TN}{TP+FP+FN+TN}$

Based on 2x2 contingency table of all pairs of documents:

	same cluster	different clusters
same class	true positives (TP)	false negatives (FN)
different classes	false positives (FP)	true negatives (TN)

$TP+FN+FP+TN$  is the total number of pairs.

$TP+FN+FP+TN = \binom{N}{2}$  for  $N$  documents.

Example:  $\binom{17}{2} = 136$  in o/◊/x example

Each pair is either positive or negative (the clustering puts the two documents in the same or in different clusters) ...

... and either “true” (correct) or “false” (incorrect): the clustering decision is correct or incorrect.



# Rand Index: Example

As an example, we compute RI for the o/◊/x example. We first compute  $\text{TP} + \text{FP}$ . The three clusters contain 6, 6, and 5 points, respectively, so the total number of “positives” or pairs of documents that are in the same cluster is:

$$\text{TP} + \text{FP} = \binom{6}{2} + \binom{6}{2} + \binom{5}{2} = 40$$

Of these, the x pairs in cluster 1, the o pairs in cluster 2, the ◊ pairs in cluster 3, and the x pair in cluster 3 are true positives:

$$\text{TP} = \binom{5}{2} + \binom{4}{2} + \binom{3}{2} + \binom{2}{2} = 20$$

Thus,  $\text{FP} = 40 - 20 = 20$ . FN and TN are computed similarly.

# Rand measure for the o/\diamond/x example

	same cluster	different clusters	
same class	TP = 20	FN = 24	RI is then
different classes	FP = 20	TN = 72	

$$(20 + 72) / (20 + 20 + 24 + 72) \approx 0.68.$$

# Two other external evaluation measures

- Two other measures
- Normalized mutual information (NMI)
  - How much information does the clustering contain about the classification?
  - Singleton clusters (number of clusters = number of docs) have maximum MI
  - Therefore: normalize by entropy of clusters and classes
- F measure
  - Like Rand, but “precision” and “recall” can be weighted

# Thank You