CalPolyPomona

Retrieval Models I – Boolean Retrieval

Lecture 6

Retrieval Models

- Provide a mathematical framework for defining the search process
  - includes explanation of assumptions
  - basis of many ranking algorithms
- Progress in retrieval models has corresponded with improvements in effectiveness
- Sometimes trial and error
- Different theories about relevance

Relevance

- Complex concept that has been studied for some time
  - Many factors to consider
  - People often disagree when making relevance judgments
- Retrieval models make various assumptions about relevance to simplify problem
  - e.g., *topical* vs. *user* relevance
  - e.g., *binary* vs. *multi-valued* relevance

Retrieval Model Overview

- Original models
  - Boolean retrieval
  - Vector Space model
- Probabilistic Models
  - BM25
  - Language models
- Combining evidence
  - Inference networks
  - Learning to Rank

Boolean Retrieval

Boolean Retrieval

- Two possible outcomes for query processing
  - TRUE and FALSE
  - “exact-match” retrieval
- Query usually specified using Boolean operators
  - AND, OR, NOT
  - proximity operators also used

Searching by Numbers

- Sequence of queries driven by number of retrieved documents
  - e.g. “lincoln” search of news articles
  - president AND lincoln
  - president AND lincoln AND NOT (automobile OR car)
  - president AND lincoln AND biography AND life AND birthplace AND gettysburg AND NOT (automobile OR car)
  - president AND lincoln AND (biography OR life OR birthplace OR gettysburg) AND NOT (automobile OR car)
- Parentheses specify the order of operations
  - A OR (B AND C) does not equal (A OR B) AND C
- If query too broad, user adds AND constraints
- If query too narrow, user adds OR constraints

Examples

president AND lincoln would retrieve:

Ford Motor Company today announced that Darryl Hazel will succeed Brian Kelley as president of Lincoln Mercury.

What if we change it to:

president AND lincoln AND NOT (automobile OR car)

Would it retrieve

Lincoln's body departs Washington in a nine-car funeral train.

Boolean Retrieval

- Advantages
  - Results are predictable, relatively easy to explain
  - Many different features can be incorporated
  - Efficient processing since many documents can be eliminated from search
- Disadvantages
  - Effectiveness depends entirely on user
  - Simple queries usually don't work well (why?)
  - Complex queries are difficult (why?)

Binary Full-text Representation
Document-term matrix

	a	*aardvark*	*abacus*	*abba*	*able*	...	*zoom*
doc_1	1	0	0	0	0	...	1
doc_2	0	0	0	0	1	...	1
::	::	::	::	::	::	...	0
doc_m	0	0	1	1	0	...	0

- 1 = the word appears in the document at least once
- 0 = the word does not appear in the document
- Does not represent word frequency, order, or position information

Processing a Boolean Query

Query: *Jack* AND *Jill*

- doc_1 Jack and Jill went up the hill
- doc_2 To fetch a pail of water.
- doc_3 Jack fell down and broke his crown,
- doc_4 And Jill came tumbling after.
- doc_5 Up Jack got, and home did trot,
- doc_6 As fast as he could caper,
- doc_7 To old Dame Dob, who patched his nob
- doc_8 With vinegar and brown paper.

Processing a Boolean Query

Query: *Jack* AND *Jill*

docid	*text*	*Jack*	*Jill*
doc_1	Jack and Jill went up the hill	1	1
doc_2	To fetch a pail of water.	0	0
doc_3	Jack fell down and broke his crown,	1	0
doc_4	And Jill came tumbling after.	0	1
doc_5	Up Jack got, and home did trot,	1	0
doc_6	As fast as he could caper,	0	0
doc_7	To old Dame Dob, who patched his nob	0	0
doc_8	With vinegar and brown paper.	0	0

Processing a Boolean Query

Query: *Jack* AND *Jill*

	Jack	*Jill*	*Jack* AND *Jill*
doc_1	1	1	
doc_2	0	0	
doc_3	1	0	
doc_4	0	1	
doc_5	1	0	
doc_6	0	0	
doc_7	0	0	
doc_8	0	0	

Processing a Boolean Query

Query: *Jack* AND *Jill*

	Jack	*Jill*	*Jack* AND *Jill*
doc_1	1	1	1
doc_2	0	0	0
doc_3	1	0	0
doc_4	0	1	0
doc_5	1	0	0
doc_6	0	0	0
doc_7	0	0	0
doc_8	0	0	0

Processing a Boolean Query

Query: *Jack* OR *Jill*

	Jack	*Jill*	*Jack* OR *Jill*
doc_1	1	1	1
doc_2	0	0	0
doc_3	1	0	1
doc_4	0	1	1
doc_5	1	0	1
doc_6	0	0	0
doc_7	0	0	0
doc_8	0	0	0

Processing a Boolean Query

Query: *Jack* AND (*up* OR *down*)

	up	*down*	*Jack* AND (*up* OR *down*)
doc_1	1	0	
doc_2	0	0	
doc_3	0	1	
doc_4	0	0	
doc_5	1	0	
doc_6	0	0	
doc_7	0	0	
doc_8	0	0	

Processing a Boolean Query

Query: *Jack* AND (*up* OR *down*)

	up	*down*	*up* OR *down*	*Jack*	*Jack* AND (*up* OR *down*)
doc_1	1	0	1	1	1
doc_2	0	0	0	0	0
doc_3	0	1	1	1	1
doc_4	0	0	0	0	0
doc_5	1	0	1	1	1
doc_6	0	0	0	0	0
doc_7	0	0	0	0	0
doc_8	0	0	0	0	0

Processing a Boolean Query

- Query: *Jack* AND NOT *Jill*

	Jack	*Jill*	*Jack* AND NOT *Jill*
doc_1	1	1	
doc_2	0	0	
doc_3	1	0	
doc_4	0	1	
doc_5	1	0	
doc_6	0	0	
doc_7	0	0	
doc_8	0	0	

Processing a Boolean Query

- Query: *Jack* AND NOT *Jill*

	Jack	*Jill*	NOT *Jill*	*Jack* AND NOT *Jill*
doc_1	1	1		
doc_2	0	0		
doc_3	1	0		
doc_4	0	1		
doc_5	1	0		
doc_6	0	0		
doc_7	0	0		
doc_8	0	0		

Processing a Boolean Query

- Query: *Jack* AND NOT *Jill*

	Jack	*Jill*	NOT *Jill*	*Jack* AND NOT *Jill*
doc_1	1	1	0	0
doc_2	0	0	1	0
doc_3	1	0	1	1
doc_4	0	1	0	0
doc_5	1	0	1	1
doc_6	0	0	1	0
doc_7	0	0	1	0
doc_8	0	0	1	0

Fixed-length inverted list (Document-term matrix)

	a	aardvark	abacus	abba	able	...	zoom
doc_1	1	0	0	0	0	...	1
doc_2	0	0	0	0	1	...	1
::	::	::	::	::	::	...	0
doc_m	0	0	1	1	0	...	0

- Based on Zipf's law, this representation is not efficient
- There are lots of zeros

Variable length inverted list

a	aardvark	abacus	abba	able	...	zoom
df=3421	df=22	df=19	df=2	df=44		df=1
1	33	2	33	66		54
33	56	10	150	134		
45	86	15		176		
::	::	::		::		
1022	1011	231		432		

- Represent only the 1's
- Much more efficient

So far: Unranked Boolean

- Retrieve the set of documents that match the boolean query (an "exact-match" retrieval model)
- Returns results in no particular order (ordered by date? document ID? title?)
- This is problematic with large collections
  - requires complex queries to reduce the result set to a manageable size
- Can we do better?

Ranked Boolean

University	*North*	*Carolina*	*UNC*
df=6	*df=4*	*df=3*	*df=5*
1, 4	1, 4	1, 4	1, 4
10, 1	10, 5	10, 5	10, 1
15, 2	16, 1	16, 1	16, 4
16, 1	68, 1		33, 2
33, 5			56, 10
67, 7			

- *docid = document identifier*
- *tf = term frequency (# of times the term appears in the document)*

Ranked Boolean

- Compute ranking scores based on term frequency
- Score computation:
  - A AND B: take the **minimum** frequency associated with expression A and expression B as the ranking score
  - A OR B: take the **sum** of frequencies associated with expression A and expression B

Ranked Boolean

- Query: (*University* AND *North* AND *Carolina*) OR *UNC*

University	*North*	*Carolina*	*Result_1*
df=6	*df=4*	*df=3*	*count=3*
1, 4	1, 4	1, 4	
10, 1	10, 5	10, 5	
15, 2	16, 1	16, 1	
16, 1	68, 1		
33, 5			
68, 7			

- AND -> ***min***
- OR -> ***sum***

Ranked Boolean

- Query: (*University* AND *North* AND *Carolina*) OR *UNC*

University	*North*	*Carolina*	*Result_1*
df=6	*df=4*	*df=3*	*count=3*
1, 4	1, 4	1, 4	1, 4
10, 1	10, 5	10, 5	10, 1
15, 2	16, 1	16, 1	16, 1
16, 1	68, 1		
33, 5			
68, 7			

- AND -> ***min***
- OR -> ***sum***

Ranked Boolean

- Query: (*University* AND *North* AND *Carolina*) OR *UNC*

Result_1	*UNC*	*Query*
count=3	*df=5*	*count=*
1, 4	1, 4	
10, 1	10, 1	
16, 1	16, 4	
	33, 2	
	56, 10	

- AND -> ***min***
- OR -> ***sum***

Ranked Boolean

- Query: (*University* AND *North* AND *Carolina*) OR *UNC*

Result_1	*UNC*	*Query*
count=3	*df=5*	*count=5*
1, 4	1, 4	1, 8
10, 1	10, 1	10, 2
16, 1	16, 4	16, 5
	33, 2	33, 2
	56, 10	56, 10

- AND -> ***min***
- OR -> ***sum***

Ranked Boolean

- Advantages:
  - same as unranked boolean: efficient, predictable, easy to understand, works well when the user knows what to look for
  - the user may be able to find relevant documents quicker and may not need to examine the entire result set
- Disadvantages:
  - same as unranked boolean: only works well when the user knows what to look for