COEN 169

Retrieval Models

Yi Fang

Department of Computer Engineering

Santa Clara University

The Search Task

- Given a query and a corpus, find relevant items

  query: a textual description of the user's information need

  corpus: a repository of documents

  relevance: satisfaction of the user's information need

What is a Retrieval Model?

A formal method that predicts the degree of relevance of a document to a query

Relevance

- Many factors affect whether a document satisfies a particular user's information need
- Topicality, freshness, diversity, novelty, reading level, authority, etc.
- Topical relevance: the document is on the same topic as the query
- For now, we will only try to predict topical relevance

Overview of Retrieval Models

- Boolean

  - unranked Boolean retrieval

  - ranked Boolean retrieval

- Vector space models — SMART, Lucene
- Probabilistic models
  - Statistical language models — Lemur
  - Two Poisson model — Okapi
- Citation/Link analysis models
  - PageRank — Google
  - Hub & authorities

Boolean Retrieval

Assumption: the user can represent their information need using boolean constraints: AND, OR, and AND NOT

- lincoln
- president AND lincoln
- president AND (lincoln OR abraham)
- president AND (lincoln OR abraham) AND NOT car
- president AND (lincoln OR abraham) AND NOT (car OR automobile)

- Parentheses can specify the order of operations
- A OR (B AND C) does not equal (A OR B) AND C

Boolean Retrieval

- If the query is too specific, precision may be high, but recall will probably be low
- If the query is too broad, recall may be high, but precision will probably be low
- Extreme cases:
  - a query that retrieves a single relevant document will have perfect precision, but low recall (unless only that one document is relevant)
  - a query that retrieves the entire collection will have perfect recall, but low precision (unless the whole collection is relevant)

Boolean Retrieval

- Easy for the system (no ambiguity in the query)

✓ the burden is on the user to formulate the right query

- The user gets transparency and control

✓ lots of results → the query is too broad

✓ no results → the query is too narrow

- Common strategy for finding the right balance is:

✓ if the query is too broad, add AND or AND NOT constraints

✓ if the query is too narrow, add OR constraints

The 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	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*	*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*	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

The Binary Full-text Representation

	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!

Sparse Representation of an 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		

- Variable-length inverted lists
  - represent only the 1's
  - each document has a unique identifier (docid)
  - df = number of documents in which the term appears at least once
- Why do we store the *df* in the index?

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?)
- 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	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*

University	*North*	*Carolina*	*UNC*	*Query*
df=6	*df=4*	*df=3*	*df=5*	*count=5*
1, 4	1, 4	1, 4	1, 4	1, 8
10, 1	10, 5	10, 5	10, 1	10, 2
15, 2	16, 1	16, 1	16, 4	16, 5
16, 1	68, 1		33, 2	33, 2
33, 5			56, 10	56, 10
68, 7				

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: works well when the user knows what to look for

Best-Match Retrieval Models

- So far, we've discussed 'exact-match' models
- Now, we start discussing 'best-match' models
- Best-match models predict the degree to which a document is relevant to a query
- Ideally, this is would be expressed as RELEVANT(q,d)
- In practice, it is expressed as SIMILAR(q,d)
- How might you compute the similarity between q and d?