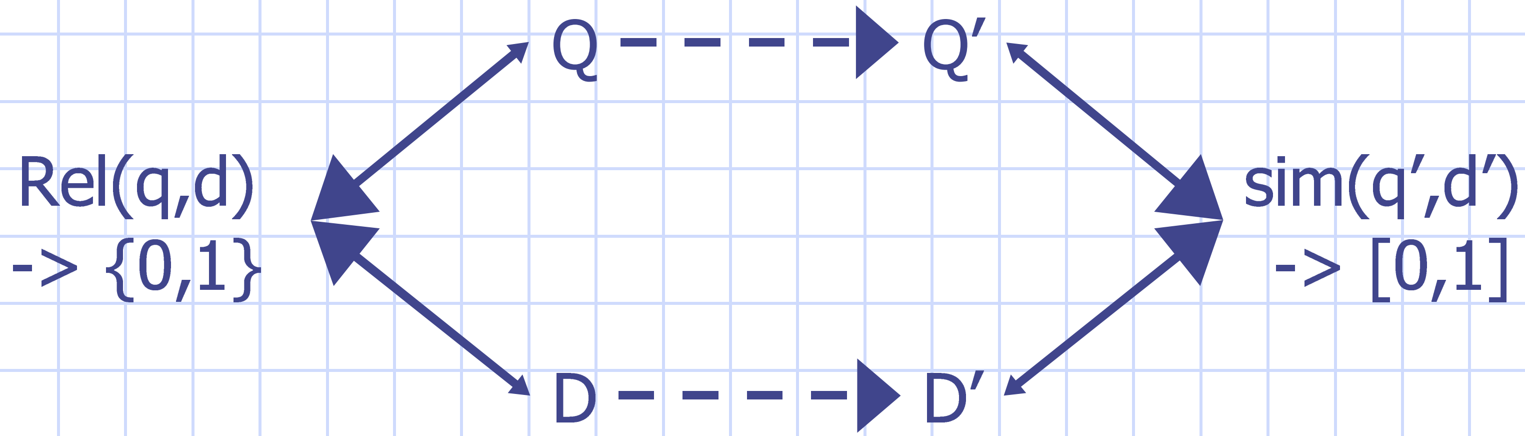
IR Models: The Probabilistic Model

Lecture 8

Probability of Relevance?



- IR is an uncertain process
  - Information need to query
  - Documents to index terms
  - Query terms and index terms mismatch
- Leads to several statistical approaches
  - probability theory, fuzzy logic, theory of evidence...

Probabilistic Retrieval

- Given a query *q*, there exists a subset of the documents R which are relevant to *q*
  - But membership of R is uncertain
- A Probabilistic retrieval model
  - ranks documents in decreasing order of *probability of relevance* to the information need: $P(R \mid q,d_i)$

Difficulties

1. Evidence is based on a lossy representation
    - Evaluate probability of relevance based on occurrence of terms in query and documents
    - Start with an initial estimate, and refine through feedback
2. Computing the probabilities exactly according to the model is intractable
    - Make some simplifying assumptions

Probabilistic Model definitions

- $d_j = (t_{1,j}, t_{2,j}, \dots t_{t,j}), t_{i,j} \in \{0,1\}$
  - terms occurrences are boolean (not counts)
  - query *q* is represented similarly
- R is the set of relevant documents, ~R is the set of irrelevant documents
- $P(R \mid d_j)$ is probability that d_j is relevant, $P(\sim R \mid d_j)$ irrelevant

Retrieval Status Value

- "Similarity" function
  - ratio of prob. of relevance to prob. of non-relevance
- Transform P(R | d_j) using Bayes' Rule
  - Compute rsv() in terms of document probabilities
- P(R) and P(~R) are constant for each document

$$rsv(d_j, q) = \frac{P(R \mid d_j)}{P(\bar{R} \mid d_j)}$$

$$P(x \mid y) = \frac{P(x) \times P(y \mid x)}{P(y)}$$

$$rsv(d_j, q) = \frac{P(d_j \mid R) \times P(R)}{P(d_j \mid \bar{R}) \times P(\bar{R})}$$

$$rsv(d_j, q) \approx \frac{P(d_j \mid R)}{P(d_j \mid \bar{R})}$$

Retrieval Status Value (2)

- d is a vector of binary term occurrences
- We assume that terms occur independently of each other

$$P(d_j|R) = \prod_{t=1} P(t_i|R)$$

$$\mathrm{rsv}(d_j, q) = \sum_{t=1} \log \frac{P(t_i|R)}{P(t_i|\overline{R})}$$

$$\begin{aligned}
\mathrm{rsv}(d_j, q) &= \sum_t \left(\log \frac{P(t_i|R)}{P(t_i|\overline{R})} - \log \frac{P(\overline{t_i}|R)}{P(\overline{t_i}|\overline{R})} \right) \\
&= \sum_t \log \frac{P(t_i|R)P(\overline{t_i}|\overline{R})}{P(t_i|\overline{R})P(\overline{t_i}|R)}
\end{aligned}$$

Computing term probabilities

- Initially, there are no retrieved documents
  - R is completely unknown
  - Assume $P(t_i|R)$ is constant (usually 0.5)
  - Assume $P(t_i|{\sim}R)$ approximated by distribution of t_i across collection – IDF

$$P(t|\overline{R}) = \log \frac{N - n + 0.5}{n + 0.5}$$

- This can be used to compute an initial rank using IDF as the basic term weight

Probabilistic Model Example

d	Document vectors <$tf_{d,t}$>									
	col	day	eat	hot	lot	nin	old	pea	por	pot
1	1.0			1.0				1.0	1.0	
2								1.0	1.0	1.0
3		1.0				1.0	1.0			
4	1.0			1.0						1.0
5								1.0	1.0	
6			1.0		1.0					
w_t	0.26	0.56	0.56	0.26	0.56	0.56	0.56	0.0	0.0	0.26

- q1 = *eat*
- q2 = *porridge*
- q3 = *hot porridge*
- q4 = *eat nine day old porridge*

$w_t = \log(N-n+0.5/n+0.5)$

Improving the ranking

- Now, suppose
  - we have shown the initial ranking to the user
  - the user has labeled some of the documents as relevant ("**relevance feedback"**)
- We now have
  - N documents in coll, R are known relevant
  - n_i documents containing t_i, r_i are relevant

Improving term estimates

for term i …	Rel	Non-rel	Total
docs containing term	r	n-r	n
docs NOT containing term	R-r	N-R-n+r	N-n
Total	R	N-R	N

$$p_i = P(t_i \mid R) = \frac{r}{R}$$

$$q_i = P(t_i \mid \overline{R}) = \frac{n-r}{N-R}$$

$$w_i = \log \frac{p_i(1-q_i)}{q_i(1-p_i)}$$

$$= \log \frac{r(N-R-n+r)}{(n-r)(R-r)}$$

Final term weight

- Add 0.5 to each term, to keep the weight from being infinite when R, r are small:

$$w_i = \log \frac{(r+0.5)(N-R-n+r+0.5)}{(n-r+0.5)(R-r+0.5)}$$

- Can continue to refine the ranking as the user gives more feedback.

Relevance-weighted Example

d	col	day	eat	hot	lot	nin	old	pea	por	pot
	Document vectors <$tf_{d,t}$>									
1	1.0			1.0				1.0	1.0	
2								1.0	1.0	1.0
3		1.0				1.0	1.0			
4	1.0			1.0						1.0
5								1.0	1.0	
6			1.0		1.0					
w_t	-0.33	0.0	0.0	-0.33	0.0	0.0	0.0	0.62	0.62	0.95

- q3 = *hot porridge*, document 2 is relevant

Summary

- Probabilistic model uses probability theory to model the uncertainty in the retrieval process
- Assumptions are made explicit
- Term weight without relevance information is inverse document frequency (IDF)
- Relevance feedback can improve the ranking by giving better term probability estimates
- No use of within-document term frequencies or document lengths

Building on the Probabilistic Model: Okapi weighting

- Okapi system
  - developed at City University London
  - based on probabilistic model
- Cost of not using tf and document length
  - doesn't perform as well as VSM
  - hurts performance on long documents
- Okapi solution
  - model within-document term frequencies as a mixture of two Poisson distributions
  - one for relevant documents and one for irrelevant ones

Okapi best-match weights

$$BM0 = \sum_{t \in Q} \log \frac{(r+0.5)(N-R-n+r+0.5)}{(n-r+0.5)(R-r+0.5)} \quad \text{(this is } \mathrm{w}^{(1)}\text{)}$$

$$BM1 = \sum_{t \in Q} \mathrm{w}^{(1)} \times \frac{t_{i,q}}{k_3 \times t_{i,q}}$$

$$BM25 = \sum_{t \in Q} \mathrm{w}^{(1)} \times \frac{(k_1+1)t_{i,j}}{K+t_{i,j}} \times \frac{(k_3+1)t_{i,q}}{k_3 \times t_{i,q}} + k_2 \cdot |Q| \cdot \frac{avdl - dl}{avdl + dl}$$

$$(K = k_1((1-b) + (b \cdot dl)/avdl))$$

in TREC-8: $k_1 = [1,2]$, $k_2 = 0$, $k_3 = 7$, $b = [0.6, 0.75]$

Okapi weighting

- Okapi weights use
  - a "tf" component similar to VSM
  - separate document and query length normalizations
  - several tuning constants which depend on the collection
- In experiments, Okapi weights give the best performance

Okapi-weights Example

d	col	day	eat	hot	lot	nin	old	pea	por	pot	dl
	Document vectors <$tf_{d,t}$>										
1	1.0			1.0				2.0	2.0		6
2								1.0	1.0	1.0	3
3		1.0				1.0	1.0				3
4	1.0			1.0						2.0	4
5								2.0	2.0		4
6			1.0		1.0						2
$w^{(1)}$	0.26	0.56	0.56	0.26	0.56	0.56	0.56	0.0	0.0	0.26	

- q1 = *eat*
- q2 = *porridge*
- q3 = *hot porridge*
- q4 = *eat nine day old porridge*

$k_1 = 1.2$ $k_3 = 7$

$k_2 = 0$ $b = 0.75$

avdl = 3.66

Okapi-weights +RF Example

d	Document vectors <$tf_{d,t}$>										dl
	col	day	eat	hot	lot	nin	old	pea	por	pot	
1	1.0			1.0				2.0	2.0		6
2								1.0	1.0	1.0	3
3		1.0				1.0	1.0				3
4	1.0			1.0						2.0	4
5								2.0	2.0		4
6			1.0		1.0						2
$w^{(1)}$	-0.33	0.0	0.0	-0.33	0.0	0.0	0.0	0.62	0.62	0.95	

- q3 = *hot porridge*, doc 2 is relevant

$k_1 = 1.2$

$k_2 = 0$

$k_3 = 7$

$b = 0.75$

$avdl = 3.66$

Ranking algorithm

1. $A = \{\}$ (set of accumulators for documents)
2. For each query term t
   - Get term, f_t, and address of I_t from lexicon
   - set $w^{(1)}$ and qtf variables
   - Read inverted list I_t
   - For each $\langle d, f_{d,t} \rangle$ in I_t
     1. If $A_d \notin A$, initialize A_d to 0 and add it to A
     2. $A_d = A_d + (w^{(1)} \times tf \times qtf) + qnorm$
3. For each A_d in A
   1. $A_d = A_d / W_d$
4. Fetch and return top *r* documents to user

Managing Accumulators

- How to store accumulators?
  - static array, 1 per document
  - grow as needed with a hash table
- How many accumulators?
  - can impose a fixed limit
  - quit processing I_t's after limit reached
  - continue processing, but add no new A_d's

Managing Accumulators (2)

- To make this work, we want to process the query terms in order of decreasing idf_t
- Also want to process I_t in decreasing $tf_{d,t}$ order
  - sort I_t when we read it in
  - or, store inverted lists in $f_{d,t}$-sorted order

  <5; (1,2) (2,2) (3,5) (4,1) (5,2)> *<f_t; (d, $f_{d,t}$)…>*

  <5; (3,5) (1,2) (2,2) (5,2) (4,1)> *sorted by $f_{d,t}$*

  <5; (5, 1:3) (2, 3:1,2,5) (1, 1:4)> *<f_t; ($f_{d,t}$, c:d,…)…>*
  - This can actually compress better, but makes Boolean queries harder to process

Getting the top documents

- Naïve: sort the accumulator set at end
- Or, use a heap and pull top *r* documents
  - much faster if r << N
- Or better yet, as accumulators are processed to add the length norm (W_d):
  - make first r accumulators into a min-heap
  - for each next accumulator
    - if A_d < heap-min, just drop it
    - if A_d > heap-min, drop the heap-min, and put A_d in