Search Engines

Information Retrieval in Practice

Retrieval Models

- Provide a mathematical framework for defining the search process
  - includes explanation of assumptions
  - basis of many ranking algorithms
  - can be implicit
- Progress in retrieval models has corresponded with improvements in effectiveness
- 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

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

Boolean Retrieval

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

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
  - Complex queries are difficult

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)

Vector Space Model

- Documents and query represented by a vector of term weights
- Collection represented by a matrix of term weights

$$D_i = (d_{i1}, d_{i2}, \ldots, d_{it}) \qquad Q = (q_1, q_2, \ldots, q_t)$$

$$\begin{array}{lcccc} & Term_1 & Term_2 & \ldots & Term_t \\ Doc_1 & d_{11} & d_{12} & \ldots & d_{1t} \\ Doc_2 & d_{21} & d_{22} & \ldots & d_{2t} \\ \vdots & \vdots & & & \\ Doc_n & d_{n1} & d_{n2} & \ldots & d_{nt} \end{array}$$

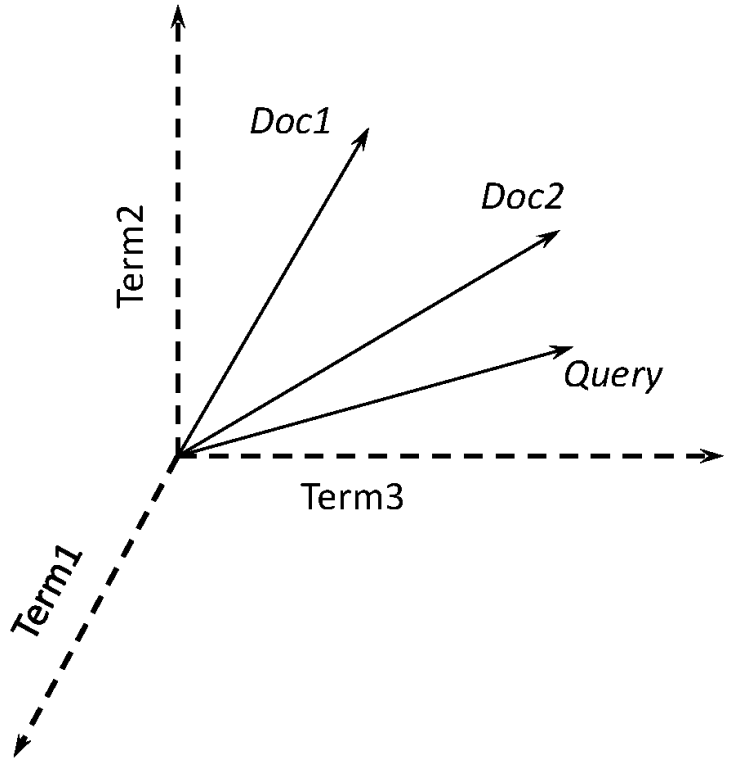
Vector Space Model

D_1 Tropical Freshwater Aquarium Fish.
D_2 Tropical Fish, Aquarium Care, Tank Setup.
D_3 Keeping Tropical Fish and Goldfish in Aquariums, and Fish Bowls.
D_4 The Tropical Tank Homepage - Tropical Fish and Aquariums.

Terms	**Documents**			
	D_1	**D_2**	**D_3**	**D_4**
aquarium	1	1	1	1
bowl	0	0	1	0
care	0	1	0	0
fish	1	1	2	1
freshwater	1	0	0	0
goldfish	0	0	1	0
homepage	0	0	0	1
keep	0	0	1	0
setup	0	1	0	0
tank	0	1	0	1
tropical	1	1	1	2

Vector Space Model

- 3-d pictures useful, but can be misleading for high-dimensional space



Vector Space Model

- Documents ranked by distance between points representing query and documents
  - *Similarity* measure more common than a distance or *dissimilarity* measure
  - e.g. Cosine correlation

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

Similarity Calculation

- Consider two documents D_1, D_2 and a query Q
  - D_1 = (0.5, 0.8, 0.3), D_2 = (0.9, 0.4, 0.2), Q = (1.5, 1.0, 0)

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

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

Term Weights

- *tf.idf* weight
  - Term frequency weight measures importance in document: $tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} f_{ij}}$
  - Inverse document frequency measures importance in collection: $idf_k = \log \frac{N}{n_k}$
  - Some heuristic modifications

$$d_{ik} = \frac{(\log(f_{ik})+1)\cdot\log(N/n_k)}{\sqrt{\sum_{k=1}^{t}[(\log(f_{ik})+1.0)\cdot\log(N/n_k)]^2}}$$

Relevance Feedback

- Rocchio algorithm
- *Optimal query*
  - Maximizes the difference between the average vector representing the relevant documents and the average vector representing the non-relevant documents
- Modifies query according to

$$q'_j = \alpha.q_j + \beta.\frac{1}{|Rel|}\sum_{D_i \in Rel} d_{ij} - \gamma.\frac{1}{|Nonrel|}\sum_{D_i \in Nonrel} d_{ij}$$

  - α, β, and γ are parameters
    - Typical values 8, 16, 4

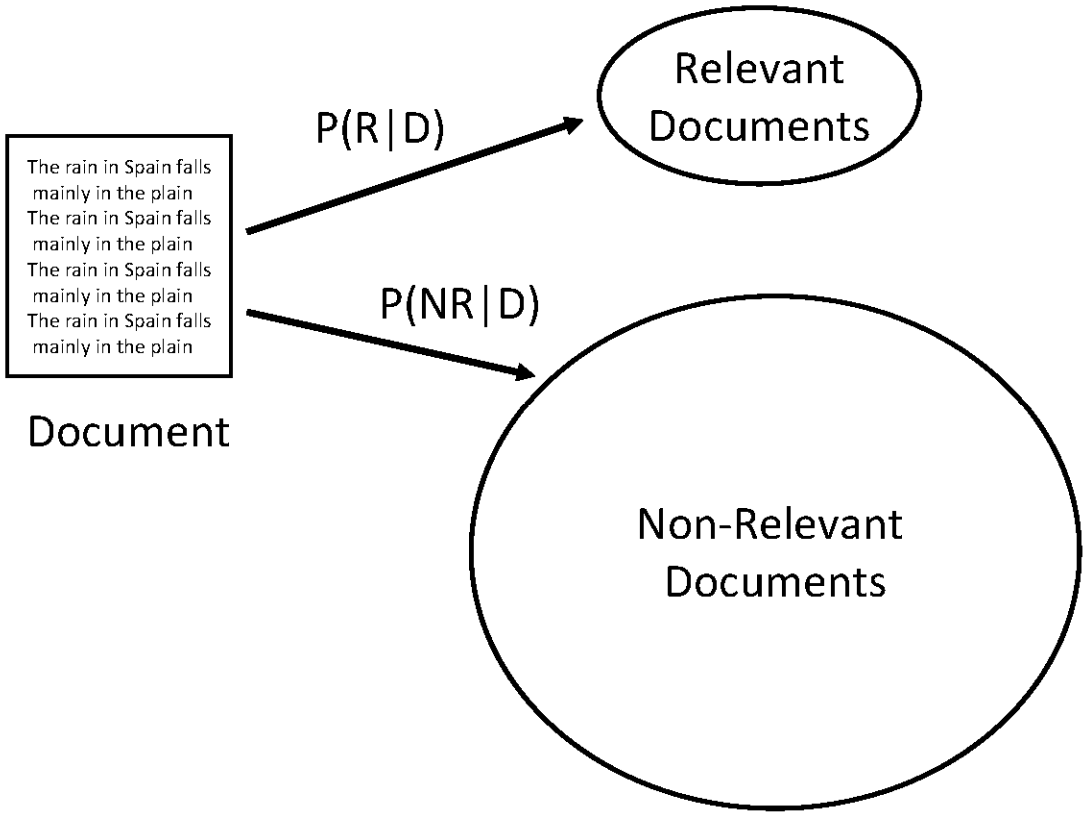
Vector Space Model

- Advantages
  - Simple computational framework for ranking
  - Any similarity measure or term weighting scheme could be used
- Disadvantages
  - Assumption of term independence
  - No *predictions* about techniques for effective ranking

Probability Ranking Principle

- Robertson (1977)
  - “If a reference retrieval system’s response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance to the user who submitted the request,
  - where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose,
  - the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data.”

IR as Classification



Bayes Classifier

- Bayes Decision Rule
  - A document *D* is relevant if $P(R|D) > P(NR|D)$
- Estimating probabilities
  - use Bayes Rule

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

  - classify a document as relevant if

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

    - lhs is *likelihood ratio*

Estimating P(D|R)

- Assume independence

$$P(D|R) = \prod_{i=1}^{t} P(d_i|R)$$

- *Binary independence model*
  - document represented by a vector of binary features indicating term occurrence (or non-occurrence)
  - p_i is probability that term i occurs (i.e., has value 1) in relevant document, s_i is probability of occurrence in non-relevant document

Binary Independence Model

$$\frac{P(D|R)}{P(D|NR)} = \prod_{i:d_i=1} \frac{p_i}{s_i} \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i}$$

$$= \prod_{i:d_i=1} \frac{p_i}{s_i} \cdot \left(\prod_{i:d_i=1} \frac{1-s_i}{1-p_i} \cdot \prod_{i:d_i=1} \frac{1-p_i}{1-s_i}\right) \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i}$$

$$= \prod_{i:d_i=1} \frac{p_i(1-s_i)}{s_i(1-p_i)} \cdot \prod_{i} \frac{1-p_i}{1-s_i}$$

Binary Independence Model

- Scoring function is

$$\sum_{i:d_i=1} \log \frac{p_i(1-s_i)}{s_i(1-p_i)}$$

- Query provides information about relevant documents
- If we assume p_i constant, s_i approximated by entire collection, get *idf*-like weight

$$\log \frac{0.5(1-\frac{n_i}{N})}{\frac{n_i}{N}(1-0.5)} = \log \frac{N-n_i}{n_i}$$

Contingency Table

	Relevant	Non-relevant	Total
$d_i = 1$	r_i	$n_i - r_i$	n_i
$d_i = 0$	$R - r_i$	$N - n_i - R + r_i$	$N - r_i$
Total	R	$N - R$	N

$$p_i = (r_i + 0.5)/(R + 1)$$

$$s_i = (n_i - r_i + 0.5)/(N - R + 1)$$

Gives scoring function:

$$\sum_{i:d_i=q_i=1} \log \frac{(r_i+0.5)/(R-r_i+0.5)}{(n_i-r_i+0.5)/(N-n_i-R+r_i+0.5)}$$

BM25

- Popular and effective ranking algorithm based on binary independence model
  - adds document and query term weights

$$\sum_{i \in Q} \log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} \cdot \frac{(k_1 + 1) f_i}{K + f_i} \cdot \frac{(k_2 + 1) q f_i}{k_2 + q f_i}$$

  - k_1, k_2 and K are parameters whose values are set empirically
  - $K = k_1((1 - b) + b \cdot \frac{dl}{avdl})$ *dl* is doc length
  - Typical TREC value for k_1 is 1.2, k_2 varies from 0 to 1000, b = 0.75

BM25 Example

- Query with two terms, “president lincoln”, (*qf = 1)*
- No relevance information (*r and R are* zero)
- N = 500,000 documents
- *“president”* occurs in 40,000 documents (n_1 = 40, 000)
- *“lincoln”* occurs in 300 documents (n_2 = 300)
- “president” occurs 15 times in doc (f_1 = 15)
- *“lincoln”* occurs 25 times (f_2 = 25)
- document length is 90% of the average length (*dl/avdl* = .9)
- k_1 = 1.2, b = 0.75, and k_2 = 100
- $K = 1.2 \cdot (0.25 + 0.75 \cdot 0.9) = 1.11$

BM25 Example

$$
\begin{aligned}
BM25(Q,D) \quad = \quad & \log \frac{(0+0.5)/(0-0+0.5)}{(40000-0+0.5)/(500000-40000-0+0+0.5)} \\
& \times \frac{(1.2+1)15}{1.11+15} \times \frac{(100+1)1}{100+1} \\
& + \log \frac{(0+0.5)/(0-0+0.5)}{(300-0+0.5)/(500000-300-0+0+0.5)} \\
& \times \frac{(1.2+1)25}{1.11+25} \times \frac{(100+1)1}{100+1}
\end{aligned}
$$

$$
\begin{aligned}
= \quad & \log 460000.5/40000.5 \cdot 33/16.11 \cdot 101/101 \\
& + \log 499700.5/300.5 \cdot 55/26.11 \cdot 101/101 \\
= \quad & 2.44 \cdot 2.05 \cdot 1 + 7.42 \cdot 2.11 \cdot 1 \\
= \quad & 5.00 + 15.66 = 20.66
\end{aligned}
$$

BM25 Example

- Effect of term frequencies

Frequency of "president"	Frequency of "lincoln"	BM25 score
15	25	20.66
15	1	12.74
15	0	5.00
1	25	18.2
0	25	15.66

Language Model

- *Unigram language model*
  - probability distribution over the words in a language
  - generation of text consists of pulling words out of a “bucket” according to the probability distribution and replacing them
- N-gram language model
  - some applications use bigram and trigram language models where probabilities depend on previous words

Language Model

- A *topic* in a document or query can be represented as a language model
  - i.e., words that tend to occur often when discussing a topic will have high probabilities in the corresponding language model
- *Multinomial* distribution over words
  - text is modeled as a finite sequence of words, where there are *t* possible words at each point in the sequence
  - commonly used, but not only possibility
  - doesn't model *burstiness*

LMs for Retrieval

- 3 possibilities:
  - probability of generating the query text from a document language model
  - probability of generating the document text from a query language model
  - comparing the language models representing the query and document topics
- Models of topical relevance

Query-Likelihood Model

- Rank documents by the probability that the query could be generated by the document model (i.e. same topic)
- Given query, start with P(D|Q)
- Using Bayes' Rule

$$p(D|Q) \overset{rank}{=} P(Q|D)P(D)$$

- Assuming prior is uniform, unigram model

$$P(Q|D) = \prod_{i=1}^{n} P(q_i|D)$$

Estimating Probabilities

- Obvious estimate for unigram probabilities is

$$P(q_i|D) = \frac{f_{q_i,D}}{|D|}$$

- *Maximum likelihood estimate*
  - makes the observed value of $f_{q_i,D}$ most likely
- If query words are missing from document, score will be zero
  - Missing 1 out of 4 query words same as missing 3 out of 4

Smoothing

- Document texts are a *sample* from the language model
  - Missing words should not have zero probability of occurring
- *Smoothing* is a technique for estimating probabilities for missing (or unseen) words
  - lower (or *discount*) the probability estimates for words that are seen in the document text
  - assign that "left-over" probability to the estimates for the words that are not seen in the text

Estimating Probabilities

- Estimate for unseen words is $\alpha_D P(q_i|C)$
  - $P(q_i|C)$ is the probability for query word *i* in the *collection* language model for collection *C* (background probability)
  - α_D is a parameter
- Estimate for words that occur is

$$(1 - \alpha_D)\, P(q_i|D) + \alpha_D\, P(q_i|C)$$

- Different forms of estimation come from different α_D

Jelinek-Mercer Smoothing

- α_D is a constant, λ
- Gives estimate of

$$p(q_i|D) = (1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|}$$

- Ranking score

$$P(Q|D) = \prod_{i=1}^{n}((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})$$

- Use logs for convenience
  - accuracy problems multiplying small numbers

$$\log P(Q|D) = \sum_{i=1}^{n}\log((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})$$

Where is *tf.idf* Weight?

$$\log P(Q|D) = \sum_{i=1}^{n} \log((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})$$

$$= \sum_{i:f_{q_i,D}>0} \log((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|}) + \sum_{i:f_{q_i,D}=0} \log(\lambda\frac{c_{q_i}}{|C|})$$

$$= \sum_{i:f_{q_i,D}>0} \log \frac{((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})}{\lambda\frac{c_{q_i}}{|C|}} + \sum_{i=1}^{n} \log(\lambda\frac{c_{q_i}}{|C|})$$

$$\overset{rank}{=} \sum_{i:f_{q_i,D}>0} \log\left(\frac{((1-\lambda)\frac{f_{q_i,D}}{|D|}}{\lambda\frac{c_{q_i}}{|C|}} + 1\right)$$

- proportional to the term frequency, inversely proportional to the collection frequency

Dirichlet Smoothing

- α_D depends on document length

$$\alpha_D = \frac{\mu}{|D|+\mu}$$

- Gives probability estimation of

$$p(q_i|D) = \frac{f_{q_i,D}+\mu\frac{c_{q_i}}{|C|}}{|D|+\mu}$$

- and document score

$$\log P(Q|D) = \sum_{i=1}^{n} \log \frac{f_{q_i,D}+\mu\frac{c_{q_i}}{|C|}}{|D|+\mu}$$

Query Likelihood Example

- For the term “president”
  - $f_{qi,D} = 15$, $c_{qi} = 160{,}000$
- For the term “lincoln”
  - $f_{qi,D} = 25$, $c_{qi} = 2{,}400$
- number of word occurrences in the document |d| is assumed to be 1,800
- number of word occurrences in the collection is 10^9
  - 500,000 documents times an average of 2,000 words
- $\mu = 2{,}000$

Query Likelihood Example

$$
\begin{aligned}
QL(Q,D) &= \log \frac{15 + 2000 \times (1.6 \times 10^5 / 10^9)}{1800 + 2000} \\
&\quad + \log \frac{25 + 2000 \times (2400/10^9)}{1800 + 2000} \\
&= \log(15.32/3800) + \log(25.005/3800) \\
&= -5.51 + -5.02 = -10.53
\end{aligned}
$$

- Negative number because summing logs of small numbers

Query Likelihood Example

Frequency of "president"	Frequency of "lincoln"	QL score
15	25	-10.53
15	1	-13.75
15	0	-19.05
1	25	-12.99
0	25	-14.40

Relevance Models

- *Relevance model* – language model representing information need
  - query and relevant documents are samples from this model
- *P(D|R)* - probability of generating the text in a document given a relevance model
  - *document likelihood* model
  - less effective than query likelihood due to difficulties comparing across documents of different lengths

Pseudo-Relevance Feedback

- Estimate relevance model from query and top-ranked documents
- Rank documents by similarity of document model to relevance model
- *Kullback-Leibler divergence* (KL-divergence) is a well-known measure of the difference between two probability distributions

KL-Divergence

- Given the *true* probability distribution *P* and another distribution *Q* that is an *approximation* to *P*,

$$KL(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

  - Use negative KL-divergence for ranking, and assume relevance model *R* is the true distribution (not symmetric),

$$\sum_{w \in V} P(w|R) \log P(w|D) - \sum_{w \in V} P(w|R) \log P(w|R)$$

KL-Divergence

- Given a simple maximum likelihood estimate for *P(w|R),* based on the frequency in the query text, ranking score is

$$\sum_{w \in V} \frac{f_{w,Q}}{|Q|} \log P(w|D)$$

  - rank-equivalent to query likelihood score
- Query likelihood model is a special case of retrieval based on relevance model

Estimating the Relevance Model

- Probability of pulling a word *w* out of the "bucket" representing the relevance model depends on the *n* query words we have just pulled out

$$P(w|R) \approx P(w|q_1 \ldots q_n)$$

- By definition

$$P(w|R) \approx \frac{P(w,q_1 \ldots q_n)}{P(q_1 \ldots q_n)}$$

Estimating the Relevance Model

- Joint probability is

$$P(w, q_1 \ldots q_n) = \sum_{D \in \mathcal{C}} p(D) P(w, q_1 \ldots q_n | D)$$

- Assume

$$P(w, q_1 \ldots q_n | D) = P(w|D) \prod_{i=1}^{n} P(q_i|D)$$

- Gives

$$P(w, q_1 \ldots q_n) = \sum_{D \in \mathcal{C}} P(D) P(w|D) \prod_{i=1}^{n} P(q_i|D)$$

Estimating the Relevance Model

- *P(D)* usually assumed to be uniform
- *P(w, q1 . . . qn)* is simply a weighted average of the language model probabilities for *w* in a set of documents, where the weights are the query likelihood scores for those documents
- Formal model for pseudo-relevance feedback
  - query expansion technique

Pseudo-Feedback Algorithm

1. Rank documents using the query likelihood score for query Q.
2. Select some number of the top-ranked documents to be the set $\mathcal{C}$.
3. Calculate the relevance model probabilities $P(w|R)$. $P(q_1 \dots q_n)$ is used as a normalizing constant and is calculated as

$$P(q_1 \dots q_n) = \sum_{w \in V} P(w, q_1 \dots q_n)$$

4. Rank documents again using the KL-divergence score

$$\sum_{w} P(w|R) \log P(w|D)$$

Example from Top 10 Docs

president lincoln	*abraham lincoln*	*fishing*	*tropical fish*
lincoln	lincoln	fish	fish
president	america	farm	tropic
room	president	salmon	japan
bedroom	faith	new	aquarium
house	guest	wild	water
white	abraham	water	species
america	new	caught	aquatic
guest	room	catch	fair
serve	christian	tag	china
bed	history	time	coral
washington	public	eat	source
old	bedroom	raise	tank
office	war	city	reef
war	politics	people	animal
long	old	fishermen	tarpon
abraham	national	boat	fishery

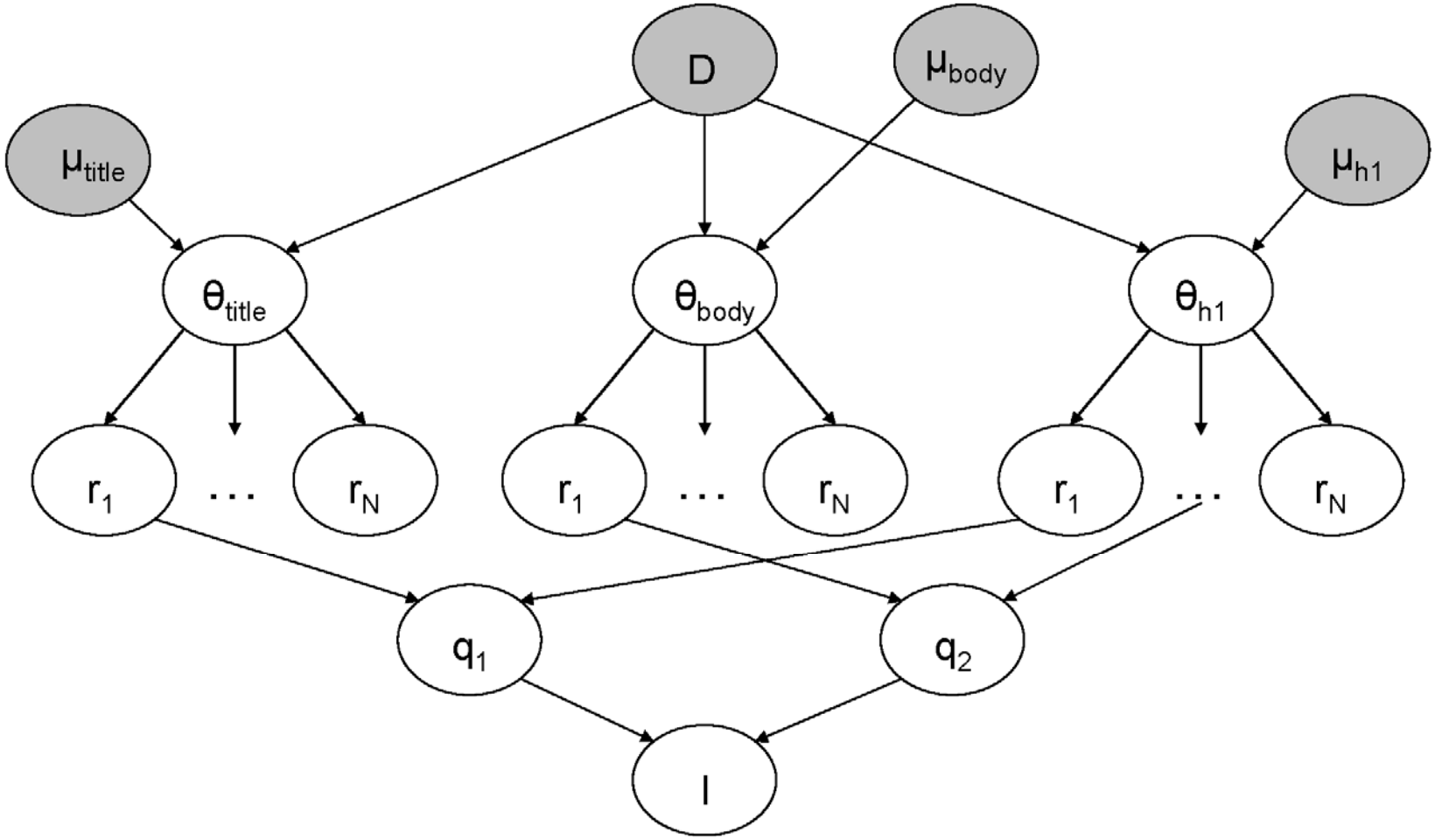
Example from Top 50 Docs

president lincoln	*abraham lincoln*	*fishing*	*tropical fish*
lincoln	lincoln	fish	fish
president	president	water	tropic
america	america	catch	water
new	abraham	reef	storm
national	war	fishermen	species
great	man	river	boat
white	civil	new	sea
war	new	year	river
washington	history	time	country
clinton	two	bass	tuna
house	room	boat	world
history	booth	world	million
time	time	farm	state
center	politics	angle	time
kennedy	public	fly	japan
room	guest	trout	mile

Combining Evidence

- Effective retrieval requires the combination of many pieces of evidence about a document's potential relevance
  - have focused on simple word-based evidence
  - many other types of evidence
    - structure, PageRank, metadata, even scores from different models
- *Inference network* model is one approach to combining evidence
  - uses Bayesian network formalism

Inference Network



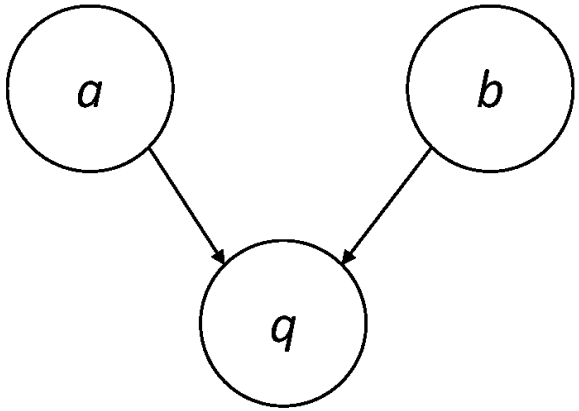
Inference Network

- *Document node* (D) corresponds to the event that a document is observed
- *Representation nodes* (r_i) are document features (evidence)
  - Probabilities associated with those features are based on language models θ estimated using the parameters μ
  - one language model for each significant document structure
  - r_i nodes can represent proximity features, or other types of evidence (e.g. date)

Inference Network

- *Query nodes* (q_i) are used to combine evidence from representation nodes and other query nodes
  - represent the occurrence of more complex evidence and document features
  - a number of combination operators are available
- *Information need node* (I) is a special query node that combines all of the evidence from the other query nodes
  - network computes $P(I|D, \mu)$

Example: AND Combination



a and *b* are *parent* nodes for q

$P(q = \text{TRUE} \mid a, b)$	a	b
0	FALSE	FALSE
0	FALSE	TRUE
0	TRUE	FALSE
1	TRUE	TRUE

Example: AND Combination

- Combination must consider all possible states of parents
- Some combinations can be computed efficiently

$$
\begin{aligned}
bel_{and}(q) &= p_{00}P(a = \text{FALSE})P(b = \text{FALSE}) \\
&\quad + p_{01}P(a = \text{FALSE})P(b = \text{TRUE}) \\
&\quad + p_{10}P(a = \text{TRUE})P(b = \text{FALSE}) \\
&\quad + p_{11}P(a = \text{TRUE})P(b = \text{TRUE}) \\
&= 0 \cdot (1 - p_a)(1 - p_b) + 0 \cdot (1 - p_a)p_b + 0 \cdot p_a(1 - p_b) + 1 \cdot p_a p_b \\
&= p_a p_b
\end{aligned}
$$

Inference Network Operators

$$
\begin{aligned}
bel_{not}(q) &= 1 - p_1 \\
bel_{or}(q) &= 1 - \prod_i^n (1 - p_i) \\
bel_{and}(q) &= \prod_i^n p_i \\
bel_{wand}(q) &= \prod_i^n p_i^{wt_i} \\
bel_{max}(q) &= max\{p_1, p_2, \ldots, p_n\} \\
bel_{sum}(q) &= \frac{\sum_i^n p_i}{n} \\
bel_{wsum}(q) &= \frac{\sum_i^n wt_i p_i}{\sum_i^n wt_i}
\end{aligned}
$$

Galago Query Language

- A document is viewed as a sequence of text that may contain arbitrary tags
- A single *context* is generated for each unique tag name
- An *extent* is a sequence of text that appears within a single begin/end tag pair of the same type as the context

Galago Query Language

```
<html>
<head>
<title>Department Descriptions</title>
</head>
<body>
The following list describes ...
<h1>Agriculture</h1> ...
<h1>Chemistry</h1> ...
<h1>Computer Science</h1> ...
<h1>Electrical Engineering</h1> ...
</body>
</html>
```

title context:

```
<title>Department Descriptions</title>
```

h1 context:

```
<h1>Agriculture</h1>
<h1>Chemistry</h1> ...
<h1>Computer Science</h1> ...
<h1>Electrical Engineering</h1> ...
```

body context:

```
<body> The following list describes ...
<h1>Agriculture</h1> ...
<h1>Chemistry</h1> ...
<h1>Computer Science</h1> ...
<h1>Electrical Engineering</h1> ...
</body>
```

Galago Query Language

Simple terms:

term – term that will be normalized and stemmed.
"term" – term is not normalized or stemmed.
Examples:
presidents
"NASA"

Galago Query Language

Proximity terms:

#od:N(...) – ordered window – terms must appear ordered, with at most N-1 terms between each.
#od(...) – unlimited ordered window – all terms must appear ordered anywhere within current context.
#uw:N(...) – unordered window – all terms must appear within a window of length N in any order.
#uw(...) – unlimited unordered window – all terms must appear within current context in any order.
Examples:
#od:1(white house) – matches "white house" as an exact phrase.
#od:2(white house) – matches "white * house" (where * is any word or null).
#uw:2(white house) – matches "white house" and "house white".

Galago Query Language

Synonyms:

#syn(...)
#wsyn(...)
Examples:
#syn(dog canine) – simple synonym based on two terms.
#syn(#od:1(united states) #od:1(united states of america)) – creates a synonym from two proximity terms.
#wsyn(1.0 donald 0.8 don 0.5 donnie) – weighted synonym indicating relative importance of terms.

Galago Query Language

Anonymous terms:

#any:.() – used to match extent types
Examples:
#any:person() – matches any occurrence of a person extent.
#od:1(lincoln died in #any:date()) – matches exact phrases of the form:"lincoln died in <date>...</date>".

Galago Query Language

Context restriction and evaluation:

expression.C1,,...,CN – matches when the expression appears in all contexts C1 through CN.
expression.(C1,...,CN) – evaluates the expression using the language model defined by the concatenation of contexts C1...CN within the document.
Examples:
dog.title – matches the term "dog" appearing in a title extent.
#uw(smith jones).author – matches when the two names "smith" and "jones" appear in an author extent.
dog.(title) – evaluates the term based on the title language model for the document.
#od:1(abraham lincoln).person.(header) – builds a language model from all of the "header" text in the document and evaluates #od:1(abraham lincoln).person in that context (i.e., matches only the exact phrase appearing within a person extent within the header context).

Galago Query Language

Belief operators:

#combine(...) – this operator is a normalized version of the $bel_{and}(q)$ operator in the inference network model. See the discussion below for more details.

#weight(...) – this is a normalized version of the $bel_{wand}(q)$ operator.

#filter(...) – this operator is similar to #combine, but with the difference that the document must contain at least one instance of all terms (simple, proximity, synonym, etc.). The evaluation of nested belief operators is not changed.

Galago Query Language

Examples:
#combine(#syn(dog canine) training) – rank by two terms, one of which is a synonym.
#combine(biography #syn(#od:1(president lincoln) #od:1(abraham lincoln))) – rank using two terms, one of which is a synonym of "president lincoln" and "abraham lincoln".
#weight(1.0 #od:1(civil war) 3.0 lincoln 2.0 speech) – rank using three terms, and weight the term "lincoln" as most important, followed by "speech", then "civil war".
#filter(aquarium #combine(tropical fish)) – consider only those documents containing the word "aquarium" and "tropical" or "fish", and rank them according to the query #combine(aquarium #combine(tropical fish)).
#filter(#od:1(john smith).author) #weight(2.0 europe 1.0 travel) – rank documents about "europe" or "travel" that have "John Smith" in the author context.

Web Search

- Most important, but not only, search application
- Major differences to TREC news
  - Size of collection
  - Connections between documents
  - Range of document types
  - Importance of spam
  - Volume of queries
  - Range of query types

Search Taxonomy

- *Informational*
  - Finding information about some topic which may be on one or more web pages
  - Topical search
- *Navigational*
  - finding a particular web page that the user has either seen before or is assumed to exist
- *Transactional*
  - finding a site where a task such as shopping or downloading music can be performed

Web Search

- For effective navigational and transactional search, need to combine features that reflect *user relevance*
- Commercial web search engines combine evidence from *hundreds* of features to generate a ranking score for a web page
  - page content, page metadata, anchor text, links (e.g., PageRank), and user behavior (click logs)
  - page metadata – e.g., “age”, how often it is updated, the URL of the page, the domain name of its site, and the amount of text content

Search Engine Optimization

- *SEO*: understanding the relative importance of features used in search and how they can be manipulated to obtain better search rankings for a web page
  - e.g., improve the text used in the title tag, improve the text in heading tags, make sure that the domain name and URL contain important keywords, and try to improve the anchor text and link structure
  - Some of these techniques are regarded as not appropriate by search engine companies

Web Search

- In TREC evaluations, most effective features for navigational search are:
  - text in the title, body, and heading (h1, h2, h3, and h4) parts of the document, the anchor text of all links pointing to the document, the PageRank number, and the inlink count
- Given size of Web, many pages will contain all query terms
  - Ranking algorithm focuses on discriminating between these pages
  - Word proximity is important

Term Proximity

- Many models have been developed
- N-grams are commonly used in commercial web search
- *Dependence model* based on inference net has been effective in TREC - e.g.

```
#weight(
        0.8 #combine(embryonic stem cells)
        0.1 #combine( #od:1(stem cells) #od:1(embryonic stem)
                      #od:1(embryonic stem cells))
        0.1 #combine( #uw:8(stem cells) #uw:8(embryonic cells)
                      #uw:8(embryonic stem) #uw:12(embryonic stem cells)))
```

Example Web Query

```
#weight(
        0.1 #weight( 0.6 #prior(pagerank) 0.4 #prior(inlinks))
        1.0 #weight(
            0.9 #combine(
                #weight( 1.0 pet.(anchor) 1.0 pet.(title)
                         3.0 pet.(body) 1.0 pet.(heading))
                #weight( 1.0 therapy.(anchor) 1.0 therapy.(title)
                         3.0 therapy.(body) 1.0 therapy.(heading)))
            0.1 #weight(
                1.0 #od:1(pet therapy).(anchor) 1.0 #od:1(pet therapy).(title)
                3.0 #od:1(pet therapy).(body) 1.0 #od:1(pet therapy).(heading))
            0.1 #weight(
                1.0 #uw:8(pet therapy).(anchor) 1.0 #uw:8(pet therapy).(title)
                3.0 #uw:8(pet therapy).(body) 1.0 #uw:8(pet therapy).(heading)))
        )
```

Machine Learning and IR

- Considerable interaction between these fields
  - Rocchio algorithm (60s) is a simple learning approach
  - 80s, 90s: learning ranking algorithms based on user feedback
  - 2000s: text categorization
- Limited by amount of training data
- Web query logs have generated new wave of research
  - e.g., “Learning to Rank”

Generative vs. Discriminative

- All of the probabilistic retrieval models presented so far fall into the category of *generative models*
  - A generative model assumes that documents were generated from some underlying model (in this case, usually a multinomial distribution) and uses training data to estimate the parameters of the model
  - probability of belonging to a class (i.e. the relevant documents for a query) is then estimated using Bayes' Rule and the document model

Generative vs. Discriminative

- A *discriminative* model estimates the probability of belonging to a class directly from the observed features of the document based on the training data
- Generative models perform well with low numbers of training examples
- Discriminative models usually have the advantage given enough training data
  - Can also easily incorporate many features

Discriminative Models for IR

- Discriminative models can be trained using explicit relevance judgments or click data in query logs
  - Click data is much cheaper, more noisy
  - e.g. Ranking Support Vector Machine (SVM) takes as input *partial rank* information for queries
    - partial information about which documents should be ranked higher than others

Ranking SVM

- Training data is

$$(q_1, r_1), (q_2, r_2), \ldots, (q_n, r_n)$$

  - r is partial rank information
    - if document d_a should be ranked higher than d_b, then $(d_a, d_b) \in r_i$
  - partial rank information comes from relevance judgments (allows multiple levels of relevance) or click data
    - e.g., d_1, d_2 and d_3 are the documents in the first, second and third rank of the search output, only d_3 clicked on → (d_3, d_1) and (d_3, d_2) will be in desired ranking for this query

Ranking SVM

- Learning a linear ranking function $\vec{w}.\vec{d}_a$
  - where *w* is a weight vector that is adjusted by learning
  - d_a is the vector representation of the features of document
  - *non-linear* functions also possible
- Weights represent importance of features
  - learned using training data
  - e.g.,

$$\vec{w}.\vec{d} = (2, 1, 2).(2, 4, 1) = 2.2 + 1.4 + 2.1 = 10$$

Ranking SVM

- Learn *w* that satisfies as many of the following conditions as possible:

$$\begin{aligned} \forall (d_i, d_j) \in r_1 \quad &: \quad \vec{w}.\vec{d_i} > \vec{w}.\vec{d_j} \\ &\ldots \\ \forall (d_i, d_j) \in r_n \quad &: \quad \vec{w}.\vec{d_i} > \vec{w}.\vec{d_j} \end{aligned}$$

- Can be formulated as an *optimization* problem

Ranking SVM

$$
\begin{aligned}
minimize: \quad & \frac{1}{2}\vec{w}.\vec{w} + C\sum \xi_{i,j,k} \\
subject\ to: & \\
\forall (d_i, d_j) \in r_1 \quad : \quad & \vec{w}.\vec{d_i} > \vec{w}.\vec{d_j} + 1 - \xi_{i,j,1} \\
& \ldots \\
\forall (d_i, d_j) \in r_n \quad : \quad & \vec{w}.\vec{d_i} > \vec{w}.\vec{d_j} + 1 - \xi_{i,j,n} \\
& \forall i \forall j \forall k : \xi_i, j, k \geq 0
\end{aligned}
$$

– ξ, known as a slack variable, allows for misclassification of difficult or noisy training examples, and *C* is a parameter that is used to prevent overfitting

Ranking SVM

- Software available to do optimization
- Each pair of documents in our training data can be represented by the vector:

$$(\vec{d}_i - \vec{d}_j)$$

- Score for this pair is:

$$\vec{w}.(\vec{d}_i - \vec{d}_j)$$

- SVM classifier will find a *w* that makes the smallest score as large as possible
  - make the differences in scores as large as possible for the pairs of documents that are hardest to rank

Topic Models

- Improved representations of documents
  - can also be viewed as improved smoothing techniques
  - improve estimates for words that are related to the topic(s) of the document
    - instead of just using background probabilities
- Approaches
  - *Latent* Semantic Indexing (LSI)
  - Probabilistic *Latent* Semantic Indexing (pLSI)
  - *Latent* Dirichlet Allocation (LDA)

LDA

- Model document as being generated from a *mixture* of topics

1. For each document D, pick a multinomial distribution θ_D from a Dirichlet distribution with parameter α ,
2. For each word position in document D,
   (a) pick a topic z from the multinomial distribution θ_D ,
   (b) Choose a word w from $P(w|z, \beta)$, a multinomial probability conditioned on the topic z with parameter β.

LDA

- Gives language model probabilities

$$P_{lda}(w|D) = P(w|\theta_D, \beta) = \sum_z P(w|z, \beta)P(z|\theta_D)$$

- Used to smooth the document representation by mixing them with the query likelihood probability as follows:

$$P(w|D) = \lambda \left(\frac{f_{w,D} + \mu \frac{c_w}{|C|}}{|D| + \mu} \right) + (1 - \lambda) P_{lda}(w|D)$$

LDA

- If the LDA probabilities are used directly as the document representation, the effectiveness will be significantly reduced because the features are *too smoothed*
  - e.g., in typical TREC experiment, only 400 topics used for the *entire* collection
  - generating LDA topics is expensive
- When used for smoothing, effectiveness is improved

LDA Example

– Top words from 4 LDA topics from TREC news

Arts	*Budgets*	*Children*	*Education*
new	million	children	school
film	tax	women	students
show	program	people	schools
music	budget	child	education
movie	billion	years	teachers
play	federal	families	high
musical	year	work	public
best	spending	parents	teacher
actor	new	says	bennett
first	state	family	manigat
york	plan	welfare	namphy
opera	money	men	state
theater	programs	percent	president
actress	government	care	elementary
love	congress	life	haiti

Summary

- Best retrieval model depends on application and data available
- Evaluation corpus (or test collection), training data, and user data are all critical resources
- Open source search engines can be used to find effective ranking algorithms
  - Galago query language makes this particularly easy
- Language resources (e.g., thesaurus) can make a big difference