

Structured Retrieval Models and Query Languages

CISC489/689-010, Lecture #14

Wednesday, April 8th

Ben Carterette

Query Language

- To this point we have assumed queries are just a few keywords in no particular order and with no structure
 - Exceptions:
 - Phrases
 - Boolean retrieval
- A query language is a way to give the engine more information about the relationships between terms, importance of terms, etc

Examples

- Boolean queries
 - $t_1 \text{ AND } (t_2 \text{ OR } t_3) \text{ AND NOT } t_4$
- Google queries
 - “ $t_1 t_2$ ” + t_3 – t_4 site:www.udel.edu
 - allintitle: “ $t_1 * t_2$ ”
- Structured language
 - #combine(biography #syn(#1(president lincoln) #1(abraham lincoln)))

Query Languages and Indexes

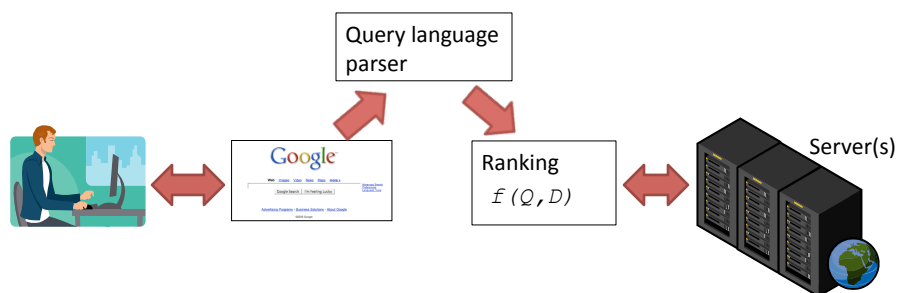
- The query language is only as good as the index
 - Your query language can support phrases,
 - It can support document structure,
 - It can support date ranges,
 - But none of that matters if there’s no support in the index!
- Monday we talked about storing information to support phrases and structure

Query Languages and Retrieval Models

- Query language is only as good as the retrieval model behind it
 - If your retrieval model does not model phrases,
 - If it does not model document structure,
 - If it does not model date ranges,
 - Then a query language supporting those is no use.
- Most of the models we've discussed so far only model terms independently
 - One exception: bigram language models

Query Process

Query → parsed according to language
 → turned into abstract representation Q according to retrieval model



Document → parsed at index time, features needed for retrieval model stored on disk
 → turned into abstract representation D according to retrieval model

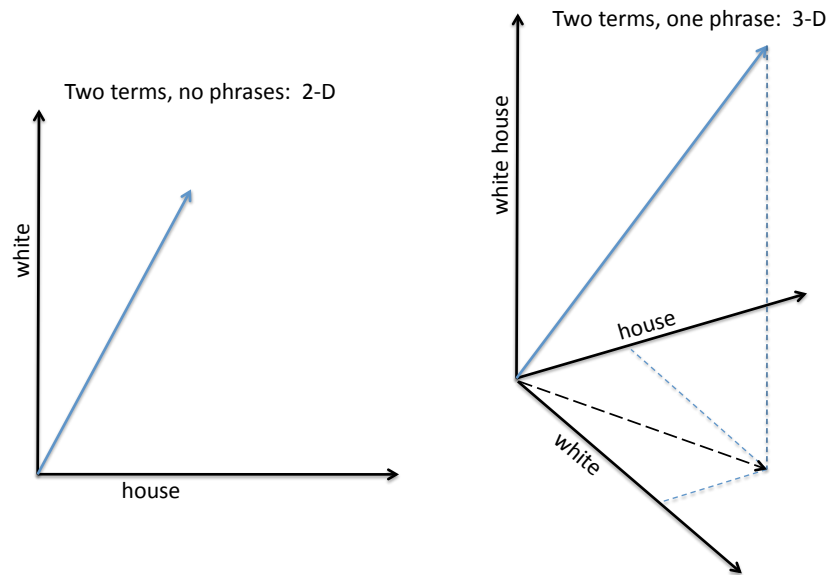
A Simple Example

- Suppose we have an inverted index that can support phrases
- We have an engine that uses the vector space retrieval model and cosine similarity scoring
- We want to add support for phrases to the query language
 - Vector space model does not support phrases by default
 - We need to add phrase support to the VSM

Phrases in the VSM

- Add phrase support to the vector space model
 - Document represented as vector of feature weights
 - Features are terms *and phrases*
 - Therefore each phrase becomes a new dimension in the vector space
 - Term weight: $w_{ik} = \frac{tf_{ik}}{|D_k|} \log \frac{N}{n_i}$
 - Phrase weight: $pw_{ik} = \frac{pf_{ik}}{|D_k|} \log \frac{N}{np_i}$

Phrasal VSM Illustration



A More Complex Example

- Suppose we have an inverted index that supports field information
- We have an engine that uses the vector space retrieval model and cosine similarity scoring
- We want to add support for field information to the query language, e.g. `intitle:white house`
 - Vector space model does not support this by default
 - We need to add field support to the VSM

Fields in the VSM

- One possibility: separate term weights for each field the term appears in

$$w_{ik}^{title} = \frac{tf_{ik}^{title}}{|title_k|} \log \frac{N}{n_i^{title}}$$



$$w_{ik}^{body} = \frac{tf_{ik}^{body}}{|body_k|} \log \frac{N}{n_i^{body}}$$

$$\dots$$

Fields in the Vector Space Model

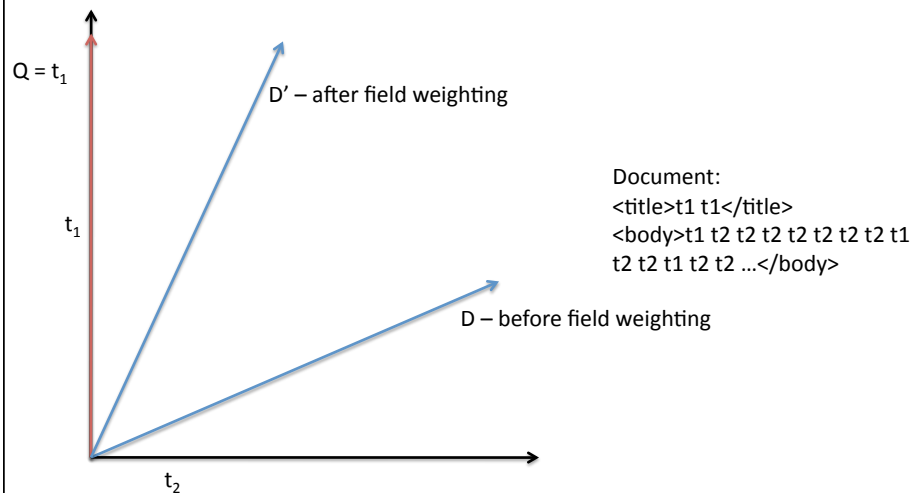
- Another possibility: weigh terms according to the fields they occur in

$$w_{ik} = \frac{tf_{ik}}{|D_k|} \log \frac{N}{n_i} \prod_{f:i \in f} \frac{fw_f}{|f_k|}$$

 a field weight factor
 length of field in k

- If <title> weight is high, terms that appear in title will count for more in cosine similarity
- If <script> weight is low, terms that appear in script will count for less in cosine similarity

Fields in the VSM: Illustration



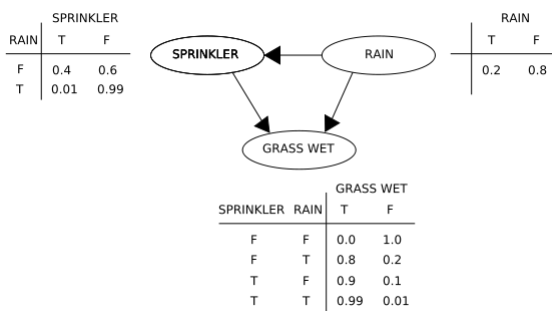
Formal Motivation

- As before, VSM provides no formal motivation for setting weights
 - How do we set field weights?
 - How do we combine field weights?
 - How do we weight terms within fields?
 - etc.
- Probabilistic model can offer formalism, but it is not straightforward to incorporate these features into probabilistic models

Combining Evidence

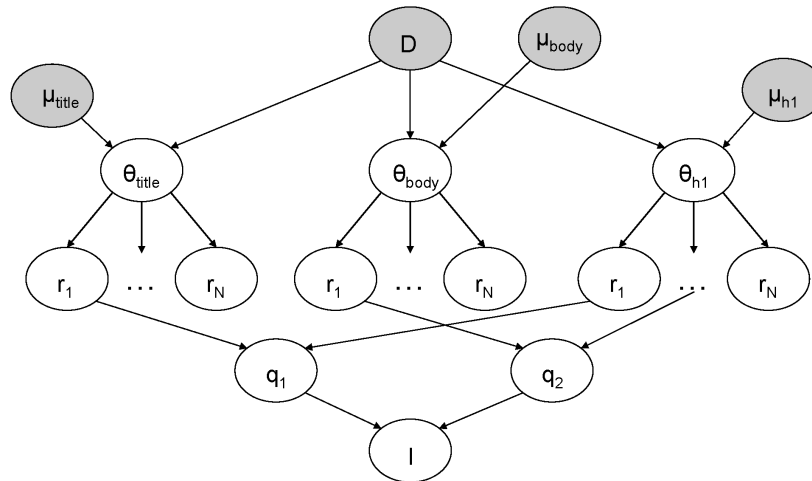
- Take each term, field, phrase, etc, as pieces of *evidence* about a document's relevance
- Combine pieces of evidence into a final score for the document
- There are formal ways to motivate the combination
- *Inference network* model is one approach to combining evidence
 - uses Bayesian network formalism with language model probabilities

Inference Network: Non-IR Example



$$\begin{aligned}
 P(R = T \mid G = T) &= \frac{P(G = T, R = T)}{P(G = T)} = \frac{\sum_{S \in \{T, F\}} P(G = T, S, R = T)}{\sum_{S, R \in \{T, F\}} P(G = T, S, R)} \\
 &= \frac{(0.99 \times 0.01 \times 0.2 = 0.00198_{TTT}) + (0.8 \times 0.99 \times 0.2 = 0.1584_{TFT})}{0.00198_{TTT} + 0.288_{TTF} + 0.1584_{TFT} + 0_{TFF}} \approx 35.77\%.
 \end{aligned}$$

Inference Network in IR



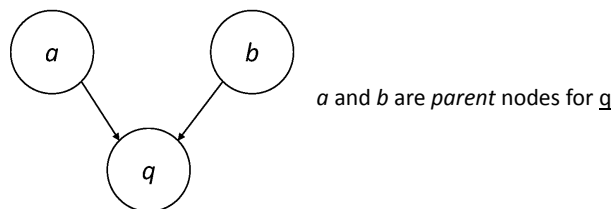
Inference Network

- *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



$P(q = \text{TRUE} a, b)$	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

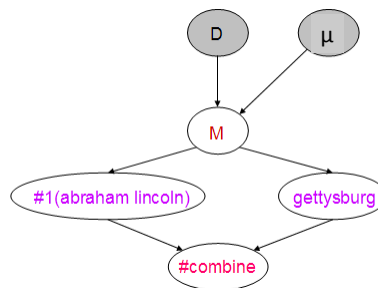
$$\begin{aligned}
 bel_{and}(q) &= p_{00}P(a = \text{FALSE})P(b = \text{FALSE}) \\
 &\quad + p_{01}P(a = \text{FALSE})P(b = \text{TRUE}) \\
 &\quad + p_{10}P(a = \text{TRUE})P(b = \text{FALSE}) \\
 &\quad + p_{11}P(a = \text{TRUE})P(b = \text{TRUE}) \\
 &= 0 \cdot (1 - p_a)(1 - p_b) + 0 \cdot (1 - p_a)p_b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

Inference Network Operators

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

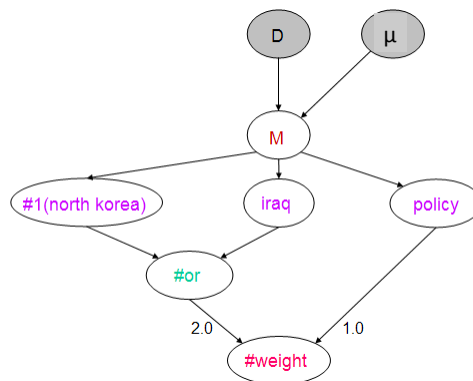
Inference Network Example

Query: #combine(#1(abraham lincoln) gettysburg)



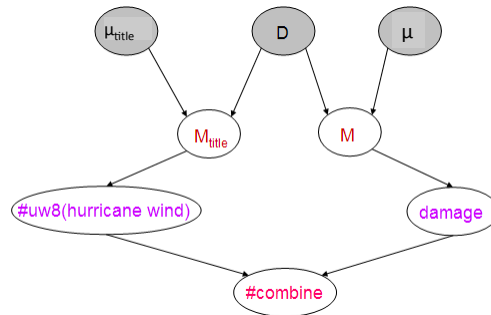
Inference Network Example

Query: #weight(2.0 #or(#1(north korea) iraq) 1.0 policy)



Inference Network Example

Query: #combine(#uw8(hurricane wind).(title) damage)



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

<pre> <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> </pre>	<p>title context:</p> <pre> <title>Department Descriptions</title> </pre> <p>h1 context:</p> <pre> <h1>Agriculture</h1> <h1>Chemistry</h1> ... <h1>Computer Science</h1> ... <h1>Electrical Engineering</h1> ... </pre> <p>body context:</p> <pre> <body> The following list describes ... <h1>Agriculture</h1> ... <h1>Chemistry</h1> ... <h1>Computer Science</h1> ... <h1>Electrical Engineering</h1> ... </body> </pre>
---	---

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.