Search Engines
Chapter 7 – Retrieval Models

21.6.2011
Felix Naumann
The Indexing Process

Text and metadata for all documents

Identifies and stores documents for indexing

Document data store

Text Acquisition

Index Creation

Transforms documents into index terms or features

Takes index terms and creates data structures (indexes) to support fast searching

Index (inverted index)

Text Transformation
The Query Process

Supports creation and refinement of query, display of results

Document data store

Uses query and indexes to generate ranked list of documents

Index

Monitors and measures effectiveness and efficiency (primarily offline)

Log data

Evaluation

Ranking (retrieval model)

User Interaction

Felix Naumann | Search Engines | Sommer 2011
Abstract Model of Ranking

Numerical values generated by feature functions

9.7 fish
4.2 tropical
22.1 tropical fish
8.2 seaweed
4.2 surfboards

Topical Features

14 incoming links
3 days since last update

Quality Features

Plus context features

High value predicts good match

tropical fish
Query

24.5 Document Score

Typically ignores very many features

Final output: Documents sorted descending by document score

Fred's Tropical Fish Shop is the best place to find tropical fish at low, low prices. Whether you're looking for a little fish or a big fish, we've got what you need. We even have fake seaweed for your fishtank (and little surfboards too).
More Concrete Model

\[ R(Q, D) = \sum_i g_i(Q) f_i(D) \]

- \( f_i \) is a document feature function
- \( g_i \) is a query feature function

Fred’s Tropical Fish Shop is the best place to find tropical fish at low, low prices. Whether you’re looking for a little fish or a big fish, we’ve got what you need. We even have fake seaweed for your fish tank (and little surfboards too).

http://www.howard.k12.md.us/res/aquariums/chichlids.html

Only few; others are zero.
Retrieval Models

- Provide a mathematical framework for defining the search process
  - Formalize human process of making decisions about relevance.
    - Framework should at least correlate well.
  - Basis of many ranking algorithms
  - Can be implicit

- Progress in retrieval models has corresponded with improvements in effectiveness.
  - Improvement of 100% in 90s (TREC)

- Mostly: Theories about relevance
Relevance

- Complex concept, studied for some time
  - Many factors to consider
  - People often disagree when making relevance judgments.
    - Inter-annotator disagreement
- Retrieval models make various assumptions about relevance to simplify problem.
  - Topical vs. user relevance
    - Topical relevance: Document is of same topic
    - User relevance: All other factors
      - Some are used in some retrieval models
  - Binary vs. multi-valued relevance
    - Relevant vs. non-relevant
    - Relevant vs. unsure vs. non-relevant
    - Retrieval models usually are more detailed (probability)
Overview

- Older models
  - Boolean retrieval
  - Vector Space model
- Probabilistic models
- Language models
- Combining evidence
- Web search
- Learning to Rank
Boolean Retrieval

- Two possible outcomes for query processing
  - TRUE or FALSE
  - “Exact-match” semantics
  - Simplest form of ranking
    - All matching documents are considered equally relevant.
- Query usually specified using Boolean operators
  - AND, OR, NOT
  - Textual proximity operators also used
Boolean Retrieval

- Advantages
  - Results are predictable and relatively easy to explain.
  - Many different features can be incorporated
    - Date, document type, ...
  - Efficient processing since many documents can be eliminated from search

- Disadvantages
  - Effectiveness depends entirely on user.
    - Presentation order not based on relevance
      - But arbitrarily on date, size, etc.
  - Simple queries usually don’t work well.
  - Complex queries are difficult to write.
    - Search intermediaries (e.g. in legal offices)

NEGLIGENCE! FAIL! NEGLIGENCE! /5 MAINT! REPAIR! /P NAVIGATE! /5 AID EQUIPMENT! LIGHT BUOY "CHANNEL MARKER"
“Searching by Numbers”

- Sequence of queries driven by number of retrieved documents
  - Search of news articles for President Lincoln
    1. lincoln
      ◊ Result: cars, places, people
    2. president AND lincoln
      ◊ Result: “Ford Motor Company today announced that Darryl Hazel will succeed Brian Kelley as president of Lincoln Mercury.”
    3. president AND lincoln AND NOT (automobile OR car)
      ◊ Not in result: “President Lincoln’s body departs Washington in a nine-car funeral train.”
    4. president AND lincoln AND biography AND life AND birthplace AND gettysburg AND NOT (automobile OR car)
      ◊ Result: Ø
    5. president AND lincoln AND (biography OR life OR birthplace OR gettysburg) AND NOT (automobile OR car)
      ◊ Top result might be: “President’s Day - Holiday activities – crafts, mazes, word searches, ... ‘The Life of Washington´ Read the entire book online! Abraham Lincoln Research Site’
Vector Space Model

- Very popular model, even today
  - Simple, intuitive
  - Useful for weighting, ranking, and relevance feedback
- Documents and query represented by a vector of term weights
  - \( t \) is number of index terms (i.e., very large)
    \[
    D_i = (d_{i1}, d_{i2}, \ldots, d_{it}) \quad Q = (q_1, q_2, \ldots, q_t)
    \]
- Collection represented by a matrix of term weights

\[
\begin{array}{cccc}
  & \text{Term}_1 & \text{Term}_2 & \ldots & \text{Term}_t \\
\text{Doc}_1 & d_{11} & d_{12} & \ldots & d_{1t} \\
\text{Doc}_2 & d_{21} & d_{22} & \ldots & d_{2t} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\text{Doc}_n & d_{n1} & d_{n2} & \ldots & d_{nt}
\end{array}
\]
Vector Space Model – Example

- **D₁**: Tropical Freshwater Aquarium Fish.
- **D₂**: Tropical Fish, Aquarium Care, Tank Setup.
- **D₃**: Keeping Tropical Fish and Goldfish in Aquariums, and Fish Bowls.
- **D₄**: The Tropical Tank Homepage - Tropical Fish and Aquariums.

<table>
<thead>
<tr>
<th>Terms</th>
<th>D₁</th>
<th>D₂</th>
<th>D₃</th>
<th>D₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>aquarium</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>bowl</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>care</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fish</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>freshwater</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>goldfish</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>homepage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>keep</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>setup</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>tank</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>tropical</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Rotated

Weights are term counts

Stopwords are removed

Query for „tropical fish“ (0 0 0 1 0 0 0 0 0 0 0 0 1)
3-d pictures useful, but can be misleading for high-dimensional space

- Intuition no longer necessarily correct
- Millions of terms (and dimensions)
Vector Space Model

- Each document ranked by distance between points representing query and document
  - *Similarity* measure more common than a distance or *dissimilarity* measure
  - Popular: Cosine correlation
    - Cosine of angle between document and query vectors
    - Normalized dot-product
      \[
      \text{Cosine}(D_i, Q) = \frac{\sum_{j=1}^{t} d_{i,j} \cdot q_j}{\sqrt{\sum_{j=1}^{t} d_{i,j}^2 \cdot \sum_{j=1}^{t} q_j^2}}
      \]
  - As retrieval model: No explicit definition of relevance
    - Implicit: Closer documents are more relevant.
Similarity Calculation – Example

- Consider three documents $D_1$, $D_2$, $D_3$ and a query $Q$
  - $D_1 = (0.5, 0.8, 0.3), D_2 = (0.9, 0.4, 0.2), D_3 = (0, 0.9, 0.1)$
  - $Q = (1.5, 1.0, 0)$

- Vector space model reflects term weights and number of matching terms (in contrast to Boolean retrieval)

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

\[
\text{Cosine}(D_2, Q) = \frac{(0.9 \times 1.5) + (0.4 \times 1.0)}{\sqrt{(0.9^2 + 0.4^2 + 0.2^2)(1.5^2 + 1.0^2)}} = \frac{1.75}{\sqrt{(1.01 \times 3.25)}} = 0.97 \quad \text{Cosine}(D_3, Q) = 0.55
\]

- But: How to assign term weights?
Term Weights – *tf.idf*

- **Term frequency weight** *tf* measures importance in document *i*:
  \[
  tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} f_{ij}}
  \]
  - Long documents have many words with only one occurrence but also many with hundreds of occurrences
  - \(\log(f_{ik})\) to reduce this impact of frequent words

- Inverse document frequency *idf* measures importance in collection: \(idf_{k} = \log \frac{N}{n_k}\)
  - Reflects “amount of information” carried by term

- **tfidf** by multiplying *tf* and *idf* with some heuristic modifications
  \[
  d_{ik} = \frac{(\log(f_{ik})+1) \cdot \log(N/n_k)}{\sqrt{\sum_{k=1}^{t} [(\log(f_{ik})+1.0) \cdot \log(N/n_k)]^2}}
  \]
  +1 to ensure non-zero weight
  Normalization usually done by cosine similarity
Relevance Feedback – Rocchio algorithm

- Determine *Optimal query*
  - Maximizes the difference between average vector representing the relevant documents and average vector representing the non-relevant documents
  - Usually only limited feedback (i.e., not for all documents). Thus, only modify query weights:
    \[ q'_j = \alpha q_j + \beta \frac{1}{|\text{Rel}|} \sum_{D \in \text{Rel}} d_{ij} - \gamma \frac{1}{|\text{Nonrel}|} \sum_{D \in \text{Nonrel}} d_{ij} \]
  - \( q_j \) is initial term weight
  - \( \text{Rel} \) is set of relevant documents
  - \( \text{Nonrel} \) is set of non-relevant documents
    - Approximate as “all unseen documents”
  - \( \alpha, \beta, \) and \( \gamma \) are parameters to control effect of components
    - Typical values 8, 16, 4
  - Even query terms with \( q_j = 0 \) can be modified: New terms may be added (usually restricted to 50).
  - And vice versa: Terms may accrue negative weight and are dropped.
Overview

- Older models
- Probabilistic models
- Language models
- Combining evidence
- Web search
- Learning to Rank
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.”

- Probability theory is a strong foundation for representing and manipulating the inherent uncertainty.
- Problem: How to estimate probability of relevance?
  - Each model has different suggestion
IR as Classification

Actually, we just need a ranking

Felix Naumann | Search Engines | Sommer 2011
Bayes Classifier

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

- **Estimating probabilities**
  - Use Bayes Rule
    \[ P(R|D) = \frac{P(D|R)P(R)}{P(D)} \]
  - Determining $P(D|R)$ should be easier: Given information about the relevant set (e.g. relevant words/query), determine how likely it is to see the same properties in $D$.

- **Example**
  - Probability of "president" in relevant set is 0.02.
  - Probability of "lincoln" in relevant set is 0.03.
  - New document with "president" and "lincoln". Probability of observing that combination is 0.0006.
Bayes Classifier

- Bayes rule \( P(R|D) = \frac{P(D|R)P(R)}{P(D)} \)
  - \( P(R) \) is apriori probability of relevance (how likely is any document to be relevant)
  - \( P(D) \) is normalizing constant.
- Before: \( D \) relevant if \( P(R|D) > P(NR|D) \).
  - \( \iff P(D|R)P(R) > P(D|NR)P(NR) \)
- Now: Classify a document as relevant if
  \[ \frac{P(D|R)}{P(D|NR)} > \frac{P(NR)}{P(R)} \]
  - lhs is likelihood ratio
- Classification needs to make decision.
- Search engine only needs to rank.
  - Rank by likelihood ratio, ignore rhs
Estimating P(D|R)

- **Binary independence model**
  - Document represented as combinations of terms:
    - Vector of binary features indicating term occurrence (or non-occurrence)
  - Represent R and NR as term-probabilities
  - $p_i$ is probability that term $i$ occurs (i.e., has value 1) in relevant document, $s_i$ is probability of occurrence in non-relevant document

- Assume independence (Naïve Bayes assumption)
  $$P(D|R) = \prod_{i=1}^{t} P(d_i|R)$$
  - Assumption is obviously incorrect, but successful

- Example:
  - Document $D$ contains words 1, 4, and 5: (1,0,0,1,1)
  - Let $p_i$ denote probability that term $i$ is in relevant set
  - Relevance-probability of $D$ is $p_1 \times (1-p_2) \times (1-p_3) \times p_4 \times p_5$
Binary Independence Model

- Let $p_i$ denote probability that term $i$ occurs in relevant set
- Let $s_i$ denote probability that term $i$ occurs in non-relevant set
- Reminder: Classify document as relevant if $\frac{P(D|R)}{P(D|NR)} > \frac{P(NR)}{P(R)}$
  - Or rank according to lhs

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

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

\[
= \prod_{i:d_i=1} \frac{p_i(1-s_i)}{s_i(1-p_i)} \cdot \prod_i \frac{1-p_i}{1-s_i}
\]
Binary Independence Model

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

- Second term is over all documents, thus ignore
- To avoid accuracy problems, use log
- 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.
  - Summation only over terms that appear in query and document
- Simplification
  - If no further information about relevant set, assume \( p_i \) constant (e.g., 0.5)
  - Approximate \( s_i \) by entire collection (because number of relevant documents is very small).
  - Get \( idf \)-like weight
    - No \( tf \)-component, because binary features
    \[
    \log \frac{0.5(1-\frac{n_i}{N})}{\frac{n_i}{N}(1-0.5)} = \log \frac{N-n_i}{n_i}
    \]
If we do have information about term occurrences in relevant and non relevant information (through relevance feedback or pseudo-relevance feedback): Store in contingency table

- $r_i$ is number of relevant documents containing term $i$.
- $R$ is number of relevant documents for query.
- $n_i$ is number of documents containing term $i$.
- $N$ is total number of documents.

<table>
<thead>
<tr>
<th>Term $i$ is present:</th>
<th>Relevant</th>
<th>Non-relevant</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_i = 1$</td>
<td>$r_i$</td>
<td>$n_i - r_i$</td>
<td>$n_i$</td>
</tr>
<tr>
<td>$d_i = 0$</td>
<td>$R - r_i$</td>
<td>$N - n_i - R + r_i$</td>
<td>$N - n_i$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term $i$ is not present:</th>
<th>Relevant</th>
<th>Non-relevant</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>$R$</td>
<td>$N - R$</td>
<td>$N$</td>
</tr>
</tbody>
</table>
Idea: Use table to estimate $p_i$ and $s_i$ for scoring function

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

Obvious choices

- $p_i = r_i/R$
- $s_i = (n_i - r_i)/(N - R)$
- Problem if $r_i = 0$
- Solution: Add 0.5 to counts and 1 to totals

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

Gives scoring function:

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

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

- Uses only matching query terms
  - But: Relevance feedback can be used to expand query
- Not very good in practice
  - Missing \( tf \) component lowers effectiveness by 50%
  - I.e., 50% less relevant documents in top 10 compared to \( tfidf \) rankings
- But: Basis for BM25
  - Best Match variant 25
BM25

- Popular and effective ranking algorithm based on binary independence model
  - Adds document and query term weights
- Scoring function
  \[
  \sum_{i \in Q} \log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} \cdot \frac{(k_1 + 1)f_i}{K + f_i} \cdot \frac{(k_2 + 1)qf_i}{k_2 + qf_i}
  \]
  - Summation over all query terms
  - \(f_i\) is frequency of term \(i\) in document
  - \(qf_i\) is frequency of term \(i\) in query
  - \(k_1, k_2\) and \(K\) are parameters whose values are set empirically.
- Reminders
  - \(r_i\) is number of relevant documents containing term \(i\).
    - Set to 0, if no relevance information
  - \(R\) is number of relevant documents for query.
    - Set to 0, if no relevance information
  - \(n_i\) is number of documents containing term \(i\).
  - \(N\) is total number of documents.
BM25 – Interpretation

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

- \( k_1 \) determines how \( tf \) component of term weight changes as \( f_i \) increases
  - \( k_1 = 0 \): term frequency ignored, only term presence
  - Typical: \( k_1 = 1.2 \), thus first few occurrences have most impact
- \( k_2 \) same for query term frequency
  - Typical: \( 0 \leq k_2 \leq 1000 \)
  - Not sensitive, because low frequencies
- \( K \) normalizes \( tf \) component by document length (\( dl \)).
  \[ K = k_1((1 - b) + b \cdot \frac{dl}{avdl}) \]
  - \( b \) regulates length normalization
    - \( b = 0 \): No normalization
    - \( b = 1 \): Full normalization
    - Typical: \( b = 0.75 \)
Query with two terms, “president lincoln”, \(qf = 1\)

No relevance information: \(r = R = 0\)

\(N = 500,000\) 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 \(dl/avdl = 0.9\)

\(k_1 = 1.2, b = 0.75,\) and \(k_2 = 100\)

\[K = 1.2 \cdot (0.25 + 0.75 \cdot 0.9) = 1.11\]
BM25 Example

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

\[ BM25(Q, D) = \]

\[ \log \frac{(0 + 0.5)/(0 - 0 + 0.5)}{(40000 - 0 + 0.5)/(500000 - 40000 - 0 + 0 + 0.5)} \]
\[ \times \frac{(1.2 + 1)15}{1.11 + 15} \times \frac{(100 + 1)1}{100 + 1} \]
\[ + \log \frac{(0 + 0.5)/(0 - 0 + 0.5)}{(300 - 0 + 0.5)/(500000 - 300 - 0 + 0 + 0.5)} \]
\[ \times \frac{(1.2 + 1)25}{1.11 + 25} \times \frac{(100 + 1)1}{100 + 1} \]

\[ = \log \frac{460000.5}{40000.5} \cdot \frac{33}{16.11} \cdot \frac{101}{101} \]
\[ + \log \frac{499700.5}{300.5} \cdot \frac{55}{26.11} \cdot \frac{101}{101} \]

\[ = 2.44 \cdot 2.05 \cdot 1 + 7.42 \cdot 2.11 \cdot 1 \]

\[ = 5.00 + 15.66 = 20.66 \]
Effect of term frequencies

<table>
<thead>
<tr>
<th>Frequency of “president”</th>
<th>Frequency of “lincoln”</th>
<th>BM25 score</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>25</td>
<td>20.66</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>12.74</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>5.00</td>
</tr>
<tr>
<td>1</td>
<td>25</td>
<td>18.2</td>
</tr>
<tr>
<td>0</td>
<td>25</td>
<td>15.66</td>
</tr>
</tbody>
</table>

Even one occurrence of *lincoln* makes for a large difference in score.
- Occurrence of president less important
- Document with very many occurrences of one word can be better than one with both words.
  - 15.66 > 12.74
BM25 – Discussion

Seems complicated, but

- Calculation of term weights at index time
- With no relevance info, just add weights for matching query terms
  - Plus some additional calculation for multiple query terms ($qf > 1$)

Well tuneable to different applications

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

- Older models
- Probabilistic models
- Language models
- Combining evidence
- Web search
- Learning to Rank
Language Model

- Language model applications
  - Speech recognition, machine translation, handwriting recognition
  - And information retrieval
- Predicts which word is next in a sequence of words.
- **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.
  - Next word not dependent on previous word(s).
  - Example for language with 5 words: (.2, .1, .35, .25, .1)
- **N-gram language model**
  - Predicts next word based on previous \( n-1 \) words.
  - Some applications use bigram and trigram language models where probabilities depend on previous words.
- Bi- and tri-grams expensive – unigram models suffice for search applications.
A topic in a document can be represented as a language model
  - i.e., as a distribution over words.
  - Words that tend to occur often when discussing a topic will have high probabilities in the corresponding language model
  - In general: Distribution over all words, but most (unimportant words) will have default probability.

**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
  - Does not model *burstiness*
    - Occurrence of a word makes repeated occurrence more likely
  - Not here...

The topic of a query can also be represented as language model.
Three possibilities to use language models for retrieval:

1. Probability of generating the query text from a document language model
2. Probability of generating the document text from a query language model
3. Comparing the language models representing the query and document topics

Models of topical relevance

- Query-Likelihood Model
- Relevance model / document-likelihood model
Query-Likelihood Model

- Rank documents by the probability that the query could be generated by the document model
  - Probability that we could pull the query words from the bucket of document words
  - i.e., same topic
- Given query, start with $P(D|Q)$
- Using Bayes’ Rule, ignoring normalizing constant $P(Q)$

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

- Assuming prior is uniform, unigram model

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

- Possible non-uniform prior: Use date or document length
Obvious estimate for unigram probabilities is

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

- **Maximum likelihood estimate**
  - makes the observed value of \( f_{q;D} \) most likely

- **Problems:**
  - If 1 query word out of 6 is missing from document, score will be zero
  - Missing 1 out of 6 query words same as missing 5 out of 6
  - Words associated with topic should have some probability, even if they do not appear in document.
    - Assign at least some small probability

- **Thus: Smoothing**
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.
    - Usually based on frequency of words in entire collection of documents
Estimating Probabilities

- Estimate for unseen words is $a_D P(q_i|C)$
  - $P(q_i|C)$ is the probability for query word $i$ in the collection language model for collection $C$ (background probability)
  - $a_D$ is a parameter between 0 and 1
- Estimate for words that occur is $(1 - a_D) P(q_i|D) + a_D P(q_i|C)$
  - To ensure summation to 1
- Different forms of estimation come from different $a_D$
- Example: Only three words in collection $w_1$, $w_2$, $w_3$
  - $P(w_1|C) = 0.3 \quad P(w_2|C) = 0.5 \quad P(w_3|C) = 0.2$
  - $P(w_1|D) = 0.5 \quad P(w_2|D) = 0.5 \quad P(w_3|D) = 0$
  - Smoothing
    - $P(w_1|D) = (1 - a_D) P(w_1|D) + a_D P(w_1|C) = (1 - a_D) 0.5 + a_D 0.3$
    - $P(w_2|D) = (1 - a_D) 0.5 + a_D 0.5$
    - $P(w_3|D) = (1 - a_D) 0.0 + a_D 0.2 \quad (= a_D 0.2 > 0 !)$
    - Test: $P(w_1|D) + P(w_2|D) + P(w_3|D) = 1$
- Variations based on different choices for $a_D$
Jelinek-Mercer Smoothing

- Simple choice: \( a_D \) is a constant, \( a_D = \lambda \)
- 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
  - Due to accuracy problems when multiplying many small numbers

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

- Small \( \lambda \) result in less smoothing, closer to Boolean AND
  - \( \lambda = 0.1 \) successful for short queries

- For high \( \lambda \) relative weighting less important, closer to Boolean OR
  - Coordination level match: Ranks by number of matching query terms
  - \( \lambda = 0.7 \) successful for very long queries

\[
\log_a \prod_{i=1}^{n} x_i = \sum_{i=1}^{n} \log_a x_i.
\]
Where is \emph{tf.idf}-like weight?

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

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

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

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

(proportional to the term frequency)

(inversely proportional to the collection frequency)

Add and subtract \[
\sum_{i:f_{q_i,D}>0} \log(\lambda \frac{c_{q_i}}{|C|})
\]

Same for all documents: Ignore

Split into words that occur and those that do not
More effective choice: let $a_D$ depend on document length:

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

Substituted in $(1 - a_D) P(q_i|D) + a_D P(q_i|C)$ gives probability estimation

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

and document score

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

Small values for $\mu$ give more importance to relative term weights.

Large values favor number of matching terms.

Typical: $1,000 \leq \mu \leq 2,000$
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

- $\mu = 2,000$

$$QL(Q, D) = \log \frac{15 + 2000 \times (1.6 \times 10^5/10^9)}{1800 + 2000} + \log \frac{25 + 2000 \times (2400/10^9)}{1800 + 2000}$$

$$= \log(15.32/3800) + \log(25.005/3800)$$

$$= -5.51 + -5.02 = -10.53$$

- Negative number because summing logs of small numbers
- Only ranking is relevant
## Query Likelihood Example

<table>
<thead>
<tr>
<th>Frequency of “president”</th>
<th>Frequency of “lincoln”</th>
<th>QL score</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>25</td>
<td>-10.53</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>-13.75</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>-19.05</td>
</tr>
<tr>
<td>1</td>
<td>25</td>
<td>-12.99</td>
</tr>
<tr>
<td>0</td>
<td>25</td>
<td>-14.40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frequency of “president”</th>
<th>Frequency of “lincoln”</th>
<th>BM25 score</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>25</td>
<td>20.66</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>12.74</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>5.00</td>
</tr>
<tr>
<td>1</td>
<td>25</td>
<td>18.2</td>
</tr>
<tr>
<td>0</td>
<td>25</td>
<td>15.66</td>
</tr>
</tbody>
</table>
Query Likelihood Discussion

- Simple probabilistic retrieval model
- Uses probability estimations as term weights
- QL with Dirichlet smoothing similar to BM25
- QL with advanced smoothing consistently better than BM25
  - Advanced smoothing: Use only similar documents instead of entire collection. Later...

- Disadvantages
  - Difficult to incorporate information about relevant documents into ranking
  - Difficult to represent the fact that a query is just one of many possible queries to describe a particular need
Relevance Models

- Represent topic of query as language model
  - Call this the *relevance model* – language model representing information need
  - Query: Very small sample generated from this model
  - Relevant documents: Larger samples from same 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
    - Large and extremely variable number of words
    - Difficulties comparing across documents of different lengths
      - $|D_a| = 5$; $|D_b| = 500$
      - $P(D_a|R)$ and $P(D_b|R)$ vs. $P(Q|D_a)$ and $P(Q|D_b)$
    - Difficult to obtain relevance model (examples for relevant documents)
Pseudo-Relevance Feedback

■ Idea:

1. Estimate relevance model from query and top-ranked documents.
2. 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 $P$,

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

- Divergence: Large values mean large difference, mean low similarity.
- $KL(P||Q) \geq 0$
- Not symmetric: $KL(P||Q) \neq KL(Q||P)$
  - Choice of “true” distribution is important.

- Use negative KL-divergence for ranking, and assume relevance model $R$ is the true distribution:

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

- Summation over all words in vocabulary

$$\log_a \frac{x}{y} = \log_a x - \log_a y$$
\[
\sum_{w \in V} P(w|R) \log P(w|D) - \sum_{w \in V} P(w|R) \log P(w|R)
\]

- Second term same for each document: Ignore for ranking
- Given a simple maximum likelihood estimate for \( P(w|R) \), based on the frequency in the query text, ranking score is
  \[
  \sum_{w \in V} \frac{f_{w,Q}}{|Q|} \log P(w|D)
  \]
  - This is rank-equivalent to query likelihood score.
    - Non-query words are iterated but contribute zero.
    - Query words with frequency \( k \) contribute \( k \) times \( \log P(w|D) \).
- Query likelihood model is a special case of retrieval based on relevance model
  - More general model allows more sophisticated estimation based on other query words. Now...
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

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

- $P(q_1, \ldots, q_n)$ is normalizing constant

\[
P(q_1 \ldots q_n) = \sum_{w \in V} P(w, q_1 \ldots q_n)
\]

- Now: Estimate $P(w,q_1, \ldots, q_n)$
Estimating the Relevance Model

- Given document set $C$ represented by language models, joint probability is

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

- Assume

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

- Gives

$$P(w, q_1 \ldots q_n) = \sum_{D \in C} P(D)P(w \mid D) \prod_{i=1}^{n} P(q_i \mid D)$$
Estimating the Relevance Model

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

- \(P(D)\) usually assumed to be uniform: Ignore
- \(\prod_{i=1}^{n} P(q_i|D)\) is query likelihood score for \(D\).
  - Thus, \(P(w, q_1 \ldots q_n)\) 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.

- We are adding words to query by smoothing relevance model using documents that are similar to query.
- This is precisely a formal model for pseudo-relevance feedback
  - Used as query expansion technique: Words with zero weight in relevance model will now have non-zero weights
Pseudo-Feedback Algorithm

1. Rank documents using the query likelihood score for query $Q$.
   - Use Dirichlet-smoothing for $P(w|D)$

2. Select number of the top-ranked documents to be the set $C$.
   - Using entire collection including low-ranked documents would not be helpful. Also: Faster calculation

3. Calculate the relevance model probabilities $P(w|R)$ using
   \[ P(w|R) \approx \frac{P(w,q_1 \ldots q_n)}{P(q_1 \ldots q_n)} \]
   - $P(q_1 \ldots q_n)$ is used as a normalizing constant and is calculated as before as
   \[ P(q_1 \ldots q_n) = \sum_{w \in V} P(w,q_1 \ldots q_n) \]

4. Rank documents again using the KL-divergence score:
   - Use Dirichlet-smoothing for $P(w|D)$
   - Iterate only over highest-probability words for efficiency

\[
\sum_{w} P(w|R) \log P(w|D)
\]
Example from Top 10 Docs

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

*16 highest-probability words from relevance model*

Strong focus on source type (news) This will reflect results of pseudo-relevance feedback
Example from Top 50 Docs

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

16 highest-probability words from relevance model

More general, because larger variety of topics in documents
Overview

- Older models
- Probabilistic models
- Language models
- Combining evidence
- Web search
- (Learning to Rank)
Effective retrieval requires the combination of many pieces of evidence about a document’s potential relevance.

- Until now: focus on simple word-based evidence
- Many other types of evidence
  - Words: Structure, proximity of word, relationships among words
  - Metadata: PageRank, publication date, document type
  - Scores from different models

Variant 1: Adapt BM25 or Query Likelihood with additional factors
- Difficult to maintain, understand and tune

Variant 2: Inference network model is one approach to combining evidence
- Probabilistic model
- Uses Bayesian network formalism
- Mechanism to define and evaluate operators in a query language
  - Operators to specify evidence
  - Operators to combine evidence
Bayesian Networks

- Probabilistic model
- Specifies set of events and dependencies between them
- Modeled as DAG – directed acyclic graph
  - Nodes: Events
    - Here: Observing a particular document or piece of evidence or some combination of evidences
    - All binary
  - Arcs: probabilistic dependencies between events
Inference Network

- Document node
- Evidence about location
- Evidence about document features (terms, proximity)
- Query nodes $q_i$ combine evidence
- One language model for each significant document structure (title, body, heading)
- Information need node $I$
- Representation nodes $r_i$
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 \(\theta\) estimated using the parameters \(\mu\)
  - 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
    - AND, OR, ...

- **Information need node** \((I)\) is a special query node that combines all of the evidence from the other query nodes
  - In all, network computes \(P(I|D, \mu)\)
  - = probability that an information need is met given the document and the parameters \(\mu_x\)
  - Used to rank documents
Inference Network

- Connections in inference network defined by query and by representation nodes
- Probabilities for representation nodes estimated using relevance model
  - Reflect probability that feature is characteristic of document
    - Not probability of occurrence
  - Node for "lincoln" represents binary event that document is about that topic.
  - Relevance model used to calculate probability that that event is TRUE.
- Document is represented by binary vector
To calculate probabilities:

\[ P(r_i|D, \mu) = \frac{f_{r_i,D} + \mu P(r_i|C)}{|D| + \mu} \]

- Same as before – Dirichlet smoothing
- \( f_{i,D} \) is number of times feature \( r_i \) occurs in \( D \)
- \( P(r_i|C) \) is collection probability for feature \( r_i \)
- \( \mu \) is Dirichlet smoothing parameter
  - Specific to the document structure of interest

Example: \( f_{i,D} \) is number of times „lincoln“ appears in title
- Collection probability calculated based on all collection titles
- \( \mu \) is title-specific
Example: AND Combination

- Query nodes are basis for operators of query language
  - Restricted to combinations that can be efficiently calculated
  - Calculate probability of each outcome (true or false) given all possible states of parent nodes
- Example for Boolean AND:

\[
\begin{align*}
\text{a} & \quad \text{b} \\
\downarrow & \quad \downarrow \\
q &
\end{align*}
\]

\(a\) and \(b\) are parent nodes for \(q\)

\[
\begin{array}{|c|c|c|}
\hline
P(q = \text{TRUE}|a, b) & a & b \\
\hline
0 & \text{FALSE} & \text{FALSE} \\
0 & \text{FALSE} & \text{TRUE} \\
0 & \text{TRUE} & \text{FALSE} \\
1 & \text{TRUE} & \text{TRUE} \\
\hline
\end{array}
\]
Example: AND Combination

- Combination must consider all possible states of parents
- Some combinations can be computed efficiently
- Let $p_{xy}$ denote probability that $q$ is TRUE given state $x$ and $y$ of parents.
  - $p_a$ is probability that $a$ is TRUE
- Calculate belief value (probability) from an AND combination:

$$bel_{\text{and}}(q) = p_{00} P(a = \text{FALSE})P(b = \text{FALSE}) + p_{01} P(a = \text{FALSE})P(b = \text{TRUE}) + p_{10} P(a = \text{TRUE})P(b = \text{FALSE}) + p_{11} P(a = \text{TRUE})P(b = \text{TRUE})$$

$$= 0 \cdot (1 - p_a)(1 - p_b) + 0 \cdot (1 - p_a)p_b + 0 \cdot p_a(1 - p_b) + 1 \cdot p_ap_b$$

$$= p_ap_b$$
Inference Network Operators

- Other operators can also be calculated efficiently.
- Let $q$ have $n$ parents,
  - each with probability $p_i$ of being true.

\[
\begin{align*}
  \text{bel}_{\text{not}}(q) &= 1 - p_1 \\
  \text{bel}_{\text{or}}(q) &= 1 - \prod_{i=1}^{n} (1 - p_i) \\
  \text{bel}_{\text{and}}(q) &= \prod_{i=1}^{n} p_i \\
  \text{bel}_{\text{wand}}(q) &= \prod_{i=1}^{n} p_i^{wt_i} \\
  \text{bel}_{\text{max}}(q) &= \max\{p_1, p_2, \ldots, p_n\} \\
  \text{bel}_{\text{sum}}(q) &= \frac{\sum_{i=1}^{n} p_i}{n} \\
  \text{bel}_{\text{wsum}}(q) &= \frac{\sum_{i=1}^{n} wt_i p_i}{\sum_{i=1}^{n} wt_i}
\end{align*}
\]

$wt_i$ is weight of parent to indicate relative importance
Galago Query Language

- Given description of underlying model and combination operators, define a query language that can be used internally in a search engine to produce rankings based on complex combinations of evidence.
- Example here: Galago (galagosearch.org, developed by authors of textbook)
- Query: "pet therapy" compiled to Galago 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 #od1(pet therapy).(anchor)
      1.0 #od1(pet therapy).(title)
      3.0 #od1(pet therapy).(body)
      1.0 #od1(pet therapy).(heading))
  0.1 #weight(
    1.0 #uw8(pet therapy).(anchor)
    1.0 #uw8(pet therapy).(title)
    3.0 #uw8(pet therapy).(body)
    1.0 #uw8(pet therapy).(heading)))
```
A document is viewed as a sequence of text that may contain arbitrary tags.

- HTML tags, XML tags

A single *context* is generated for each unique tag name \( T \).

- All text and tags that appear within tags of type \( T \).
  - Examples: \(<\text{body}>, \:<\text{title}>, \:<\text{h1}>, \ldots\)
- Context may be nested.
- Terms can appear in multiple contexts.
- Tags used beyond mere structure: Entity / feature extraction

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 – Terms

- Term is basic building block
  - Corresponds to representation nodes in inference network
- Large variety of terms defined
  - Simple, ordered phrase, synonym, ...

- 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

- $\#N(...)$
  - 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.

- $\#uwN(...)$
  - 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:
- $\#1(\text{white house})$ – matches “white house” as an exact phrase.
- $\#2(\text{white house})$ – matches “white * house” (where * is any word or null).
- $\#uw2(\text{white house})$ – matches “white house” and “house white”.

Felix Naumann | Search Engines | Sommer 2011
Galago Query Language – Synonyms

- **#syn( ... )**
  - Treat all listed terms as synonyms

- **#wsyn( ... )**
  - Treat all listed terms as synonyms
  - Allows assignment of weights

**Examples:**
- #syn(dog canine) – simple synonym based on two terms.
- #syn( #1(united states) #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.
#any(.)

- Used to match extent types

Examples:

- #any(PERSON) – matches any occurrence of a person extent.
- #1(lincoln died in #any(DATE)) – matches exact phrases of the form: “lincoln died in <date>...</date>”. 
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 relevance model for the document: Probability of occurrence for dog based on number of times word occurs in title field, normalized for number of words in title. Smoothing using only title fields in collection.
- #1(abraham lincoln).person.(header) - builds a relevance model from all of the “header” text in the document and evaluates #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

- Used to combine evidence
- Weights can specify relative importance of evidence.
- `#combine(...)`
  - Normalized version of the $bel_{and}(q)$ operator in the inference network model.
- `#weight(...)`
  - Normalized version of the $bel_{wand}(q)$ operator.
- `#filter(...)`
  - Similar to `#combine`, but all terms (simple, proximity, synonym, etc.) are evaluated without smoothing. Document must contain at least one instance of the term.
- \#combine( \#syn(dog canine) training )
  - Rank by two terms, one of which is a synonym.

- \#combine( biography \#syn(
    \#1(president lincoln) \#1(abraham lincoln)) )
  - Rank using two terms, one of which is a synonym of “president lincoln” and “abraham lincoln”.

- \#weight( 1.0 \#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 rank them according to the query \#combine(tropical fish).

- \#filter( \#weight( 2.0 europe 1.0 travel)
    \#1(john smith).author )
  - Rank documents about “europe” and “travel” that have “John Smith” in the author context.
Overview

- Older models
- Probabilistic models
- Language models
- Combining evidence
- Web search
- Learning to Rank
Web Search

- Retrieval models in practice
  - Web search most important, but not only, search application
- Major differences to TREC news
  - Size of collection
    - Billions
  - Connections between documents
    - Links
  - Range of document types
  - Importance of spam
  - Volume of queries
    - Tens of millions per day
  - Range of query types
    - Informational, navigational, transactional
**Search Taxonomy**

- **Informational**
  - Finding information about some topic that 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
  - Known-item search

- **Transactional**
  - Finding a site where a task such as shopping or downloading music can be performed
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
  - “Age”, how often it is updated
  - URL of the page
  - Domain name of its site
  - Amount of text content
- Anchor text
- Links (e.g., PageRank)
- User behavior (click logs)
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
  - 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
  - Improve the anchor text and link structure
- Some of these techniques are regarded as not appropriate by search engine companies
In TREC evaluations, most effective features for navigational search are:

- Text in the title, body, headings (h1, h2, h3, and h4)
- Anchor text of all links pointing to the document
- PageRank number and inlink count

Given size of Web, many pages will contain all query terms

- Search engines can use AND semantics
  - Dangerous for smaller collections
    - Site search, news search, ...
    - TREC: Only 50% of relevant pages contain all search terms
- Ranking algorithm focuses on discriminating between these pages
- Term proximity is important.
Term Proximity

- Assumption: Query terms are likely to appear in close proximity within relevant documents
  - “Green party political views”
- 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
  - Let $S_Q$ be the set of all non-empty subsets of $Q$ (power set)
    - Every $s \in S_Q$ that consists of contiguous query terms is likely to appear as an exact phrase in a relevant document
      - Represented using the #1 operator
    - Every $s \in S_Q$ such that $|s| > 1$ is likely to appear (ordered or unordered) within a reasonably sized window of text in a relevant document
      - Represented as #uw8 for $|s| = 2$ and #uw12 for $|s| = 3$
Term Proximity

- Example query „embryonic stem cells“
  - \( S_Q = \{ \text{embryonic}, \text{stem}, \text{cells}, \text{embryonic stem}, \text{stem cells}, \text{embryonic cells}, \text{embryonic stem cells} \} \)
- Compiled to Galago query
  - \[
  \text{#weight(}
  \begin{align*}
  &0.8 \text{ #combine( embryonic stem cells )} \\
  &0.1 \text{ #combine( } \\
  &\quad \text{#od1(stem cells)} \\
  &\quad \text{#od1(embryonic stem)} \\
  &\quad \text{#od1(embryonic stem cells)} \\
  &0.1 \text{ #combine( } \\
  &\quad \text{#uw8(stem cells)} \\
  &\quad \text{#uw8(embryonic cells)} \\
  &\quad \text{#uw8(embryonic stem)} \\
  &\quad \text{#uw12(embryonic stem cells) } \\
  \end{align*}
  \)
  \]

Ordered window, distance 1
Unordered window, distance 8
Example Web Query

- Query: “pet therapy“
- Compiled to Galago query

```plaintext
#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 #od1(pet therapy).(anchor)
      1.0 #od1(pet therapy).(title)
      3.0 #od1(pet therapy).(body)
      1.0 #od1(pet therapy).(heading))
    0.1 #weight(
      1.0 #uw8(pet therapy).(anchor)
      1.0 #uw8(pet therapy).(title)
      3.0 #uw8(pet therapy).(body)
      1.0 #uw8(pet therapy).(heading))
))
```

PageRank and inlinks calculated at index time

Proximity can be indexed, but increases index size
Query types

- Insights gained from TREC experiments
- Topical search:
  - Simple terms and proximity features suffice
- Navigational search:
  - More evidence is helpful
- Pseudo-relevance feedback
  - Helps topical search
  - Is detrimental for navigational search
- But: How can we determine query type?

- Other evidence is in general useful
  - User behavior: Clicked-on pages, dwell time, links followed
- But: How to weight and combine more and more evidence?
  - Idea: Machine learning
Overview

- Older models
- Probabilistic models
- Language models
- Combining evidence
- Web search
- Learning to Rank
Considerable interaction between these fields

- Rocchio algorithm (60s) is a simple learning approach
  \[ q'_j = \alpha \cdot q_j + \beta \cdot \frac{1}{|Rel|} \sum_{D_i \in Rel} d_{ij} - \gamma \cdot \frac{1}{|Nonrel|} \sum_{D_i \in Nonrel} d_{ij} \]
- 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 probabilistic retrieval models presented so far fall into the category of *generative models*.
  - Assume that documents were generated from some underlying model
  - Use 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 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
- 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 $\rightarrow$ $(d_3, d_1)$ and $(d_3, d_2)$ will be in desired ranking for this query
Summary

- Best retrieval model depends on application and data available
- Evaluation corpus (or test collection), training data, and user data are all critical resources.
- Language resources (e.g., thesaurus) can make a big difference