Boolean and Vector Space Retrieval Models

- 2013 CS 290N
- Some of slides from R. Mooney (UTexas), J. Ghosh (UT ECE), D. Lee (USTHK).
Table of Content

- Boolean model
- Statistical vector space model
Retrieval Models

• A retrieval model specifies the details of:
  - Document representation
  - Query representation
  - Retrieval function: how to find relevant results

• Determines a notion of relevance.
  - Notion of relevance can be binary or continuous
Classes of Retrieval Models

• Boolean models (set theoretic)
  ▪ Extended Boolean

• Vector space models (statistical/algebraic)
  ▪ Generalized VS
  ▪ Latent Semantic Indexing

• Probabilistic models
Retrieval Tasks

- **Ad hoc retrieval**: Fixed document corpus, varied queries.
- **Filtering**: Fixed query, continuous document stream.
  - User Profile: A model of relative static preferences.
  - Binary decision of relevant/not-relevant.
- **Routing**: Same as filtering but continuously supply ranked lists rather than binary filtering.
Common Preprocessing Steps

- Strip unwanted characters/markup (e.g. HTML tags, punctuation, numbers, etc.).
- Break into tokens (keywords) on whitespace.
- Possibly use stemming and remove common stopwords (e.g. a, the, it, etc.).
- Detect common phrases (possibly using a domain specific dictionary).
- Build inverted index (keyword → list of docs containing it).
Boolean Model

• A document is represented as a set of keywords.
• Queries are Boolean expressions of keywords, connected by AND, OR, and NOT, including the use of brackets to indicate scope.
  ▪ [[Rio & Brazil] | [Hilo & Hawaii]] & hotel & !Hilton]
• Output: Document is relevant or not. No partial matches or ranking.
Boolean Retrieval Model

• Popular retrieval model because:
  ▪ Easy to understand for simple queries.
  ▪ Clean formalism.

• Boolean models can be extended to include ranking.

• Reasonably efficient implementations possible for normal queries.
Boolean Models – Problems

• Very rigid: AND means all; OR means any.
• Difficult to express complex user requests.
• Difficult to control the number of documents retrieved.
  ▪ All matched documents will be returned.
• Difficult to rank output.
  ▪ All matched documents logically satisfy the query.
• Difficult to perform relevance feedback.
  ▪ If a document is identified by the user as relevant or irrelevant, how should the query be modified?
Example

- Which plays of Shakespeare contain the words *Brutus AND Caesar* but *NOT Calpurnia*?

- Could *grep* all of Shakespeare’s plays for *Brutus* and *Caesar*, then strip out lines containing *Calpurnia*?
  - Slow (for large corpora)
  - *NOT Calpurnia* is non-trivial
  - Other operations (e.g., find the phrase *Romans and countrymen*) not feasible
## Term-document incidence

<table>
<thead>
<tr>
<th>Term</th>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brutus</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>worser</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

1 if play contains word, 0 otherwise
Incidence vectors

- So we have a 0/1 vector for each term.
- To answer query: take the vectors for Brutus, Caesar and Calpurnia (complemented) $\Rightarrow$ bitwise AND.
- $110100 \text{ AND } 110111 \text{ AND } 101111 = 100100$. 
Inverted index

- For each term $T$, must store a list of all documents that contain $T$.

What happens if the word Caesar is added to document 14?
Inverted index

- Linked lists generally preferred to arrays
  - Dynamic space allocation
  - Insertion of terms into documents easy
  - Space overhead of pointers

Dictionary

Postings

Sorted by docID (more later on why).
Inverted index construction

Documents to be indexed.

Token stream.

More on these later.

Modified tokens.

Inverted index.

Friends, Romans, countrymen.

More on these later.

Inverted index.

Documents to be indexed.

Token stream.

More on these later.

Modified tokens.

Inverted index.

Friends, Romans, countrymen.
The index we just built

- **How do we process a query?**
  - What kinds of queries can we process?
- **Which terms in a doc do we index?**
  - All words or only “important” ones?
- **Stopword list: terms that are so common**
  - they MAY BE ignored for indexing.
  - *e.g.*, *the, a, an, of, to …*
  - language-specific.
  - May have to be included for general web search
Query processing

Consider processing the query:

Brutus AND Caesar

- Locate Brutus in the Dictionary;
  - Retrieve its postings.
- Locate Caesar in the Dictionary;
  - Retrieve its postings.
- “Merge” the two postings:
The merge

- Walk through the two postings simultaneously, in time linear in the total number of postings entries

If the list lengths are m and n, the merge takes $O(m+n)$ operations.
Crucial: postings sorted by docID.
Boolean queries: Exact match

• Queries using *AND, OR and NOT* together with query terms
  ▪ Views each document as a set of words
  ▪ Is precise: document matches condition or not.
• Primary commercial retrieval tool for 3 decades.
• Professional searchers (e.g., Lawyers) still like Boolean queries:
  ▪ You know exactly what you’re getting.
Example: WestLaw  http://www.westlaw.com/

- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992)
- About 7 terabytes of data; 700,000 users
- Majority of users *still* use boolean queries
- Example query:
  - What is the statute of limitations in cases involving the federal tort claims act?
  - LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM
- Long, precise queries; proximity operators; incrementally developed; not like web search
More general merges

• **Exercise**: Adapt the merge for the queries:

  *Brutus AND NOT Caesar*
  *Brutus OR NOT Caesar*

  Can we still run through the merge in time $O(m+n)$?
Statistical Models

• A document is typically represented by a bag of words (unordered words with frequencies).
• Bag = set that allows multiple occurrences of the same element.
• User specifies a set of desired terms with optional weights:
  ▪ Weighted query terms:
    Q = < database 0.5; text 0.8; information 0.2 >
  ▪ Unweighted query terms:
    Q = < database; text; information >
  ▪ No Boolean conditions specified in the query.
Statistical Retrieval

- Retrieval based on *similarity* between query and documents.
- Output documents are ranked according to similarity to query.
- Similarity based on occurrence *frequencies* of keywords in query and document.
- Automatic relevance feedback can be supported:
  - Relevant documents “added” to query.
  - Irrelevant documents “subtracted” from query.
The Vector-Space Model

• Assume \( t \) distinct terms remain after preprocessing; call them index terms or the vocabulary.

• These “orthogonal” terms form a vector space.

  \[
  \text{Dimension} = t = |\text{vocabulary}|
  \]

• Each term, \( i \), in a document or query, \( j \), is given a real-valued weight, \( w_{ij} \).

• Both documents and queries are expressed as \( t \)-dimensional vectors:

  \[
  d_j = (w_{1j}, w_{2j}, \ldots, w_{tj})
  \]
Document Collection

• A collection of $n$ documents can be represented in the vector space model by a term-document matrix.

• An entry in the matrix corresponds to the “weight” of a term in the document; zero means the term has no significance in the document or it simply doesn’t exist in the document.

\[
\begin{array}{cccc}
T_1 & T_2 & \ldots & T_t \\
D_1 & w_{11} & w_{21} & \ldots & w_{t1} \\
D_2 & w_{12} & w_{22} & \ldots & w_{t2} \\
\vdots & \vdots & \vdots & & \vdots \\
D_n & w_{1n} & w_{2n} & \ldots & w_{tn}
\end{array}
\]
Example:

\[ D_1 = 2T_1 + 3T_2 + 5T_3 \]
\[ D_2 = 3T_1 + 7T_2 + T_3 \]
\[ Q = 0T_1 + 0T_2 + 2T_3 \]

- Is \( D_1 \) or \( D_2 \) more similar to \( Q \)?
- How to measure the degree of similarity? Distance? Angle? Projection?
Issues for Vector Space Model

• How to determine important words in a document?
  ▪ Word n-grams (and phrases, idioms, …) → terms

• How to determine the degree of importance of a term within a document and within the entire collection?

• How to determine the degree of similarity between a document and the query?

• In the case of the web, what is a collection and what are the effects of links, formatting information, etc.?
Term Weights: Term Frequency

• More frequent terms in a document are more important, i.e. more indicative of the topic.

\[ f_{ij} = \text{frequency of term } i \text{ in document } j \]

• May want to normalize term frequency (tf) across the entire corpus:

\[ tf_{ij} = f_{ij} / \max\{f_{ij}\} \]
Term Weights: Inverse Document Frequency

- Terms that appear in many *different* documents are *less* indicative of overall topic.

\[ df_i = \text{document frequency of term } i \]

\[ = \text{number of documents containing term } i \]

\[ idf_i = \text{inverse document frequency of term } i, \]

\[ = \log_2 (N/ df_i) \]

(N: total number of documents)

- An indication of a term’s *discrimination* power.
- Log used to dampen the effect relative to *tf.*
TF-IDF Weighting

• A typical combined term importance indicator is *tf-idf weighting*:

\[ w_{ij} = tf_{ij} \cdot idf_i = tf_{ij} \log_2 \left( \frac{N}{df_i} \right) \]

• A term occurring frequently in the document but rarely in the rest of the collection is given high weight.

• Many other ways of determining term weights have been proposed.

• Experimentally, *tf-idf* has been found to work well.
Computing TF-IDF -- An Example

Given a document with term frequencies:

\[ \text{A}(3), \text{B}(2), \text{C}(1) \]

Assume collection contains 10,000 documents and
document frequencies of these terms are:

\[ \text{A}(50), \text{B}(1300), \text{C}(250) \]

Then:

\[ \begin{align*}
\text{A}: & \quad \text{tf} = \frac{3}{3}; \quad \text{idf} = \log\left(\frac{10000}{50}\right) = 5.3; \quad \text{tf-idf} = 5.3 \\
\text{B}: & \quad \text{tf} = \frac{2}{3}; \quad \text{idf} = \log\left(\frac{10000}{1300}\right) = 2.0; \quad \text{tf-idf} = 1.3 \\
\text{C}: & \quad \text{tf} = \frac{1}{3}; \quad \text{idf} = \log\left(\frac{10000}{250}\right) = 3.7; \quad \text{tf-idf} = 1.2
\end{align*} \]
A similarity measure is a function that computes the degree of similarity between two vectors.

Using a similarity measure between the query and each document:

- It is possible to rank the retrieved documents in the order of presumed relevance.
- It is possible to enforce a certain threshold so that the size of the retrieved set can be controlled.
Similarity Measure - Inner Product

- Similarity between vectors for the document $d_j$ and query $q$ can be computed as the vector inner product:

$$\text{sim}(d_j, q) = d_j \cdot q = \sum_{i=1}^{t} w_{ij} \cdot w_{iq}$$

where $w_{ij}$ is the weight of term $i$ in document $j$ and $w_{iq}$ is the weight of term $i$ in the query.

- For binary vectors, the inner product is the number of matched query terms in the document (size of intersection).

- For weighted term vectors, it is the sum of the products of the weights of the matched terms.
Properties of Inner Product

• The inner product is unbounded.

• Favors long documents with a large number of unique terms.

• Measures how many terms matched but not how many terms are not matched.
Inner Product -- Examples

Binary:
- D = 1, 1, 1, 0, 1, 1, 0
- Q = 1, 0, 1, 0, 0, 1, 1

\[ \text{sim}(D, Q) = 3 \]

Weighted:

\[ D_1 = 2T_1 + 3T_2 + 5T_3 \quad D_2 = 3T_1 + 7T_2 + 1T_3 \]
\[ Q = 0T_1 + 0T_2 + 2T_3 \]

\[ \text{sim}(D_1, Q) = 2*0 + 3*0 + 5*2 = 10 \]
\[ \text{sim}(D_2, Q) = 3*0 + 7*0 + 1*2 = 2 \]
Cosine Similarity Measure

- Cosine similarity measures the cosine of the angle between two vectors.
- Inner product normalized by the vector lengths.

\[
\text{CosSim}(d_j, q) = \frac{\vec{d}_j \cdot \vec{q}}{|\vec{d}_j| \cdot |\vec{q}|} = \frac{\sum_{i=1}^{t} (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2} \cdot \sqrt{\sum_{i=1}^{t} w_{iq}^2}}
\]

\[
D_1 = 2T_1 + 3T_2 + 5T_3 \quad \text{CosSim}(D_1, Q) = \frac{10}{\sqrt{(4+9+25)(0+0+4)}} = 0.81
\]
\[
D_2 = 3T_1 + 7T_2 + 1T_3 \quad \text{CosSim}(D_2, Q) = \frac{2}{\sqrt{(9+49+1)(0+0+4)}} = 0.13
\]
\[
Q = 0T_1 + 0T_2 + 2T_3
\]

\(D_1\) is 6 times better than \(D_2\) using cosine similarity but only 5 times better using inner product.
Comments on Vector Space Models

• Simple, mathematically based approach.
• Considers both local ($tf$) and global ($idf$) word occurrence frequencies.
• Provides partial matching and ranked results.
• Tends to work quite well in practice despite obvious weaknesses.
• Allows efficient implementation for large document collections.
Problems with Vector Space Model

• Missing semantic information (e.g. word sense).
• Missing syntactic information (e.g. phrase structure, word order, proximity information).
• Assumption of term independence (e.g. ignores synonomy).
• Lacks the control of a Boolean model (e.g., requiring a term to appear in a document).
  ▪ Given a two-term query “A B”, may prefer a document containing A frequently but not B, over a document that contains both A and B, but both less frequently.