Information Retrieval and Web Search

Cornelia Caragea

Computer Science
University of Illinois at Chicago

Credits for slides: Mooney

Retrieval Models

Required Reading

- "Information Retrieval" textbook
  - Chapter 1: Boolean retrieval
  - Chapter 6: Scoring, term weighting & the vector space model

Classes of Retrieval Models

- Exact match
  - Boolean models (set theoretic)
- Ranking - “Best” match
  - Vector space models (algebraic)
  - Probabilistic models

Exact vs. Best Match

- Exact-match
  - Query specifies precise retrieval criteria
  - Every document either matches or fails to match query
  - Result is a set of documents
    - Unordered in pure exact match
- Best-match
  - Query describes good or "best" matching document
  - Every document matches query to some degree
  - Result is a ranked list of documents

Exact-match Pros & Cons

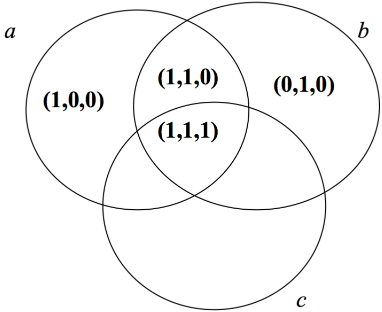
- Advantages of exact match
  - Can be very efficiently implemented
  - Predictable, easy to explain
  - Structured queries for pinpointing precise documents - very expressive
  - Works well when you know exactly (or roughly) what the collection contains and what you are looking for
- Disadvantages of exact match
  - Query formulation difficult for most users
  - Difficulty increases with collection size
  - The indexing vocabulary must be the same as query vocabulary
  - Ranking models are consistently shown to be better

Boolean Retrieval Models

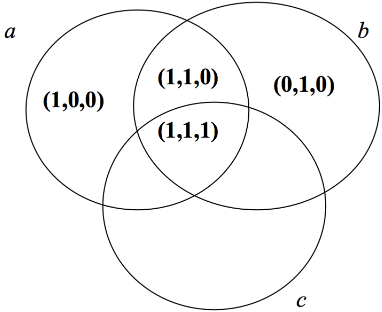
- Exact-match models
  - Simple models based on set theory
  - Neat formalism, precise semantic $q = a \wedge (b \vee \neg c)$
  - Queries are logic expressions with document features as operands, and specify precise relevance criteria
  - The models retrieve documents iff they satisfy a Boolean expression
  - Documents are returned in no particular order
- Supported operators (query language)
  - Logical operators: AND, OR, NOT
  - Most systems support simple regular expressions as search terms to match spelling variants
    - colou?r
    - ab*c
    - ab+c

Boolean Model

- Consider
  - $q = a \wedge (b \vee \neg c)$



Boolean Model



- Consider
    - $q = a \wedge (b \vee \neg c)$

- Result: $(1,1,1) \vee (1,1,0) \vee (1,0,0)$

Drawbacks of the Boolean Model

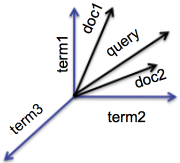
- Retrieval is based on binary decision criteria with no notion of partial matching
- No ranking of the documents is provided (absence of a grading scale)
- Information need has to be translated into a Boolean expression which most users find awkward
- The Boolean queries formulated by the users are most often too simplistic

Best-Match Retrieval

- Best-match or ranking models are more common
- Advantages:
  - Significantly more effective than exact match
  - Easier to use (supports full text queries)
- Disadvantages:
  - Efficiency is always less than exact match (cannot reject documents early)
- Boolean or structured queries can be part of a best-match retrieval model

Vector Space Models

- Key idea: Everything (documents, queries, terms) is a vector in a high-dimensional space.



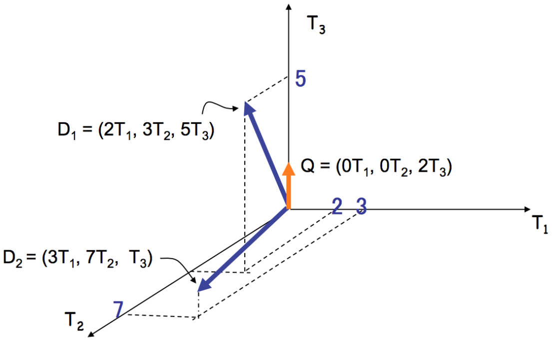
- The geometry of space induces a similarity measure between documents
- The documents are ranked based on their similarity with the query
- History:
  - Invented by Gerald Salton (1960/70)

Issues for Vector Space Models

- How to determine important words in a document?
  - How to select basis vectors (dimensions)
- How to convert objects into vectors?
  - Documents, queries, terms
- Assumption - not all terms are equally useful for representing the document contents, less frequent terms allow identifying a narrower set of documents
  - The importance of the index terms is represented by weights associated to them.
  - How to determine the degree of importance of a term within a document and within the entire collection?
- How to compare objects in the vector space?
  - How to determine the degree of similarity between a document and the query?
- In the case of the web, what is a collection?

Example Graphical Representation

- $D_1 = (2T_1, 3T_2, 5T_3)$
- $D_2 = (3T_1, 7T_2, 1T_3)$
- $Q = (0T_1, 0T_2, 2T_3)$



- Is D_1 or D_2 more similar to Q?
- How to measure the degree of similarity? Distance? Angle?

The Vector-Space Model

- Assume t distinct terms remain after preprocessing; call them index terms or the vocabulary.
- These "orthogonal" terms form a basis of a vector space. 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:
  $d_j = (w_{1j}, w_{2j}, \cdots, w_{tj})$

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 does not exist in the document.

$$\begin{array}{cc} & \begin{matrix} T_1 & T_2 & \cdots & T_t \end{matrix} \\ \begin{matrix} D_1 \\ D_2 \\ \vdots \\ \vdots \\ D_n \end{matrix} & \begin{pmatrix} w_{11} & w_{21} & \cdots & w_{t1} \\ w_{12} & w_{22} & \cdots & w_{t2} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ w_{1n} & w_{2n} & \cdots & w_{tn} \end{pmatrix} \end{array}$$

Term Weights: Term Frequency

- More frequent terms in a document are more important, i.e. more indicative of the topic.
  f_{ij} = frequency of term i in document j
- May want to normalize *term frequency (tf)*
  - e.g. by dividing by the frequency of the most common term in the document:

$$tf_{ij} = \frac{f_{ij}}{max_i\{f_{ij}\}}$$

Term Weights: Inverse Document Frequency

- Terms that appear in many *different* documents are less indicative of the overall topic.
  - df_i = document frequency of term i = number of documents containing term i
  - idf_i = 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:

$$w_{ij} = tf_{ij} idf_i = tf_{ij} \log_2(N/df_i)$$

- A term occurring frequently in the document but rarely in the rest of the collection is given high weight.
- Experimentally, *tf-idf* has been found to work well.

Computing *tf-idf* - An Example

- Given a document containing terms with given frequencies:

$$A(3), B(2), C(1)$$

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

$$A(50), B(1300), C(250)$$

- Compute *tf*, *idf*, *tf-idf*?

$$w_{ij} = tf_{ij} idf_i = (f_{ij}/max_i\{f_{ij}\}) \cdot \log_2(N/df_i)$$

Computing *tf-idf* - An Example

- Given a document containing terms with given frequencies:

$$A(3), B(2), C(1)$$

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

$$A(50), B(1300), C(250)$$

- Then:

$$A : tf = 3/3; idf = \log_2(10000/50) = 7.6; tf\text{-}idf = 7.6$$

$$B : tf = 2/3; idf = \log_2(10000/1300) = 2.9; tf\text{-}idf = 2.0$$

$$C : tf = 1/3; idf = \log_2(10000/250) = 5.3; tf\text{-}idf = 1.8$$

Query Vector

- Query vector is typically treated as a document and is also *tf-idf* weighted.
- The alternative is for the user to supply weights for the given query terms.
    - Weighted query terms:
      $Q = <$ database 0.5; text 0.8; information 0.2 $>$
    - Unweighted query terms:
      $Q = <$ database; text; information $>$

Similarity Measure

- 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.

Desiderata for Proximity

- If d_1 is near d_2, then d_2 is near d_1.
- If d_1 near d_2, and d_2 near d_3, then d_1 is not far from d_3.
- No document is closer to d than d itself.

Vector Space Similarity: Common Measures

Sim(X,Y)	Binary Term Vectors	Weighted Term Vectors
Inner product	$\lvert X \cap Y \rvert$	$\sum x_i \cdot y_i$
Dice coefficient	$\frac{2\lvert X \cap Y \rvert}{\lvert X \rvert + \lvert Y \rvert}$	$\frac{2\sum x_i \cdot y_i}{\sum x_i^2 + \sum y_i^2}$
Cosine coefficient	$\frac{\lvert X \cap Y \rvert}{\sqrt{\lvert X \rvert}\sqrt{\lvert Y \rvert}}$	$\frac{\sum x_i \cdot y_i}{\sqrt{\sum x_i^2 \cdot \sum y_i^2}}$
Jaccard coefficient	$\frac{\lvert X \cap Y \rvert}{\lvert X \rvert + \lvert Y \rvert - \lvert X \cap Y \rvert}$	$\frac{\sum x_i \cdot y_i}{\sum x_i^2 + \sum y_i^2 - \sum x_i \cdot y_i}$

Inner Product

- Similarity between vectors for the document d_j and query q can be computed as the vector inner product (or the dot product):

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

  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.

Inner Product - Examples

- Binary:

	retrieval	database	architecture	computer	text	management	information
D =	1,	1,	1,	0,	1,	1,	0
Q =	1,	0 ,	1,	0,	0,	1,	1

  Size of vector = size of vocabulary = 7;
  0 means corresponding term not found in document or query
  $sim(D, Q) = ?$

- Weighted:

$$D_1 = (2T_1, 3T_2, 5T_3), D_2 = (3T_1, 7T_2, 1T_3),$$

$$Q = (0T_1, 0T_2, 2T_3)$$

  $sim(D_1, Q) = ?$
  $sim(D_2, Q) = ?$

Inner Product - Examples

- Binary:
  Size of vector = size of vocabulary = 7; 0 means corresponding term not found in document or query
  $sim(D,Q) = 3$

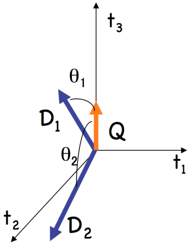
	retrieval	database	architecture	computer	text	management	information
D =	1,	1,	1,	0,	1,	1,	0
Q =	1,	0 ,	1,	0,	0,	1,	1

- Weighted:

$$D_1 = (2T_1, 3T_2, 5T_3), D_2 = (3T_1, 7T_2, 1T_3),$$

$$Q = (0T_1, 0T_2, 2T_3)$$

$sim(D_1, Q) = 2 \cdot 0 + 3 \cdot 0 + 5 \cdot 2 = 10$
$sim(D_2, Q) = 3 \cdot 0 + 7 \cdot 0 + 1 \cdot 2 = 2$



Cosine Similarity Measure

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

$$CosSim(d_j, q) = \frac{\langle d_j, q \rangle}{\|d_j\| \cdot \|q\|} = \frac{\sum_{i=1}^{t} w_{ij} w_{iq}}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$

$$D_1 = (2T_1, 3T_2, 5T_3), D_2 = (3T_1, 7T_2, 1T_3),$$

$$Q = (0T_1, 0T_2, 2T_3)$$

$CosSim(D_1, Q) = ?$
$CosSim(D_2, Q) = ?$

Cosine Similarity Measure

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

$$CosSim(d_j, q) = \frac{\langle d_j, q \rangle}{\|d_j\| \cdot \|q\|} = \frac{\sum_{i=1}^{t} w_{ij} w_{iq}}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$

$$D_1 = (2T_1, 3T_2, 5T_3), D_2 = (3T_1, 7T_2, 1T_3),$$

$$Q = (0T_1, 0T_2, 2T_3)$$

$$CosSim(D_1, Q) = 10/\sqrt{(4+9+25)(0+0+4)} = 0.81$$
$$CosSim(D_2, Q) = 2/\sqrt{(9+49+1)(0+0+4)} = 0.13$$

D_1 is 6 times better than D_2 using cosine similarity but only 5 times better using inner product.

Vector Space Summary

- Very simple
    - Map everything to a vector
    - Compare using angle between vectors
- Challenge is mostly finding good weighting scheme
    - Variants on *tf-idf* are most common
- Another challenge is comparison function
    - Cosine comparison is most common
    - Generic inner product (without unit vectors) also occurs
- 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.

Problems with Vector Space Model

- Missing semantic information (e.g. word sense).
- Missing syntactic information (e.g. phrase structure, word order, proximity information).
- 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
- Implementation?

Naïve Implementation

- Convert all documents in collection $\mathcal{D}$ to *tf-idf* weighted vectors d_j for keyword vocabulary V.
- Convert query to a *tf-idf*-weighted vector q.
- For each d_j in $\mathcal{D}$ do
  - Compute score $s_j = CosSim(d_j, q)$
- Sort documents by decreasing score.
- Present top ranked documents to the user.

Time complexity?

Naïve Implementation

- Convert all documents in collection $\mathcal{D}$ to tf-idf weighted vectors d_j for keyword vocabulary V.
- Convert query to a tf-idf-weighted vector q.
- For each d_j in $\mathcal{D}$ do
  - Compute score $s_j = CosSim(d_j, q)$
- Sort documents by decreasing score.
- Present top ranked documents to the user.

Time complexity: O(|V|·|D|) Bad for large V & D !
|V| = 10,000; |D| = 100,000; |V|·|D| = 1,000,000,000

Practical Implementation

- Based on the observation that documents containing none of the query keywords do not affect the final ranking
- Try to identify only those documents that contain at least one query keyword
- Actual implementation of an inverted index