CIS 833 – Information Retrieval and Text Mining

Lecture 6

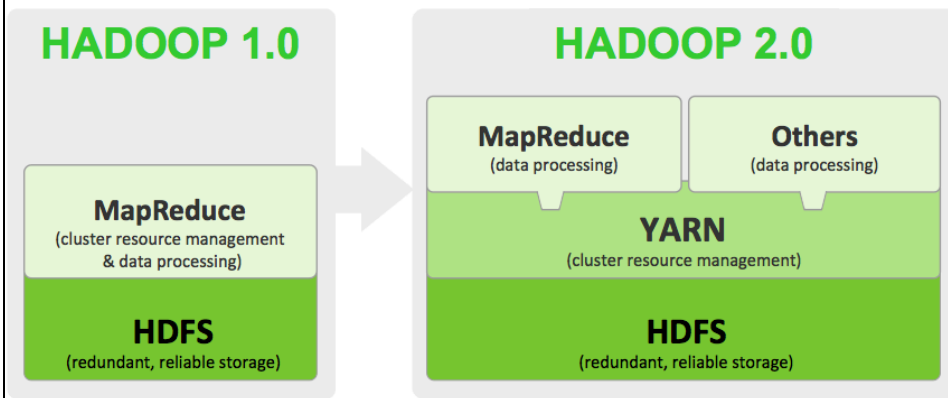
Retrieval Models

September 10, 2015

Credits for slides: Hofmann, Mihalcea, Mobasher, Mooney, Schutze.

Assignments

- HW1 was posted yesterday (due next Wed)
- The *warmup* WordCount MapReduce programming assignment will also be posted online soon (due September 23rd)



http://www.tomsitpro.com/articles/hadoop-2-vs-1,2-718.html

Required Reading

- "Information Retrieval" textbook
  - Chapter 1: Boolean Retrieval
  - Chapter 2: Term Vocabulary and Posting Lists
  - Chapter 4: Index Construction

Information Retrieval

- The processing, indexing and retrieval of textual documents.
- Concerned firstly with retrieving *relevant* documents to a query.
- Concerned secondly with retrieving from *large* sets of documents *efficiently*.

Classes of Retrieval Models

- Boolean models (set theoretic)
  - Extended Boolean
- Vector space models (algebraic)
  - Generalized VS
  - Latent Semantic Indexing
- Probabilistic models
  - Inference Networks
  - Belief Networks

Exact match

Ranking - "Best" match

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 ranked list of documents
- Popular approaches often provide some of each
  - E.g., some type of ranking of result set (best of both worlds)
  - E.g., best-match query language that incorporates exact-match operators

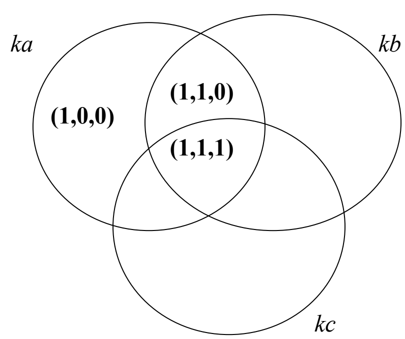
Exact-match Pros & Cons

- 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’re looking for
- Disadvantages of exact match
  - Query formulation difficult for most users
  - Difficulty increases with collection size
  - Indexing vocabulary same as query vocabulary
  - Acceptable precision generally means unacceptable recall
  - Ranking models consistently shown to be better

Boolean Retrieval Model

- Most popular exact-match model
  - simple model based on set theory
  - neat formalism, precise semantic $q = ka \wedge (kb \vee \neg kc)$
  - queries are logic expressions with document features as operands, specify precise relevance criteria
  - retrieve documents iff they satisfy a Boolean expression
  - documents returned in no particular order
- Supported operators (query language)
  - logical operators: AND, OR, NOT
  - most systems allow proximity operators: near, sentence, paragraph
  - most systems support simple regular expressions as search terms to match spelling variants

Boolean Model

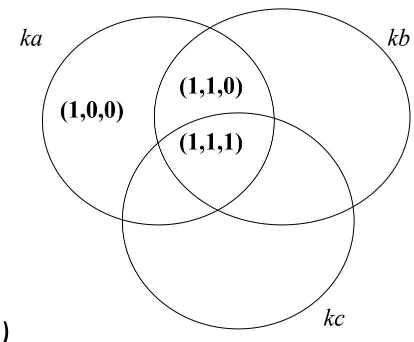


Consider

$q = ka \wedge (kb \vee \neg kc)$

Result?

Boolean Model



Consider

$q = ka \wedge (kb \vee \neg kc)$

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

Drawbacks of the Boolean Model

- Retrieval 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
- As a consequence, the Boolean model frequently returns either too few or too many documents in response to a user query

Classes of Retrieval Models

- Boolean models (set theoretic)
  - Extended Boolean
- Vector space models (algebraic)
  - Generalized VS
  - Latent Semantic Indexing
- Probabilistic models
  - Inference Networks
  - Belief Networks

Exact match

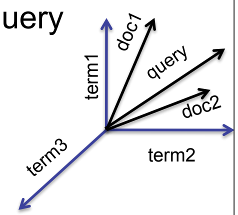
Ranking - “Best” match

Best-Match Retrieval

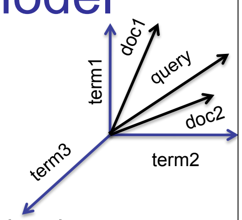
- Best-match or ranking models more common
- Advantages:
  - Significantly more effective than exact match
  - Uncertainty is a better model than certainty
  - Easier to use (supports full text queries)
- Disadvantages:
  - More difficult to convey an appropriate cognitive model (“control”)
  - Full-text does not mean natural language understanding (no “magic”)
  - 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 Model

- Key idea: Everything (documents, queries, terms) is a vector in a high-dimensional space.
- Geometry of space induces a similarity measure between documents
- Rank documents based on their similarity with query
- History:
  - Invented by Gerald Salton (1960/70)
    - utilized in SMART system (Cornell)
  - Lucene (popular open source engine written in Java)
  - Most Web search engines are similar



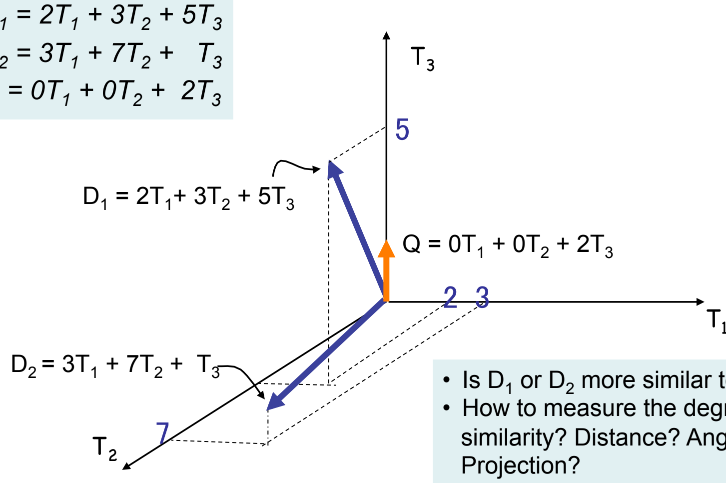
Issues for Vector Space Model



- 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 and what are the effects of links, formatting information, etc.?

Example Graphical Represenation

$D_1 = 2T_1 + 3T_2 + 5T_3$
$D_2 = 3T_1 + 7T_2 + T_3$
$Q = 0T_1 + 0T_2 + 2T_3$



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

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* = |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.

$$
\begin{array}{c} \\ D_1 \\ D_2 \\ : \\ : \\ D_n \end{array}
\begin{pmatrix}
T_1 & T_2 & \ldots & T_t \\
w_{11} & w_{21} & \ldots & w_{t1} \\
w_{12} & w_{22} & \ldots & w_{t2} \\
: & : & & : \\
: & : & & : \\
w_{1n} & w_{2n} & \ldots & w_{tn}
\end{pmatrix}
$$

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:

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

Term Weights: Inverse Document Frequency

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

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

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-idf = 7.6

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

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

Query Vector

- Query vector is typically treated as a document and also tf-idf weighted.
- 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.

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.
  - Sometimes it is a good idea to determine the maximum possible similarity as the similarity between a document d and 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 $\boldsymbol{d}_i$ and query $\boldsymbol{q}$ can be computed as the vector inner product (a.k.a. dot product):

$$\text{sim}(\boldsymbol{d}_j, \boldsymbol{q}) = \boldsymbol{d}_j \bullet \boldsymbol{q} = \sum_{i=1}^{t} w_{ij} w_{iq}$$

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.

Properties of Inner Product

- The inner product is (usually) 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:

	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:

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

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

Inner Product -- Examples

Binary:

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

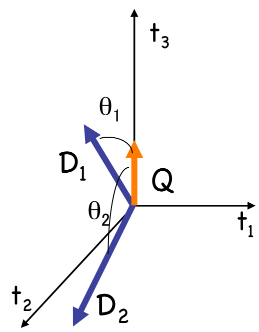
Size of vector = size of vocabulary = 7
0 means corresponding term not found in document or query

sim(D, Q) = 3

Weighted:

$D_1 = 2T_1 + 3T_2 + 5T_3$ $D_2 = 3T_1 + 7T_2 + 1T_3$
$Q = 0T_1 + 0T_2 + 2T_3$

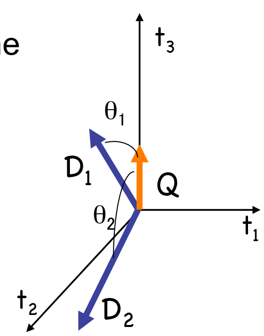
sim(D_1 , Q) = 2*0 + 3*0 + 5*2 = 10
sim(D_2 , Q) = 3*0 + 7*0 + 1*2 = 2



Cosine Similarity Measure

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

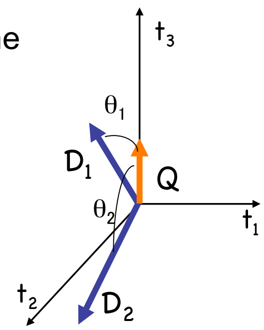
CosSim($\mathbf{d}_j$, $\mathbf{q}$) = $$\frac{\vec{d_j}\cdot\vec{q}}{\left|\vec{d_j}\right|\cdot\left|\vec{q}\right|} = \frac{\sum_{i=1}^{t}(w_{ij}\cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$



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

Cosine Similarity Measure

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



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

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

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

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).
- Assumption of term independence (e.g. ignores synonymy).
- 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 D to tf-idf weighted vectors, $\boldsymbol{d}_j$, for keyword vocabulary V.

Convert query to a tf-idf-weighted vector $\boldsymbol{q}$.

For each $\boldsymbol{d}_j$ in D do

  Compute score $s_j = \text{cosSim}(\boldsymbol{d}_j, \boldsymbol{q})$

Sort documents by decreasing score.

Present top ranked documents to the user.

Time complexity?