Boolean and Vector Space Retrieval Models

Retrieval Models

- A retrieval model specifies the details of
  - document representation
  - query representation
  - retrieval function

- Three main classes of retrieval models
  - Boolean model
  - (statistical) vector space model
  - probabilistic model

- Distinction between model and implementation:
  - A model can be implemented in many ways; the inverted file (or b-tree) is just an implementation method, not a retrieval model

Boolean Model

Or called "bag of words" or "bow"

- A document is represented as a set of keywords
  - The keywords may be preprocessed by stopword removal and stemming
  - No word order, no document structure (no title, heading, etc), single words (Hong Kong becomes {Hong, Kong}; if phrase can be detected {hongkong} or {hong-kong}, etc.)
- Queries are expressed as a Boolean expression of keywords, connected by AND, OR, and NOT, including brackets
  - E.g., DBMS AND Oracle AND NOT Sybase
  - How do you specify Boolean operators in Google or Yahoo!?
- A document is retrieved based on whether or not its keywords satisfy the Boolean query -> a YES/NO decision

Boolean Models – Strengths

- By far the most popular retrieval model because
  - easy to understand for *simple* queries
  - good control of the result set by using complex Boolean expressions
- Rather efficient implementations (using inverted file; see later lectures)
  - as far as identifying documents containing a term is concerned
  - need other supporting structure, e.g., forward index for deletion
- Users with some training can formulate *simple* Boolean queries easily
- Boolean models can be extended to include ranking (see Extended Boolean model)

Boolean Models – Problems

- Very rigid: AND means all; OR means either one
  - what about "I want *m* of these *n* words"?
- Difficult to express complex user requests
  - Boolean expressions with two or more levels of brackets are practically incomprehensible to typical users
- The meaning of AND and OR may be interpreted differently
  - In a restaurant menu, you may see "Breakfast includes two eggs, fried or scrambled; toast or biscuit, and Coffee or juice"
  - Common sense: either fried or scrambled; math sense: could be both
  - "I bought some books on computer and economics", which could mean you have books about computers and books about economics

Enhancements could be incorporated, e.g., for "A or B", rank results containing both A and B first, then results containing either A or B.
For "m out of n words", get results containing all n words first then those containing n-1 words next, etc.

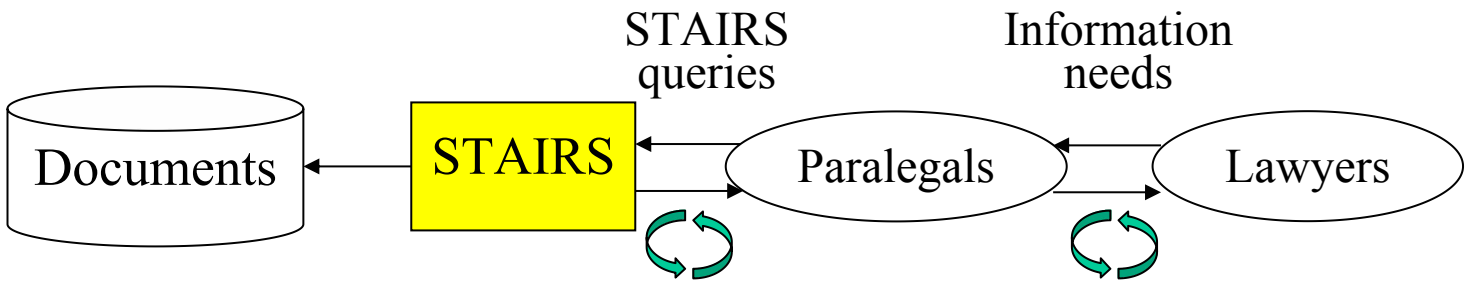
Boolean Models – Problems (Cont.)

- Difficult to control the number of documents retrieved
  - *all* matched documents in principle will be returned
  - To get fewer results: add conjunctive terms to or remove disjunctive terms (if any) from the query
  - To get more: remove conjunctive terms from or add disjunctive terms to the query
- Difficult to rank output
  - *all* matched documents satisfy the query in the same way
  - Can only rank by date, alphabetical order, etc.
- Difficult to perform automatic relevance feedback
  - if a document is identified by the user as relevant, how do I incorporate new terms into the query? AND? OR?

A Case Study in 1985

- It is an old study; we will see why we care about it
- Conducted by Blair and Maron [CACM, March 1985], professors of University of Michigan and UC Berkeley, respectively
- It was the first large-scale performance study of full-text retrieval systems using real applications and real people (may be the largest ever conducted)
  - Data: 40,000 legal documents (350,000 pages in hardcopy).
  - System: STAIRS, full-text retrieval system
    - Support Boolean queries and rank documents by the number of matches in documents
    - STAIR is a document management system from IBM in the 70's
  - Users: legal professionals (lawyers and paralegals)

Case Study: Methodology



- Original queries were given by lawyers.
- Boolean queries were formulated by paralegals with expertise in STAIRS **[Why?]**
- Interaction between lawyers and paralegals were allowed to come up with the "best" query formulations in Boolean
- Iterations continued until the lawyers were satisfied with the result
- Average *Precision* = 80% and *Recall* = 20%, respectively (i.e., 80% of the returned results are relevant, and 20% of all relevant results is returned) **[Are you satisfied with this performance?]**

Case Study: Impact

- Blair, David C., and Melvin E. Maron. "An evaluation of retrieval effectiveness for a full-text document-retrieval system." *Communications of the ACM* 28.3 (1985): 289-299.
- The paper generated a number of discussions (rare!)
  - Salton, Gerard. "Another look at automatic text-retrieval systems." *Communications of the ACM* 29.7 (1986): 648-656.
  - Again: An Evaluation of Retrieval Effectiveness for a Full-Text Document Retrieval System, CACM, Feb, 1986: 148-149 (Technical correspondence)
  - Blair, David C., and M. E. Maron. "Full-Text Information Retrieval: Further Analysis and Clarification." *Information Processing and Management* 26.3 (1990): 437-47.
  - DanRegard, and Tom Matzen A Re-- - Examination of Blair & Maron (1985), DESI V Workshop, June, 2013

Precision versus Recall

Informally:

Precision => Percentage of retrieved results are relevant to the queries (How much noise is in the retrieved results)

Recall => Percentage of relevant results that are retrieved (Have I seen all of the relevant results? How many useful results am I missing?)

When do we want high precision: ???

When do we want high recall:???

Statistical Models

- Keyword based (a document is represented by a list of keywords) plus statistical information about the keywords
  - D1= ⟨ text 1.0; retrieval 1.0; database 0.5; computer 0.8; information 0.2 ⟩
  - A keyword's weight roughly indicates the importance of the word in the document (but then what does it mean by "importance"?)
- 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
- Retrieval based on *similarity* between query and documents
  - Similarity is based on the *statistical* properties of the documents (which is often based on the occurrence frequencies of keywords in the documents)
- Outputs are ranked according to similarity

- Query and document vectors are structurally the same
  - Query and document vectors can be added, subtracted, etc.
- Facilitate automatic relevance feedback
  - Good/Bad terms in feedback document can be added to or subtracted from the original query

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

D_i = ⟨ retrieval 1.0, database 0.5, architecture 1.2, computer 0.0, text 0.8, management 0.9, information 0.0 ⟩

Suppose D_i is marked as relevant
Good query terms: database, text
Good query terms missed: architecture, retrieval, management
Bad (query) terms: information

Q′ = ⟨ retrieval 0.8, database 1, architecture 0.8, computer 0, text 1, management 0.8, information 0 ⟩

- Query term weights in revised query can be set differently; here, added query terms are given lower weights than original query terms

- Compare to statistical models, Boolean expressions are difficult to modify

- Suppose query Q_B is:

  database AND text AND information

- How can you improve Q_B based on the fact that D_i is relevant?
  - Database AND text AND architecture AND retrieval?
  - Database AND text AND architecture AND retrieval AND NOT information?
  - Database AND text AND (information OR architecture OR retrieval)?
  - (Database AND architecture) OR (text AND retrieval)?
  - … …
- Revised queries are more complex and more difficult to modify in further rounds

Problems to Solve in Statistical Models

- How to determine important words in a document?
- How to determine the degree of importance of a word *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 WWW, what is a collection and what are the effects of links, document structures, and other format information (bold, etc)?

How to determine important words in a document?

Intra-document importance of a word: Local weight

Inter-document importance of a word: Global weight

Document and Term Weights

- Document term weights are calculated using frequencies in documents (tf) and in collection (idf)

  tf_{ij} = frequency of term j in document i

  df_j = document frequency of term j

  = number of documents containing term j

  idf_j = inverse document frequency of term j

  = $\log_2 (N/ df_j)$ (N: number of documents in collection)

- Inverse document frequency -- an indication of a term's ability as a document discriminator.
- df_j / N probability that a random document contains term j

Term Weight

- A typical combined term importance indicator

$$w_{ij} = tf_{ij} \bullet idf_j = tf_{ij} \bullet \log_2 (N/ df_j)$$

- A term occurs frequently in the document but rarely in the remaining of the collection has a high weight

- $tf_{ij} \bullet idf_j$ gives a weight that is directly proportional to the number of occurrences of a term in a document

- Another example using normalized term frequency:

$$w_{ij} = (tf_{ij} / \max_l\{tf_{lj}\}) \bullet idf_j = (tf_{ij} / \max_l \{tf_{lj}\}) \bullet \log_2 (N/ df_j)$$

$\max_l\{tf_{lj}\}$ is the term frequency of the most frequent term in document j

- There are many other ways of determining term weights

Improved Term Weights

$$w_{ij} = \left(0.5 + 0.5\frac{tf_{ij}}{\max_k(tf_{jk})}\right) \bullet \log\left(1 + \frac{N}{df_j}\right)$$

- The "modified tf" term ranges from near 0.5 to 1
- The "modified idf" term ranges from log(2) to log(N+1)
  - I.e., a term appearing in all documents still has a non-zero weight

Improved Term Weights (BM25)

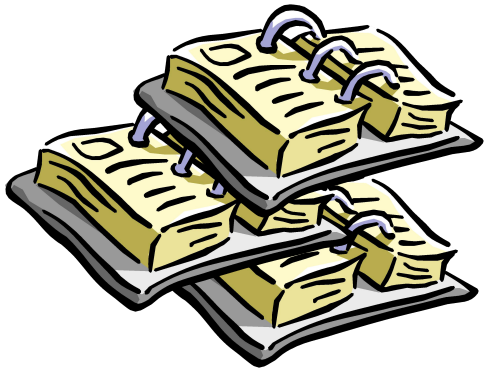
$$w_{ij} = \ln \frac{N - df_j + 0.5}{df_j + 0.5} \cdot \frac{(k_1 + 1) \cdot tf_{ij}}{\left(k_1 \cdot (1 - b) + \frac{b \cdot l_i}{l_{avg}} \right) + tf_{ij}}$$

- Known as BM25, implemented in an experimental system called OKAPI **(City University, London)**
  - k_1 and b typically set to 2.0 and 0.75, respectively
  - l_i : length of document i
  - l_{avg} : average document length in the collection
- Document length affects term weighting
  - If term weights are pre-computed: Very expensive to update
  - OK if term weights are computed dynamically

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

The Vector-Space Model

- *T* distinct terms are available; call them *index terms* or the *vocabulary*
- The index terms represent important terms for an application
  - What might be the index terms for a computer science library?



computer science collection

architecture
bus
computer
database
....
xml

index terms or *vocabulary* or *vector space* of the collection

For now: we only consider single terms (no phrases)

Representing a Vector Space

- Previously, we say a document can be represented by a list of keywords:
  - D1 = ⟨ text 1.0; retrieval 1.0; database 0.5; computer 0.8; information 0.2 ⟩ or
  - D2 = ⟨ database 1.0; architecture 0.8; retrieval 0.8; management 0.2 ⟩
- A collection of *N* documents can be represented as a document-term matrix
  - A document is a term vector
  - An entry in the matrix corresponds to the "weight" of a term in the document; zero means the term has no significance or simply doesn't exist in the document

	retrieval	database	architecture	computer	text	management	information
D1 =	1.0	0.5	0.0	0.8	1.0	0.0	0.2
D2 =	0.8	1.0	0.8	0.0	0.0	0.2	0.0

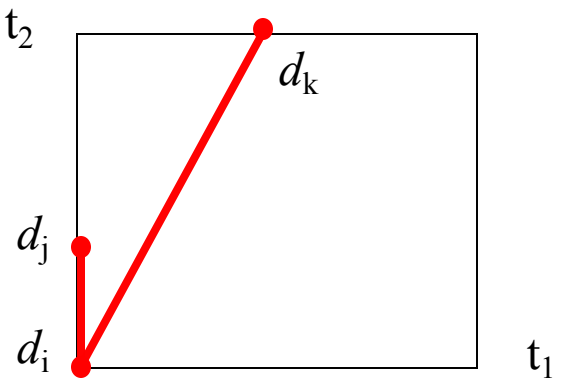
$$\begin{array}{c} \\ D_1 \\ D_2 \\ \vdots \\ \vdots \\ D_n \end{array} \begin{array}{c} \begin{array}{cccc} T_1 & T_2 & \ldots. & T_t \end{array} \\ \begin{pmatrix} d_{11} & d_{12} & \ldots & d_{1t} \\ d_{21} & d_{22} & \ldots & d_{2t} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ d_{n1} & d_{n2} & \ldots & d_{nt} \end{pmatrix} \end{array}$$

The Vector-Space Model

- A vocabulary of 2 terms forms a 2D space; a document may contain 0, 1 or 2 terms

  t_1 t_2

  - $d_i = \langle\ 0,\ \ 0\ \rangle$ (contains none of the index terms)
  - $d_j = \langle\ 0,\ \ 0.7\ \rangle$ (contains one of the two index terms)
  - $d_k = \langle\ 1,\ \ 2\ \rangle$ (contains both index terms)
- Likewise, a vocabulary of 3 terms forms a 3D space
- A vocabulary of *n* terms forms a *n*-dimensional space
- A document or query can be represented as a linear combination of *T*'s



Why do we bother about the empty document?

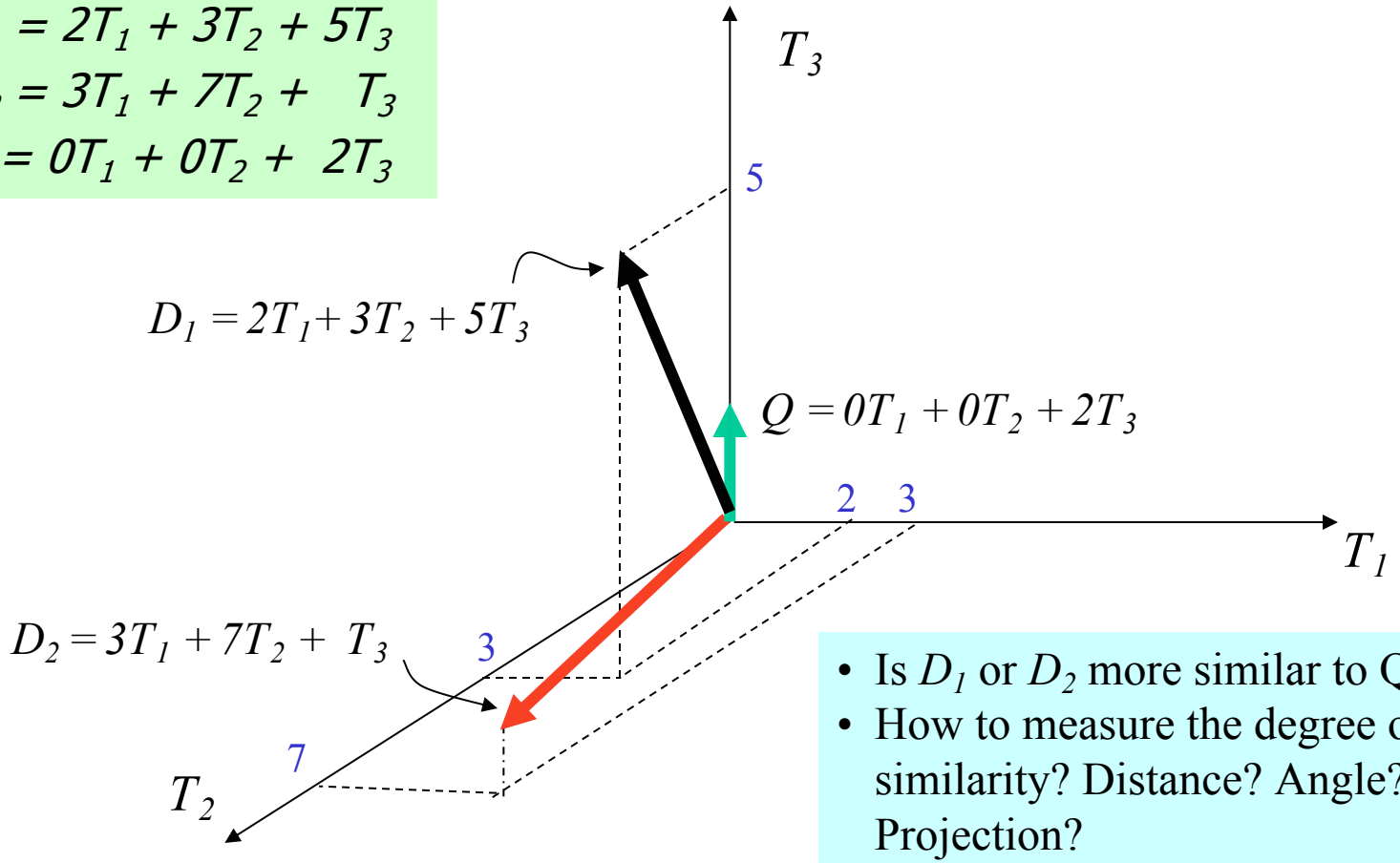
Graphic Representation

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?

Similarity Measure

- A similarity measure is a function which computes the *degree of similarity* between a pair of vectors
  - since queries and documents are both vectors, similarity can be computed between two documents, two queries, or a document and a query

- There are a large number of similarity measures proposed in the literature, because the *best* similarity measure doesn't exist (yet!)

- With similarity measure between query and documents
  - it is possible to rank the retrieved documents in the order of presumed importance
  - it is possible to enforce certain threshold so that the size of the retrieved set can be controlled
  - the results can be used to reformulate the original query in relevance feedback (e.g., combining a document vector with the query vector)

Similarity Measure - Inner Product

- Similarity between documents D_i and query Q can be computed as the inner vector product:

$$\text{sim}(D_i, Q) = \sum_{k=1}^{t} d_{ik}\, q_k$$

  - where d_{ik} is the weight of Term k in document i *and* q_k is the weight of Term k in the query
- For binary vectors, the inner product is the number of matched query terms in the document
- 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

sim(D, Q) = 3

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

Weighted:

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

$Q = 0T_1 + 0T_2 + 2T_3$ $\quad$ $Q = 0T_1 + 0T_2 + 2T_3$

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

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

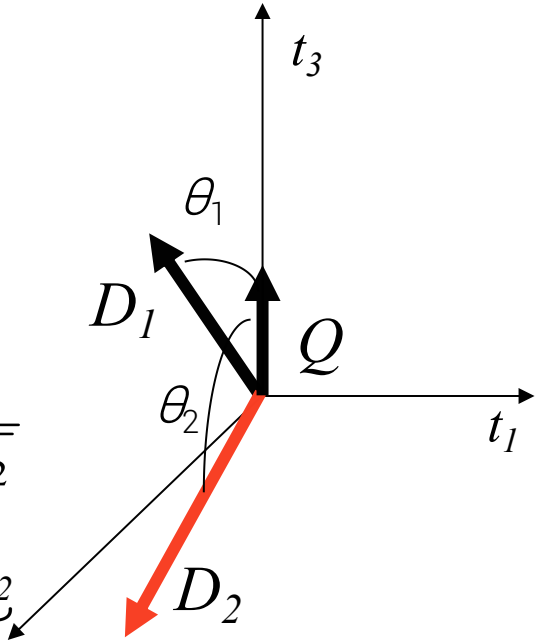
Properties of Inner Product

- The inner product similarity is unbounded
- Favors long documents
  - long document $\Rightarrow$ a large number of unique terms, each of which may occur many times
  - measures how many terms matched but not how many terms *not* matched

Cosine Similarity Measures

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

$$\text{CosSim}(D_i, Q) = \frac{D \circ Q}{|D||Q|} = \frac{\sum_{k=1}^{t} (d_{ik} q_k)}{\underbrace{\sqrt{\sum_{k=1}^{t} d_{ik}^2}}_{\text{Document length}} \underbrace{\sqrt{\sum_{k=1}^{t} q_{k_{t_2}}^2}}_{\text{Query length}}}$$



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

$\text{CosSim}(D_1, Q) = 5*2 / (\sqrt{38}\sqrt{4}) = 0.81$
$\text{CosSim}(D_2, Q) = 1*2 / (\sqrt{59}\sqrt{4}) = 0.13$

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

Jaccard Coefficient

- By Paul Jaccard, Prof of Botany and Plant Physiology, in 1901

Jaccard Coefficient: $$\frac{\sum_{k=1}^{t}(d_{ik} q_k)}{\sum_{k=1}^{t} d_{ik}^2 + \sum_{k=1}^{t} q_k^2 - \sum_{k=1}^{t}(d_{ik} q_k)}$$

$D_1 = 2T_1 + 3T_2 + 5T_3$ $\quad \text{Sim}(D_1, Q) = 10 / (38+4-10) = 10/32 = 0.31$

$D_2 = 3T_1 + 7T_2 + T_3$ $\quad \text{Sim}(D_2, Q) = 2 / (59+4-2) = 2/61 = 0.04$

$Q = 0T_1 + 0T_2 + 2T_3$

- D_1 is 9.5 times better than D_2
- Difference between Jaccard and CosSim?

Dice Coefficient

- Ranges from 0 to 1 but does not satisfy triangle inequality

Dice Coefficient:

$$\frac{2\sum_{k=1}^{t}(d_{ik}\,q_k)}{\sum_{k=1}^{t} d_{ik}^{\,2} + \sum_{k=1}^{t} q_k^{\,2}}$$

$D_1 = 2T_1 + 3T_2 + 5T_3$ $\quad \mathrm{Sim}(D_1, Q) = 2*10 / (38+4) = 20/42 = 0.48$

$D_2 = 3T_1 + 7T_2 + T_3$ $\quad \mathrm{Sim}(D_2, Q) = 2*2 / (59+4) = 4/63 = 0.06$

$Q = 0T_1 + 0T_2 + 2T_3$

- D_1 is 8 times better than D_2

Binary Versions of Similarity Measures

	Non-binary weights	Binary weights
Inner Product:	$\sum_{k=1}^{t}(d_{ik} q_k)$	$\lvert d_i \cap q_k \rvert$
Cosine:	$\frac{\sum_{k=1}^{t}(d_{ik} q_k)}{\sqrt{\sum_{k=1}^{t} d_{ik}^2 \sum_{k=1}^{t} q_k^2}}$	$\frac{\lvert d_i \cap q_k \rvert}{\sqrt{\lvert d_i \rvert \lvert q_k \rvert}}$
Jaccard :	$\frac{\sum_{k=1}^{t}(d_{ik} q_k)}{\sum_{k=1}^{t} d_{ik}^2 + \sum_{k=1}^{t} q_k^2 - \sum_{k=1}^{t}(d_{ik} q_k)}$	$\frac{\lvert d_i \cap q_k \rvert}{\lvert d_i \rvert + \lvert q_k \rvert - \lvert d_i \cap q_k \rvert} \Rightarrow \frac{\lvert d_i \cap q_k \rvert}{\lvert d_i \cup q_k \rvert}$
	d_i and q_k are vectors	d_i and q_k are sets of keywords $\lvert d_i \rvert$ or $\lvert q_k \rvert$: Number of non-zero elements in the set (Note: since the weights are binary, the square in the non-binary formula can be ignored)

Jaccard with binary weights is the intersection of the query and document divided by their union.
When $d_i = q_k$, Jaccard similarity = 1 (highest possible)
When $d_i \supset q_k$, Jaccard similarity decreases as $|d_i|$ increases (penalized long documents)

Similarity between a Query and a Document

- Given a query containing terms: 123, 345, 544, 642, 850, the corresponding DF's and the document vector
- Assume the collection contains 10,000 documents

Query

Q-ID	DF
123	50
345	540
544	1300
642	35
850	250

Document

Term-ID	TF
123	5
145	2
320	3
344	1
390	1
450	1
482	1
544	1
580	1
590	3
610	1
630	1
661	1
702	2
758	1
850	1
887	1
950	2

Term-ID	tf()	idf()	tf*idf
123	5	$\log_2(10000/50) = 7.64$	38.2
345	0	No need to calculate it!	0
544	1	$\log_2(10000/1300) = 2.94$	2.94
642	0	No need to calculate it!	0
850	1	$\log_2(10000/250) = 5.32$	5.32

Inner(D,Q) = 38.2 + 2.94 + 5.32 = 46.46

How to obtain cosine?

Document Clusters and Centroids

Comparing Documents against Documents

- Similarity can be computed:
  - Between document and query vectors ⇒ search engine (ranking)
  - Between document and document vectors ⇒ document clustering
  - Between query and query vectors ⇒ query suggestion, query expansion, etc.

$D_1 = 1, 1, 1, 0, 1, 1, 0$

$D_2 = 1, 0, 1, 0, 0, 1, 1$

$D_3 = 0.5, 2, 0, 0, 1, 1, 0.5$

- What are the similarity between these documents?
- How close are these documents?
  - Total similarity, average similarity (needs n*(n-1)/2 similarity computations)

Document Centroid Example

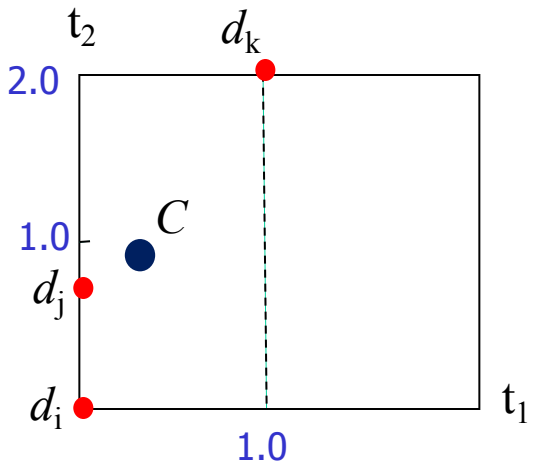
- Given three documents:

  t_1 t_2

  - $d_i = \langle 0, \quad 0 \rangle$
  - $d_j = \langle 0, \quad 0.7 \rangle$
  - $d_k = \langle 1, \quad 2 \rangle$

- The x and y coordinates of the centroid is the average of the x and y coordinates of the documents:

  C = $\langle 0.33, \quad 0.9 \rangle$

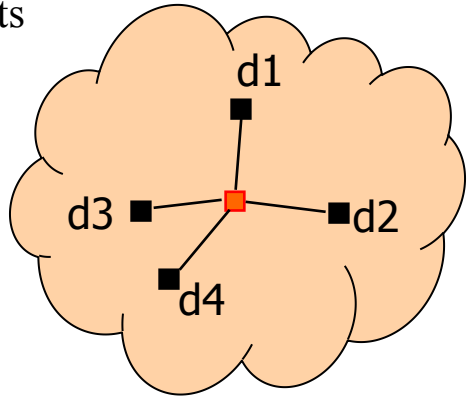


Centroid of Multiple Documents

- Given a set of *m* document vectors :

c_k: average over m documents

d1=[	$T_{1,1}$	…	$T_{1,k}$	…	…	$T_{1,t}$]
d2=[	$T_{2,1}$	…	$T_{2,k}$	…	…	$T_{2,t}$]
d3=[	$T_{3,1}$	…	$T_{3,k}$	…	…	$T_{3,t}$]
d4=[	$T_{4,1}$	…	$T_{4,k}$	…	…	$T_{4,t}$]



- The centroid of the *m* document vectors is a "virtual" document vector C:

$$C = (c_1, c_2, \ldots, c_k \ldots, c_t) \text{ where } c_k = \frac{\sum_{i=1}^{m} T_{i,k}}{m}$$

  - Centroid is a convenient and compact representation of a set of documents
  - The closeness of a document set can be measured by the distances of the documents to their centroid

Geometric Interpretation of Similarity Measures

- Cosine similarity: we all know by now that it is the angle between two vectors (query and document vectors)
- Inner product:
- What about Euclidian distance?

Choice of Similarity Measures

- The vector space model can match queries against documents, or documents against documents, but the interpretations and match criteria are different

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

- Q wants "retrieval", "architecture", "information" and "management". It doesn't care about other things.
- D meets three out of the four "wants"; match well

D_1 =	1,	1,	1,	0,	1,	1,	0
D_2 =	1,	0 ,	1,	0,	0,	1,	1

- D_1 talks about "database", "retrieval", "architecture", "text", and "management" but not about "information".
- D_2 talks about "retrieval", "architecture", "information", and "management" but not about "database" and "text".
- Are D_1 and D_2 very similar? Three matches, three mismatches, and one common absence (i.e., computer).

Similarity Measures for Documents Comparison

	Document / Document	Document / Query
Euclidean	OK	Bad
Cosine	OK	OK
Inner	Fair	OK
Jaccard	Fair+	OK

Some Issues about Similarity Measures

Query Term Weights

- The weight of a query term is assumed to be 1 if the term presents in the query; 0 otherwise
- A weight may be given by the user to each query term
- A natural language query can be regarded as a document
  - The query "MP3 songs sung by Aaron Kwok" may be transformed into:
    <MP3, song, sung, Aaron, Kwok>
  - The query "I am interested in gold medals or gold investment" may be transformed into:
    <gold 2, medal 1, investment 1> after filtering out "I am interested in"
- Similarity of two documents can be computed in the same way

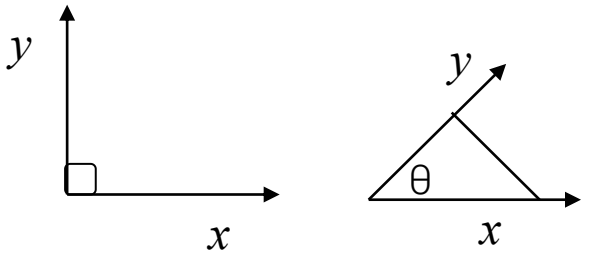
Information Abundance on Web

- Question: Is the vector space model more like 'OR' or 'AND'?
- Given the answer to the above question, do you think the vector space model is good or bad in
  - A professional database (e.g., legal or medical datasets)?
  - web retrieval?

Just type in as many relevant words as you can think of, you will find something useful from the web (Google), especially for popular topics (e.g., presidential election), since there are many sources of information on popular topics.

Term Independence Assumption (I)

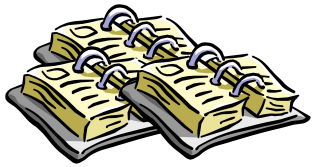
- Each term *i* is identified as T_i
- A document is represented as a set of unordered terms, called the bag-of-words (bow) representation
- The terms are assumed to be independent (or uncorrelated or orthogonal) to each other and form an orthogonal vector space
- On the left: x and y are orthogonal; y = x cos 90° = 0 * x
- On the right: x and y are not orthogonal; y = x cos θ



- What does it mean to retrieval?
  - D1 = < computer, CPU, Intel >
  - D2 = < computer, analysis, price >
  - Q = < computer >
  - Intuitively, is D1 or D2 a better match? What does vector space model tell you?
  - Should D1 be represented as < computer 2.5 > ?
- How to measure term dependence and incorporate it into retrieval?

Term Independence Assumption (II)

- In real life, it is hard to find two terms that are absolutely independent to each other
- Independence can be judged with respect to a document collection.



CS collection

"computer" "science" "business"

Important words in the collection

- In reality, these three terms are correlated to each other!
  - When you can find "computer" in a document, there is a very high chance that you also find "science" in it
  - When you can find "computer" in a document, there is a medium chance that you also find "business" in it
  - When you can find "business" in a document, there is a small chance that you also find "science" in it
- We can judge term independence by checking whether or not two terms frequently co-occur in a document (or a sentence, a window of text, etc.)

Synonyms

- Two terms have the same meaning (synonyms), more or less
  - D1 = < company, earning, invest >
  - Q = < corporation, earning >
  - Straight keyword matching will cause a mismatch between company and corporation

  - D1 = < furniture, table, desk, chair, lamp >
  - Q = < furniture >
  - Straight keyword matching will have only one matching keyword and so D1 has a small similarity to Q

Unbalanced Property of Vector Space Model

- If Q = ⟨x, y⟩, then
  - D_2 contains ⟨x, x⟩ (talk only about x but a lot of it) and D_1 contains ⟨x, y⟩ (talk about both x and y) can have the same similarity to Q

Example 1: Inner product

x y

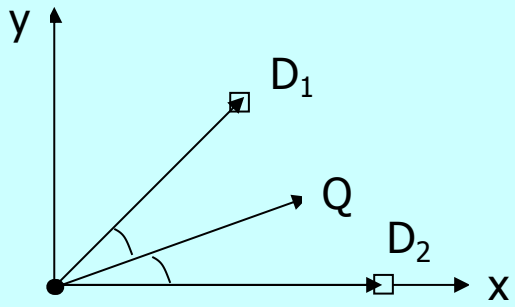
D2 = ⟨ 2, 0 ⟩

D1 = ⟨ 1, 1 ⟩

Q = ⟨ 1, 1 ⟩

Sim(Q,D1)=Sim(Q,D2)

Example 2: Cosine



Suppose

Q = ⟨ 1, 0.414 ⟩

Sim(Q,D1)=Sim(Q,D2)

What Problems does it Cause?

- What are the impacts of the unbalanced property on the following queries?
  - Education University: documents with high tf of "education" alone or "university", vs a balance of both words
  - Travel moon

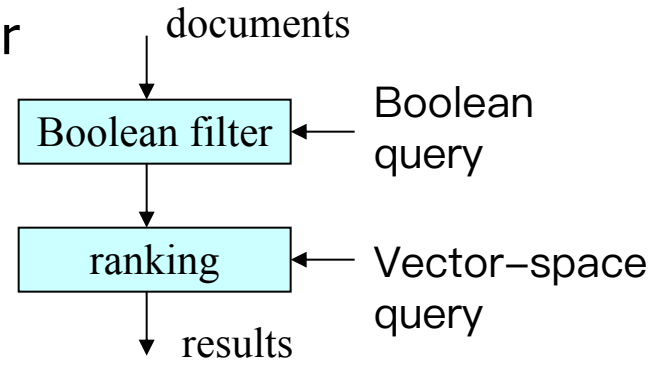
How to Favor "Balanced" Documents?

- Multi-step filtering
  - Filter out all documents containing both x and y, rank them higher
  - Filter out all documents containing either x or y, rank them lower

- Compute similarity as usual but weight the similarity by the number of query terms matched
  - $Sim'(D, Q) = Sim(D, Q) * [| D \cap Q | / | D \cup Q |]$

Boolean Models – Strengths

- Statistical model is more like the "OR" operator
- Boolean and vector space models can be combined by doing Boolean filtering first followed by ranking:
  - First evaluate the Boolean query as usual
  - collect all documents satisfying the Boolean query
  - Rank the result using the vector space query



Altavista Advanced Search: Boolean query and ranking criteria are separated
Query: gold AND investment
Ranking Criteria: precious metal

In most other systems, the vector space query is the same as the Boolean query with Boolean operators ignored

Summary

- Cosine similarity is independent of document sizes
- Based on occurrence frequencies only
- Consider both local and global occurrence frequencies
- Advantages
  - simplicity
  - able to handle weighted terms
  - easy to modify term vectors
- Disadvantages
  - assumption of term independence
  - lack the control of Boolean model (e.g., requiring a term to appear in a document); e.g., two documents $D_i = \langle x, x \rangle$ and $D_j = \langle x, y \rangle$ may yield the same similarity score for $Q = \langle x, y \rangle$ if x and y have the same weight; this may not be desirable.