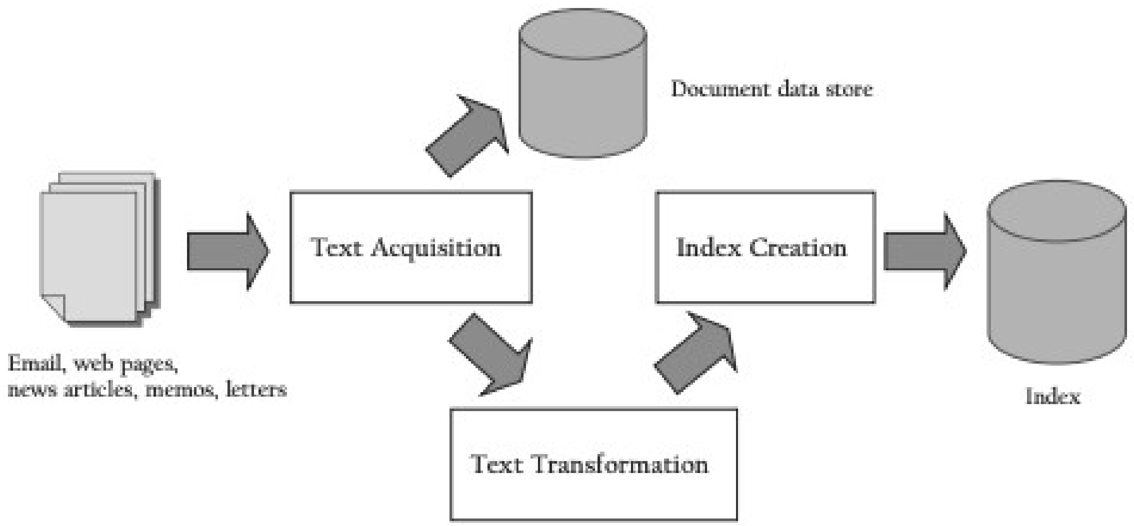
BOOLEAN AND VECTOR SPACE RETRIEVAL MODELS

Chapter 1,2 (IR Overview)

Chapter 7 (Boolean/Vector-Space Models)

Search Engine Architecture

- Search Engine has two main components:
  - ***Indexing process:*** build structures onto of the document collection to enable searching
  - ***Query process:*** uses those structures and the user query to produce a ranked list of documents.



Search Engine Architecture (Cont.)

- ***Text acquisition*** identifies the documents that will be searched.
  - This may include crawling or scanning the Web or identifying a subset of documents in a given collection.

- ***Index Creation*** creates the data structures (indexes) that enable fast searching.
  - Index creation must be efficient in terms of time and space and be efficiently updated when new documents are acquired.
  - Inverted indexes are the most common form of index used for search.

first	2205, 2265, … 5543
data	3342
school	4655, 5567
rover	2234,2235,2265
soccer	3456,6654,…, 7245
government	3651, … 9123

Search Engine Architecture (Cont.)

- ***Text transformation*** transforms documents into index terms or features.
  - Index terms can be words, phrases, names of people, dates, etc.
  - This process can involve :
    1. Removing stop words (common words from the document) since they contribute little to the description of the document content. Ex. ”for”, “to”, “and”, etc.
    2. Grouping words that are derived from a common stem (called stemming). Ex. Grouping the words “fish”, “fishes”, and “fishing” to just “fish” is just one example.
    3. Strip unwanted characters/markup (e.g. HTML tags, punctuation, numbers, etc.).

- ***Ranking*** component transforms the user’s query into terms and generates a ranked list of documents using scores based on the retrieval model.
  - Evaluation (using log data that records query execution time and user behavior) is used to evaluate the retrieval model.

Index Creation

- Index creation is the heart of the retrieval model.
- Not every term or feature should be used to build the index however, plus, we need some way to rank documents based on term importance.

- ***Document statistics :*** document statistics are stored in lookup tables
  - Records the counts of index terms in each document.
  - The position occurrence of index terms in each document,
  - Aggregated occurrence count of index terms in the document collection.

- ***Weighting :*** reflects the relative importance of words in documents which is then used in computing scores for ranking.
  - TF-IDF : Term Frequency – Inverse Document Frequency

User Query

- Usually there is a query interface that parses the user query and transforms the query to standardize and extract relevant terms.

- Query Transformation: several techniques are used to improve the initial query such as spell checking, query suggestion, stemming, stop-word removal, etc.

Ranking

- Ranking (or query processing), calculates scores for documents.
- The most basic ranking algorithm is aggregating the score of all terms in the collection.

$$\sum_i q_i d_i$$

Retrieval Models

- **Boolean Model (or Exact-Match)**
- **Vector-Space Model (ranking by query similarity)**
- Probabilistic Ranking
- PageRank (ranking using link analysis - document rank)
- Combination of various methods

What is a Retrieval Model?

- A **Retrieval Model** defines the relevance (or similarity) between a query and a document and makes it possible to rank the documents.
- In this context, we will discuss:
  - Types of retrieval models / functions
    - Boolean Retrieval
    - Vector Space Model
  - How to compute the similarity between a document and a query
    - Inner Product
    - Cosine Similarity
  - Feature weights (either binary or weighted terms)
    - Term-Frequence
    - TF-IDF

Retrieval Models

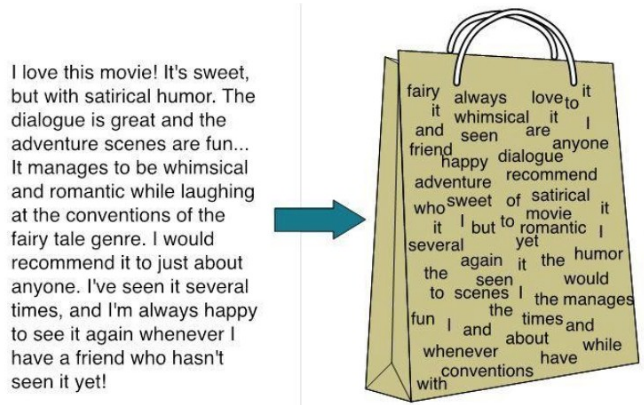
- Retrieval Models essentially ‘break’ a document and the query into parts
  - Terms, keywords, phrases, etc.

- The query and the document is represented as vectors of parts

- Then a similarity metric is applied to determine how similar is the query vector to the document vector
  - Ranking is performed based on the similarity function

How to represent a document / query?

- Simplest representation is the Bag-of-Words Model
  - Chop the document into words



  - The order of the words does not matter

Boolean Model (or Exact-Match Retrieval)

- A document is represented as a set of keywords (or features)
- 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] , or
  - "William" AND "Shakespeare" AND NOT ("Marlowe" OR "bacon")
- The output of the Boolean model is a collection of documents that are relevant
  - ***Note: this does not support partial matches or ranking***

Boolean Model (or Exact-Match Retrieval)

- Advantages
  - Popular retrieval model because it is easy to understand and explain to users.
  - Reasonably efficient implementations possible for normal queries.
- Disadvantages
  - A document is typically represented by a bag of words/terms (unordered words with frequencies) and hence the effectiveness of this approach depends on the user's query and their inclusion of good "terms".
  - Difficult to add relevance or limit number of documents returned.

Example

- Assume the following fragments comprise your document collection
- Assume the following are stopwords: **an, and, do, in, not**

Doc 1: Interest in real estate speculation

Doc 2: Interest rates and rising home costs

Doc 3: Kids do not have an interest in banking

Doc 4: Lower interest rates, hotter real estate market

Doc 5: Feds interest in raising interest rates rising

Lets work on the handout!!

	Doc1	Doc2	Doc3	Doc4	Doc5
banking					
costs					
estate					
feds					
have					
home					
hotter					
...					

Construct the term-document index

What documents are returned to query:

- interest NOT rates
- (interest AND rates) NOT (rising OR kids)

Vector Space Model

- Instead of Boolean retrieval, lets rank the document based on relevance.
  - Measures similarity, and does not assume exact match

- **Problem:** Given two text documents (query is a special type of document), how similar are they?

- First, we need to extract features or tokens from documents.

- Example
  - D_1 the cat is blue
  - D_2 the cat is blue and the dog is green
  - Q cat dog

- Can use "Bag of Words" method in which words are extracted from text and thrown into a "bag" without order.
  - So, for D_1 "the cat is blue" is indistinguishable from "blue cat is the"

Vector-Space Model

- Both documents and queries are represented by a vector of index terms.

- For example document D_i is represented by a vector of index terms.
  - $D_i = (w_{i1}, w_{i2}, w_{i3}, \ldots, w_{ij})$,
  - where w_{ij} represents the weight of the ***jth*** term
  - zero means the term has no significance or does not exist in the document.

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

- Queries are represented in the same method.

Incidence matrix (Binary Weighting)

	text	terms
d_1	The cat is green	<the, cat, is, green>
d_2	The cat is red, the dog is green	<the, car, is, red, dog, green>
Q	I want a green cat	<I, want, a , green, cat>

3 vectors in 9-dimentional term vector space

	the	is	cat	dog	red	green	I	want	a
d_1	1	1	1			1			
d_2	1	1	1	1	1	1			
Q			1			1	1	1	1

<u>Weights:</u> w_{ij} = 1 if document I contains term j and zero otherwise

What about non-binary weights? Well, we will talk about TF-IDF later

Vector-Space Model (Cont.)

- Ranking is done by *computing the distance* between each document and the query.

- A ***similarity measure*** is used (rather than a distance or dissimilarity measure), so that the documents with the highest scores are the most similar to the query.

- 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_i and query q can be computed as the vector inner product (a.k.a. dot product):

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

  - 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 unbounded.
- Favors long documents with a large number of unique terms.
- Measures how many terms matched but not how many terms are *not* matched.

Dot Product Example (Binary Weights)

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
	the	is	cat	dog	red	green	I	want	a
d_1	1	1	1			1			
d_2	1	1	1	1	1	0			
Q			1			1	1	1	1

Same applies to non-binary weights

- sim(d_1 , Q) = $d_1 \bullet Q$

  $= w_{11}w_{Q1} + w_{12}w_{Q2} + w_{13}w_{Q3} + w_{14}{}^*w_{Q4} + w_{15}{}^*w_{Q5} + w_{16}{}^*w_{Q6} + w_{17}{}^*w_{Q7} + w_{18}w_{Q8} + w_{19}w_{Q9}$

  $= 1*0 + 1*0 + 1*1 + 0*0 + 0*0 + 1*1 + 0*1 + 0*1 + 0*1 = 2$

- sim(d_1 , Q) = $d_2 \bullet Q$ =

  $= w_{21}w_{Q1} + w_{22}w_{Q2} + w_{23}w_{Q3} + w_{24}{}^*w_{Q4} + w_{25}{}^*w_{Q5} + w_{26}{}^*w_{Q6} + w_{27}{}^*w_{Q7} + w_{28}w_{Q8} + w_{29}w_{Q9}$

  $= 1*0 + 1*0 + 1*1 + 1*0 + 1*0 + 0*1 + 0*1 + 0*1 + 0*1 = 1$

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:
  - $D_1 = <2, 3, 5>$
  - $D_2 = <3, 7, 1>$
  - $Q = <0, 0, 2>$
  - $sim(D_1, Q) = 2*0 + 3*0 + 5*2 = 10$
  - $sim(D_2, Q) = 3*0 + 7*0 + 1*2 = 2$

Cosine similarity

- The cosine correlation measures the cosine of the angle between the query and the document vectors.

- Note, the vectors must be normalized so all documents are represented by vectors of equal length.

- Given normalized vectors, the cosine angle between
  - two identical vectors is 1 (the angle is zero), and
  - two vectors that don’t share any common terms, the cosine is 0 (the angle is 90).

Quick Review

- Given d = (x1, x2, x3, …, xn) is a vector in an n-dimentional vector space.

- Length of x is given by
  - $|d|^2 = x_1^2 + x_2^2 + x_3^2 + \ldots + x_n^2$
  - $|d| = \sqrt{x_1^2 + x_2^2 + x_3^2 + \ldots + x_n^2}$

- If d_1 and d_2 are document vectors:

  - Dot product is given by

    $$d_1 \cdot d_2 = w_{11} \cdot w_{21} + w_{12}w_{22} + w_{13}w_{23} + \ldots + w_{1n} * w_{2n}$$

  - Cosine angle between d1 and d2 determine doc similarity

    - $\cos(\theta) = \frac{d_1 \cdot d_2}{|d_1||d_2|}$

Cosine Similarity Measure

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

$$CosSim(d_i, Q) = \frac{d_1 \cdot d_2}{|d_1||d_2|} = \frac{\sum_{i=1}^{t}(w_{ij} w_{iQ})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \sum_{i=1}^{t} w_{iQ}^2}}$$

$D_1 = <2,3,5>$

$D_2 = <3,7,1>$

$Q = <0,0,2>$

$$CosSim(d_1, Q) = \frac{d_1 \cdot Q}{|d_1||Q|} = \frac{0*2+0*3+2*5}{\sqrt{(2^2+3^2+5^2)*(0^2+0^2+2^2)}} = 0.81$$

$$CosSim(d_2, Q) = \frac{d_2 \cdot Q}{|d_2||Q|} = \frac{0*3+0*7+2*1}{\sqrt{(3^2+7^2+1^2)*(0^2+0^2+2^2)}} = 0.13$$

D_1 is 6 times better than D_2 using cosine similarity but only 5 times better if we were to use dot product.

Term Weights: Term Frequency

- But how to compute the weight of terms?

- The term frequency component, tf, reflects the importance of a termk in a document d_i.
  - f_{kDi} = frequency of term ***k*** in document ***Di***

- This is usually computed as a normalized count of the term occurrences in a document, for example : $tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} f_{ij}}$

Where,

- tf_{ik} is the term frequency weight of term ***k*** in document D_i , and
- f_{ik} is the number of occurrences of term ***k*** in the document.

Term Weights: Inverse Document Frequency

- Terms that appear in many different documents are less indicative of overall topic, hence, term-frequency alone may not be the best weight.

- The inverse document frequency component (***idf***) reflects the importance of the term in the collection of documents.

- The more documents that a term occurs in, the less discriminating the term is between documents and, consequently, the less useful it will be in retrieval.

$$idf_k = \log \frac{N}{n_k}$$

Usually written as $1+n_k$

Where,

- Where idf_k is the inverse document frequency weight for term k,
- N is the number of documents in the collection, and
- n_k is the number of documents in which term k occurs.

- Log used to dampen the effect relative to tf.

Tf-IDF weighting

- Tf-IDF weighting is the most common term frequency weighting scheme.

$$w_{ij} = tf_{ij} * idf_i$$

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

- The effects of these two weights are combined by multiplying them (hence the name tf.idf).

- The reason for combining them this way is mostly empirical (developed by intuition and experiment).

Computing TF-IDF -- An Example

- Given a document containing terms with given frequencies:

  $tf(d) = <t_1,t_2,t_3> = <3, 2,1>$

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

  $tf(D) = <t_1,t_2,t_3> = <50, 1300,250>$

Then:

- $tf_a = 3/3$; $idf_a = \log2(10000/50) = 7.6$; tf-idf = 7.6
- $tf_b = 2/3$; $idf_b = \log2 (10000/1300) = 2.9$; tf-idf = 2.0
- $tf_c = 1/3$; $idf_c = \log2 (10000/250) = 5.3$; tf-idf = 1.8

Problems with Vector Space Model

- Advantages
  - 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.

- Disadvantages
  - 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).
  - Doesn't deal with conditions such as :
    - synonyms (ex. Car and automobile) or
    - Polysems, words that have multiple meanings (ex. Java)

Fast TF-IDF

- Assume we are computing Cosine Similarity using TF-IDF weights
- One approach
  - Traverse entries calculating the product
  - Accumulate the vector lengths and divide at the end
- But how do we do this faster when we have a very sparse representation?

Index construction: collect documentIDs

Doc 1

I did enact Julius
Caesar I was killed
i' the Capitol;
Brutus killed me.

Doc 2

So let it be with
Caesar. The noble
Brutus hath told you
Caesar was ambitious

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Index construction: sort dictionary

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

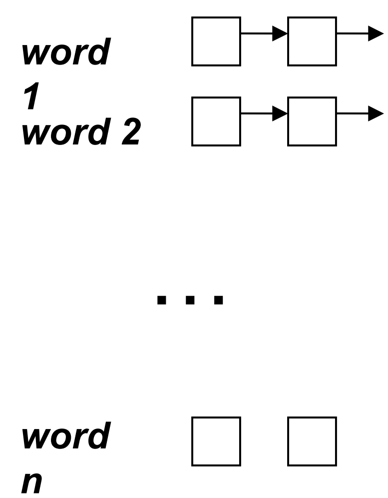
sort based on terms

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Index construction: create postings list

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

create postings lists
from identical entries



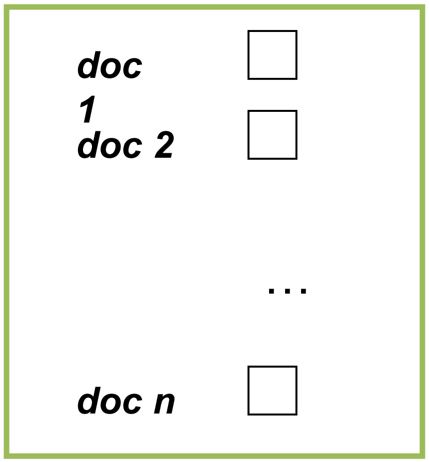
$$tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} f_{ij}}$$

$$idf_k = \log \frac{N}{n_k}$$

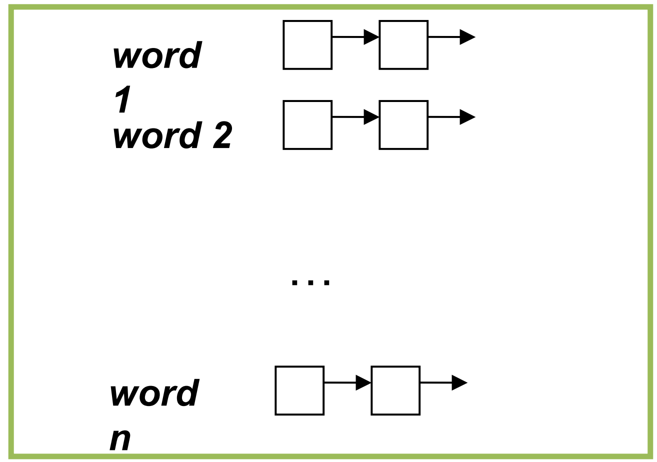
$$w_{ik} = tf_{ik} * idf_k$$

Do we have all the information we need?

Index construction: Document Length



Document Length Index



Posting List

$$tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} f_{ij}} \qquad idf_k = \log \frac{N}{n_k}$$

$$w_{ik} = tf_{ik} * idf_k$$

Computing Cosine Scores

- Function CosineScore (q):

scores[N] ← 0 (*cosine scores*)
length[N] ← *doc length list*
postings[T] ← *term frequency list*

for each query term t do
  calculate $w_{t,q}$ and fetch postings list for t
  for each pair (d, $tf_{t,d}$) in postings list do
    Scores[d] += $w_{t,d}$ * $w_{t,q}$
for each document d in Length do
  Scores[d] = Scores[d] / Length[d]

return top k components of Scores[]