Information Retrieval (IR)

Based on slides by

Prabhakar Raghavan, Hinrich Schütze,
Ray Larson

Query

- 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* is hard to do
  - Other operations (e.g., find the ***Romans*** *NEAR* ***countrymen***) not feasible

Term-document incidence

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

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) ➔ bitwise *AND*.
- 110100 *AND* 110111 *AND* 101111 = 100100.

Answers to query

- Antony and Cleopatra, Act III, Scene ii
- *Agrippa* [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,
- When Antony found Julius ***Caesar*** dead,
- He cried almost to roaring; and he wept
- When at Philippi he found ***Brutus*** slain.
- Hamlet, Act III, Scene ii
- *Lord Polonius:* I did enact Julius ***Caesar*** I was killed i' the
- Capitol; ***Brutus*** killed me.

Bigger corpora

- Consider n = 1M documents, each with about 1K terms.
- Avg 6 bytes/term incl spaces/punctuation
  - 6GB of data.
- Say there are m = 500K *distinct* terms among these.

Can’t build the matrix

- 500K x 1M matrix has half-a-trillion 0’s and 1’s.
- But it has no more than one billion 1’s. Why?
  - matrix is extremely sparse.
- What’s a better representation?

Inverted index

- Documents are parsed to extract words and these are saved with the document ID.

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

- After all documents have been parsed the inverted file is sorted by terms

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

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

- Multiple term entries in a single document are merged and frequency information added

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

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

Issues with index we just built

- How do we process a query?
- What terms in a doc do we index?
  - All words or only “important” ones?
- Stopword list: terms that are so common that they’re ignored for indexing.
  - *e.g.*, ***the, a, an, of, to*** …
  - language-specific.

Issues in what to index

Cooper’s concordance of Wordsworth was published in 1911. The applications of full-text retrieval are legion: they include résumé scanning, litigation support and searching published journals on-line.

- ***Cooper’s*** vs. ***Cooper*** vs. ***Coopers***.
- ***Full-text*** vs. ***full text*** vs. {***full, text***} vs. ***fulltext.***
- Accents: ***résumé*** vs. ***resume***.

Punctuation

- ***Ne'er***: use language-specific, handcrafted "locale" to normalize.
- ***State-of-the-art***: break up hyphenated sequence.
- ***U.S.A.*** vs. ***USA*** - use locale.
- ***a.out***

Numbers

- 3/12/91
- Mar. 12, 1991
- 55 B.C.
- B-52
- 100.2.86.144
  - Generally, don't index as text
  - Creation dates for docs

Case folding

- Reduce all letters to lower case
  - exception: upper case in mid-sentence
    - *e.g.*, ***General Motors***
    - ***Fed*** vs. ***fed***
    - ***SAIL*** vs. ***sail***

Thesauri and soundex

- Handle synonyms and homonyms
  - Hand-constructed equivalence classes
    - e.g., ***car*** = ***automobile***
    - ***your*** ≠ ***you're***
- Index such equivalences, or expand query?
  - More later ...

Spell correction

- Look for all words within (say) edit distance 3 (Insert/Delete/Replace) at query time
  - *e.g.*, ***Alanis Morisette***
- Spell correction is expensive and slows the query (up to a factor of 100)
  - Invoke only when index returns zero matches?
  - What if docs contain mis-spellings?

Lemmatization

- Reduce inflectional/variant forms to base form
- E.g.,
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be different color*

Stemming

- Reduce terms to their "roots" before indexing
  - language dependent
  - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

for example compressed and compression are both accepted as equivalent to compress. → for exampl compres and compres are both accept as equival to compres.

Porter's algorithm

- Commonest algorithm for stemming English
- Conventions + 5 phases of reductions
  - phases applied sequentially
  - each phase consists of a set of commands
  - sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*
- Porter's stemmer available: http//www.sims.berkeley.edu/~hearst/irbook/porter.html

Typical rules in Porter

- *sses* → *ss*
- *ies* → *i*
- *ational* → *ate*
- *tional* → *tion*

Beyond term search

- What about phrases?
- Proximity: Find ***Gates*** *NEAR* ***Microsoft***.
  - Need index to capture position information in docs.
- Zones in documents: Find documents with (*author* = ***Ullman***) *AND* (text contains ***automata***).

Evidence accumulation

- 1 vs. 0 occurrence of a search term
  - 2 vs. 1 occurrence
  - 3 vs. 2 occurrences, etc.
- Need term frequency information in docs

Ranking search results

- Boolean queries give inclusion or exclusion of docs.
- Need to measure proximity from query to each doc.
- Whether docs presented to user are singletons, or a group of docs covering various aspects of the query.

Test Corpora

TABLE 4.3 Common Test Corpora

Collection	NDocs	NQrys	Size (MB)	Term/Doc	Q-D RelAss
ADI	82	35			
AIT	2109	14	2	400	>10,000
CACM	3204	64	2	24.5	
CISI	1460	112	2	46.5	
Cranfield	1400	225	2	53.1	
LISA	5872	35	3		
Medline	1033	30	1		
NPL	11,429	93	3		
OSHMED	34,8566	106	400	250	16,140
Reuters	21,578	672	28	131	
TREC	740,000	200	2000	89-3543	» 100,000

Standard relevance benchmarks

- TREC - National Institute of Standards and Testing (NIST) has run large IR testbed for many years
- Reuters and other benchmark sets used
- “Retrieval tasks” specified
  - sometimes as queries
- Human experts mark, for each query and for each doc, “Relevant” or “Not relevant”
  - or at least for subset that some system returned

Sample TREC query

Topic: Tobacco company advertising and the young
Description: A document will provide information on what is a widely held opinion that the tobacco industry aims its advertising at the young.
Narrative: A relevant document must report on tobacco company advertising and its relation to young people. A relevant document can address either side of the question: (1) Do tobacco companies consciously target the young, or (2) As the tobacco industry argues, is this an erroneous public perception. The "young" may be identified as youth, children, adolescents, teenagers, high school students, and college students.

Credit: Marti Hearst

Precision and recall

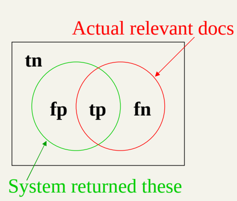
- **Precision**: fraction of retrieved docs that are relevant = P(relevant|retrieved)
- **Recall**: fraction of relevant docs that are retrieved = P(retrieved|relevant)

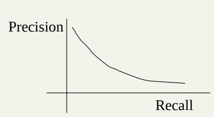
	Relevant	Not Relevant
Retrieved	tp	fp
Not Retrieved	fn	tn

- Precision P = tp/(tp + fp)
- Recall R = tp/(tp + fn)

Precision & Recall

- Precision $\frac{tp}{tp+fp}$
  - Proportion of selected items that are correct
- Recall $\frac{tp}{tp+fn}$
  - Proportion of target items that were selected
- Precision-Recall curve
  - Shows tradeoff





Precision/Recall

- Can get high recall (but low precision) by retrieving all docs on all queries!
- Recall is a non-decreasing function of the number of docs retrieved
  - Precision usually decreases (in a good system)
- Difficulties in using precision/recall
  - Binary relevance
  - Should average over large corpus/query ensembles
  - Need human relevance judgements
  - Heavily skewed by corpus/authorship

A combined measure: F

- Combined measure that assesses this tradeoff is F measure (weighted harmonic mean):

$$F = \frac{1}{\alpha \frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

- People usually use balanced F_1 measure
  - i.e., with $\beta = 1$ or $\alpha = ½$
- Harmonic mean is conservative average
  - See CJ van Rijsbergen, *Information Retrieval*

Precision-recall curves

- Evaluation of ranked results:
  - You can return any number of results ordered by similarity
  - By taking various numbers of documents (levels of recall), you can produce a *precision-recall curve*

Precision-recall curves

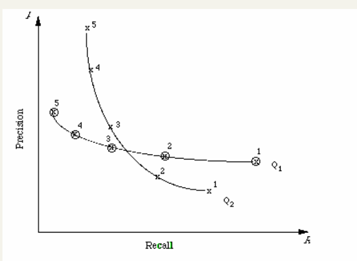


Figure 7.1. The precision-recall curves for two queries. The ordinals indicate the values of the control parameter λ

Evaluation

- There are various other measures
  - Precision at fixed recall
    - This is perhaps the most appropriate thing for web search: all people want to know is how many good matches there are in the first one or two pages of results
  - 11-point interpolated average precision
    - The standard measure in the TREC competitions: Take the precision at 11 levels of recall varying from 0 to 1 by tenths of the documents, using interpolation (the value for 0 is always interpolated!), and average them

Ranking models in IR

- Key idea:
  - *We wish to return in order the documents most likely to be useful to the searcher*
- To do this, we want to know which documents *best* satisfy a query
  - An obvious idea is that if a document talks about a topic *more* then it is a better match
- A query should then just specify terms that are relevant to the information need, without requiring that all of them must be present
  - Document relevant if it has a lot of the terms

Binary term presence matrices

- Record whether a document contains a word: document is binary vector in $\{0,1\}^v$
- Idea: Query satisfaction = overlap measure:

$$|X \cap Y|$$

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Overlap matching

- What are the problems with the overlap measure?
- It doesn't consider:
  - Term frequency in document
  - Term scarcity in collection
    - (How many documents mention term?)
  - Length of documents

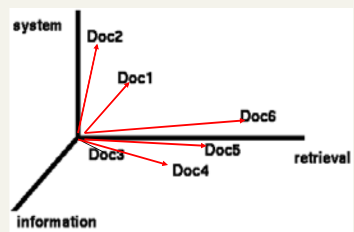
Many Overlap Measures

$\lvert Q \cap D \rvert$	Simple matching (coordination level match)
$2\frac{\lvert Q \cap D \rvert}{\lvert Q \rvert + \lvert D \rvert}$	Dice's Coefficient
$\frac{\lvert Q \cap D \rvert}{\lvert Q \cup D \rvert}$	Jaccard's Coefficient
$\frac{\lvert Q \cap D \rvert}{\lvert Q \rvert^{\frac{1}{2}} \times \lvert D \rvert^{\frac{1}{2}}}$	Cosine Coefficient
$\frac{\lvert Q \cap D \rvert}{\min(\lvert Q \rvert, \lvert D \rvert)}$	Overlap Coefficient

Documents as vectors

- Each doc *j* can be viewed as a vector of *tf* values, one component for each term
- So we have a vector space
  - terms are axes
  - docs live in this space
  - even with stemming, may have 20,000+ dimensions
- (The corpus of documents gives us a matrix, which we could also view as a vector space in which words live – transposable data)

Documents in 3D Space



Assumption: Documents that are "close together" in space are similar in meaning.

The vector space model

Query as vector:
- Regard query as *short document*
- Return the docs, ranked by distance to the query
- Easy to compute, since both query & docs are vectors.

- Developed in the SMART system (Salton, c. 1970) and standardly used by TREC participants and web IR systems

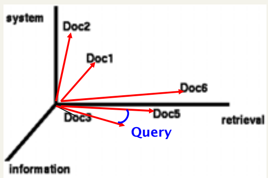
Vector Representation

- Documents & Queries represented as **vectors**.
- Position 1 corresponds to term 1, …position t to term t
- The **weight** of the term is stored in each position

$$D_i = w_{d_{i1}}, w_{d_{i2}}, \ldots, w_{d_{it}}$$
$$Q = w_{q1}, w_{q2}, \ldots, w_{qt}$$
$$w = 0 \text{ if a term is absent}$$

- Vector distance measure used to rank retrieved documents

Documents in 3D Space



Documents that are close to query (measured using vector-space metric)
=> returned first.

Document Space has High Dimensionality

- What happens beyond 2 or 3 dimensions?
  - Similarity still has to do with the number of shared tokens.
  - More terms -> harder to understand which subsets of words are shared among similar documents.
- We will look in detail at ranking methods
  - One approach to handling high dimensionality: Clustering

Word Frequency

- Which word is more indicative of document similarity?
  - 'book,' or 'Rumplestiltskin'?
  - Need to consider "document frequency"---how frequently the word appears in doc collection.
- Which doc is a better match for the query "Kangaroo"?
  - One with a single mention of Kangaroos… or a doc that mentions it 10 times?
  - Need to consider "term frequency"---how many times the word appears in the current document.

TF x IDF

$$w_{ik} = tf_{ik} * \log(N / n_k)$$

T_k = term k in document D_i

tf_{ik} = frequency of term T_k in document D_i

idf_k = inverse document frequency of term T_k in C

N = total number of documents in the collection C

n_k = the number of documents in C that contain T_k

$$idf_k = \log\left(\frac{N}{n_k}\right)$$

Inverse Document Frequency

- IDF provides high values for rare words and low values for common words

$$\log\left(\frac{10000}{10000}\right) = 0$$

$$\log\left(\frac{10000}{5000}\right) = 0.301$$

$$\log\left(\frac{10000}{20}\right) = 2.698$$

$$\log\left(\frac{10000}{1}\right) = 4$$

TF-IDF normalization

- Normalize the term weights
  - so longer docs not given more weight (fairness)
  - force all values to fall within a certain range: [0, 1]

$$w_{ik} = \frac{tf_{ik}\log(N / n_k)}{\sqrt{\sum_{k=1}^{t}(tf_{ik})^2[\log(N / n_k)]^2}}$$

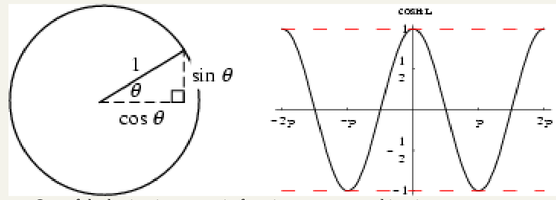
Vector space similarity
(use the weights to compare the documents)

Now, the similarity of two documents is :

$$sim(D_i, D_j) = \sum_{k=1}^{t} w_{ik} * w_{jk}$$

This is also called the cosine, or normalized inner product.
(Normalization was done when weighting the terms.)

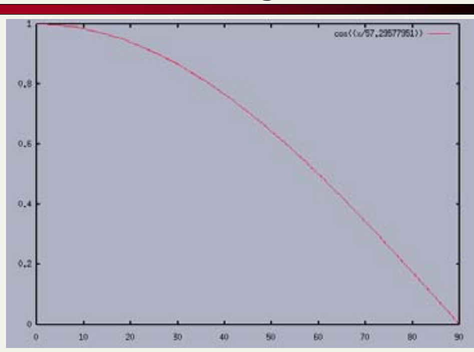
What's Cosine anyway?



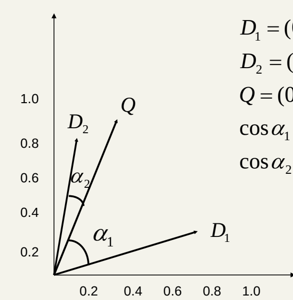
One of the basic trigonometric functions encountered in trigonometry. Let theta be an angle measured counterclockwise from the x-axis along the arc of the unit circle. Then cos(theta) is the horizontal coordinate of the arc endpoint. As a result of this definition, the cosine function is periodic with period 2pi.

From http://mathworld.wolfram.com/Cosine.html

Cosine Detail (degrees)



Computing Cosine Similarity Scores



$D_1 = (0.8, 0.3)$
$D_2 = (0.2, 0.7)$
$Q = (0.4, 0.8)$
$\cos\alpha_1 = 0.74$
$\cos\alpha_2 = 0.98$

Computing a similarity score

Say we have query vector $Q = (0.4,0.8)$
Also, document $D_2 = (0.2,0.7)$
What does their similarity comparison yield?

$$sim(Q, D_2) = \frac{(0.4*0.2)+(0.8*0.7)}{\sqrt{[(0.4)^2+(0.8)^2]*[(0.2)^2+(0.7)^2]}}$$

$$= \frac{0.64}{\sqrt{0.42}} = 0.98$$

To Think About

- How does this ranking algorithm behave?
  - Make a set of hypothetical documents consisting of terms and their weights
  - Create some hypothetical queries
  - How are the documents ranked, depending on the weights of their terms and the queries' terms?

Summary: Why use vector spaces?

- User's query treated as a (very) short document.

- Query ➔ a vector in the same space as the docs.
- Easily measure each doc's proximity to query.
- Natural measure of scores/ranking
  - No longer Boolean.