Text Processing

Text Pre-Processing

- Stemming
- Stop words removal

Stemming

- Reduce terms to their “roots” before further processing
- “Stemming” suggests crude affix chopping
  - language dependent
  - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.
- Porter Stemmer: most common algorithm for stemming English
  - Results suggest at least as good as other stemming options

Stemming

***for example compressed and compression are both accepted as equivalent to compress*.**

→

for exampl compress and compress ar both accept as equival to compress

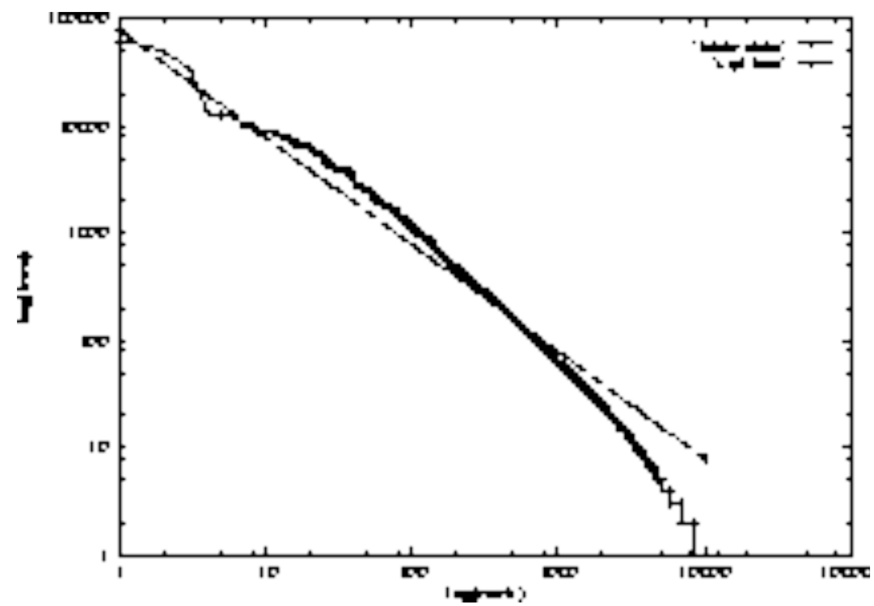
Stop words

- With a stop list, you exclude from the dictionary entirely the commonest words. Intuition:
  - They have little semantic content: *the, a, and, to, be*
  - There are a lot of them
- But the trend is away from doing this:
  - Good compression techniques means the space for including stop words in a system is very small
  - Good query optimization techniques mean you pay little at query time for including stop words.
  - You need them for:
    - Phrase queries: "King of Denmark"
    - Various song titles, etc.: "Let it be", "To be or not to be"
    - "Relational" queries: "flights to London"

Text - Inversion

- postings list – more Zipf distribution: e.g., rank-frequency plot of 'Bible'
  - The frequency of any word is roughly inversely proportional to its rank in the frequency table.
  - The most frequent word will occur approximately twice as often as the 2nd most frequent word, which occurs twice as often as the 4th most frequent word, etc.

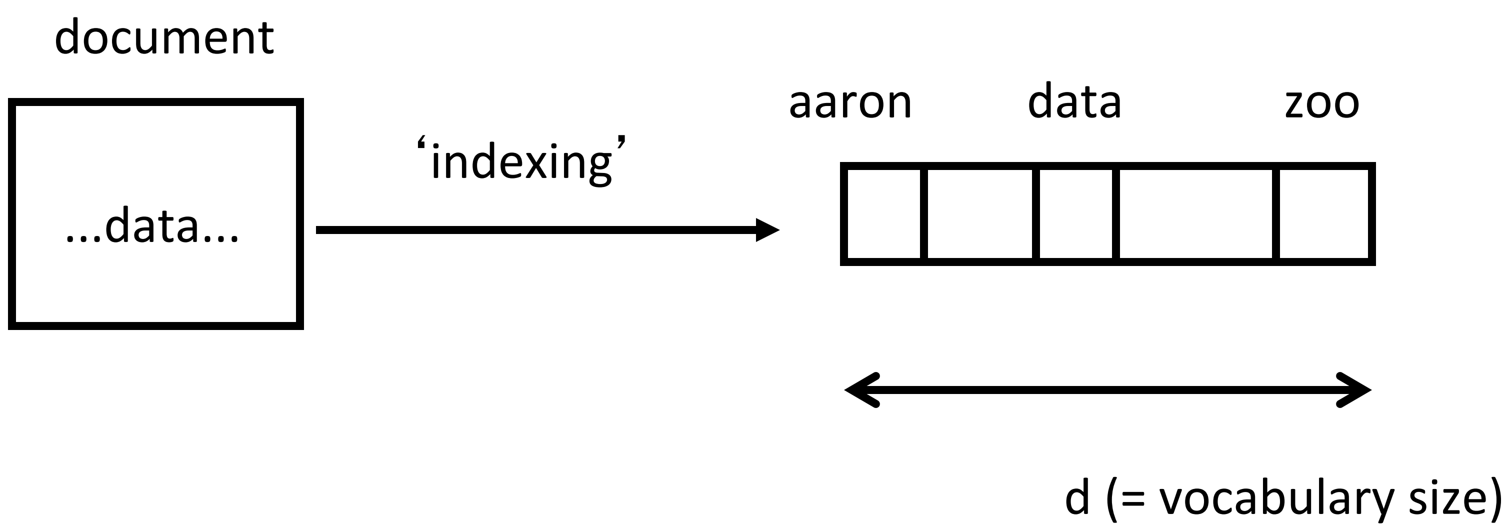
log(freq)



log(rank)

Vector Space Model

- main idea: each document is a vector of size d: d is the number of different terms in the **database**



Document Vectors

- Documents are represented as "bags of words"
- Represented as vectors when used computationally
  - A vector is like an array of floating points
  - Has direction and magnitude
  - Each vector holds a place for every term in the collection
  - Therefore, most vectors are sparse

Document Vectors
One location for each word

	nova	galaxy	heat	hollywood	film	role	diet	fur
A	10	5	3					
B								
C								
D								
E								
F								
G								
H								
I								

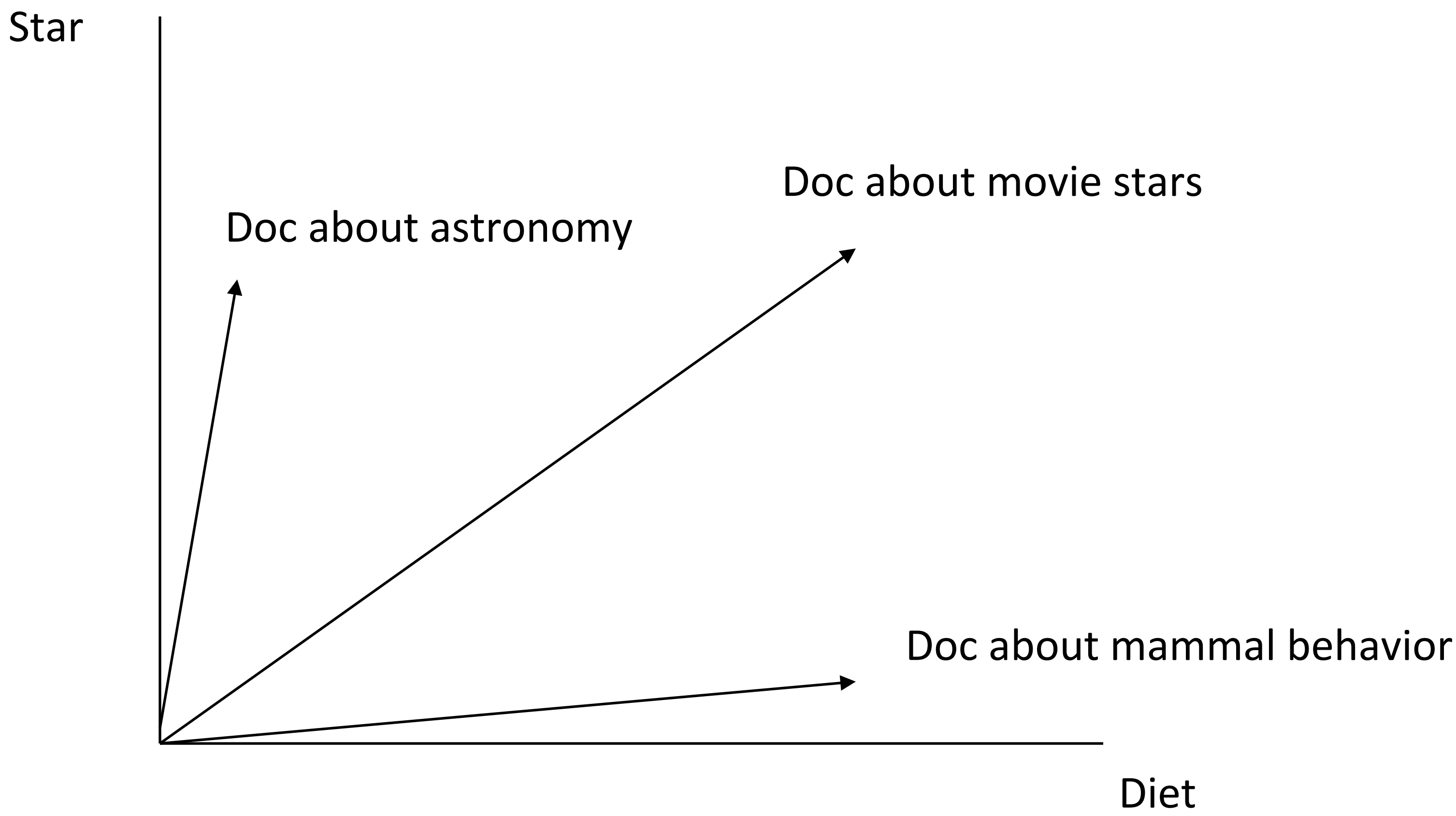
"nova" occurs 10 times in document A
"galaxy" occurs 5 times in document A
"heat" occurs 3 times in document A
(Blank means 0 occurrences.)

Document Vectors

Document ids

	nova	galaxy	heat	h'wood	film	role	diet	fur
A	10	5	3					
B	5	10						
C				10	8	7		
D				9	10	5		
E							10	10
F							9	10
G	5		7				9	
H		6	10		2	8		
I				7	5		1	3

We Can Plot the Vectors



Assigning Weights to Terms

- Binary Weights
- Raw term frequency
- tf x idf
  - Recall the Zipf distribution
  - Want to weight terms highly if they are
    - frequent in relevant documents … BUT
    - infrequent in the collection as a whole

Binary Weights

- Only the presence (1) or absence (0) of a term is included in the vector

docs	*t1*	*t2*	*t3*
D1	1	0	1
D2	1	0	0
D3	0	1	1
D4	1	0	0
D5	1	1	1
D6	1	1	0
D7	0	1	0
D8	0	1	0
D9	0	0	1
D10	0	1	1
D11	1	0	1

Raw Term Weights

- The frequency of occurrence for the term in each document is included in the vector

docs	*t1*	*t2*	*t3*
D1	2	0	3
D2	1	0	0
D3	0	4	7
D4	3	0	0
D5	1	6	3
D6	3	5	0
D7	0	8	0
D8	0	10	0
D9	0	0	1
D10	0	3	5
D11	4	0	1

Assigning Weights

- tf x idf measure:
  - term frequency (tf)
  - inverse document frequency (idf) -- a way to deal with the problems of the Zipf distribution
- Goal: assign a tf * idf weight to each term in each document

tf x idf

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

T_k = term k

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

For a collection of 10000 documents

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

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

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

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

Similarity Measures for document vectors

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

tf x idf normalization

- Normalize the term weights (so longer documents are not unfairly given more weight)
  - normalize usually means force all values to fall within a certain range, usually between 0 and 1, inclusive.

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

Vector space similarity

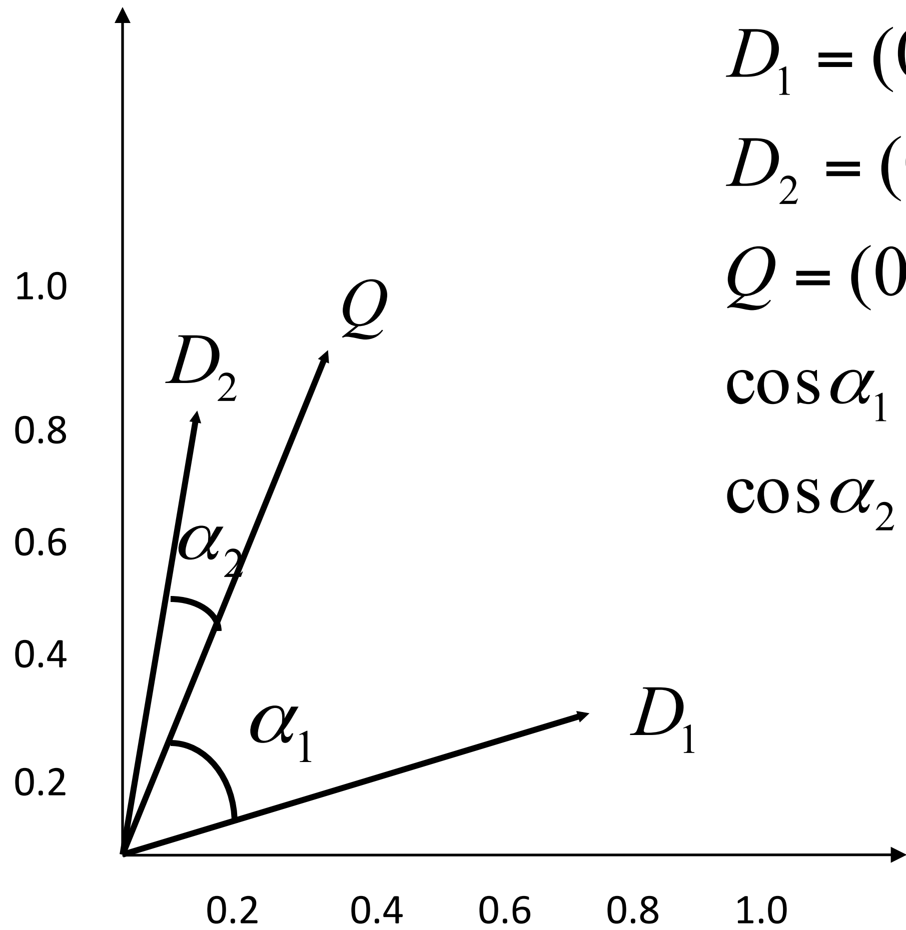
(use the weights to compare the documents)

Now, the similarity of two documents is:

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

This is also called the cosine, or normalized inner product.

Computing Similarity Scores



$$D_1 = (0.8, 0.3)$$

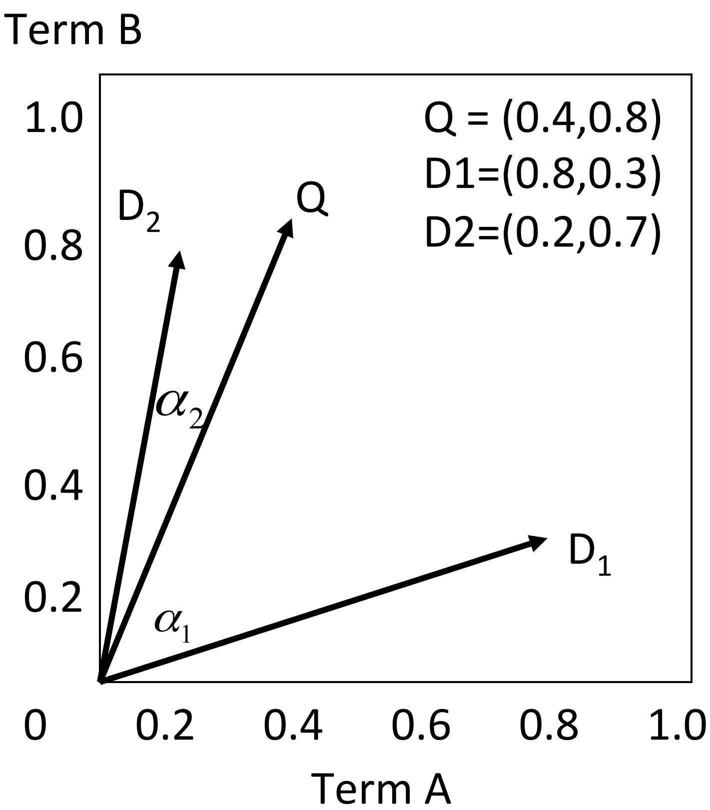
$$D_2 = (0.2, 0.7)$$

$$Q = (0.4, 0.8)$$

$$\cos\alpha_1 = 0.74$$

$$\cos\alpha_2 = 0.98$$

Vector Space with Term Weights and Cosine Matching



$D_i=(d_{i1}, w_{di1}; d_{i2}, w_{di2}; \ldots; d_{it}, w_{dit})$
$Q=(q_{i1}, w_{qi1}; q_{i2}, w_{qi2}; \ldots; q_{it}, w_{qit})$

$$sim(Q, D_i) = \frac{\sum_{j=1}^{t} w_{q_j} w_{d_{ij}}}{\sqrt{\sum_{j=1}^{t} (w_{q_j})^2 \sum_{j=1}^{t} (w_{d_{ij}})^2}}$$

$$sim(Q, D2) = \frac{(0.4 \cdot 0.2) + (0.8 \cdot 0.7)}{\sqrt{[(0.4)^2 + (0.8)^2] \cdot [(0.2)^2 + (0.7)^2]}}$$

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

$$sim(Q, D_1) = \frac{.56}{\sqrt{0.58}} = 0.74$$