Search Engines

Stephen Shaw
<stesh@netsoc.tcd.ie>

Netsoc

18th of February, 2014

Me

- M.Sc. Artificial Intelligence, University of Edinburgh
  - *Would recommend*
- B.A. (Mod.) Computer Science, Linguistics, French, TCD
  - *Would recommend*
- One time Netsoc sysadmin
  - *Would recommend*
- Talk inspired by 'Text Technologies' lecture course given by Dr Victor Lavrenko at the University of Edinburgh
  - *Would recommend*
  - `http://www.inf.ed.ac.uk/teaching/courses/tts/`

Information Retrieval

- Field of AI
- Perfecting the task of getting information from data
- Efficiency
- Accuracy
- Hot topic right now
  - 'Big Data'
  - Buzzwords

The task

- Match a query, *Q*, to a document, *D*
- Not so clear what 'matching' is
- A tricky supervised learning problem
  - Easy to get feedback (they clicked on it = good)
  - But by then it's too late (they've already got the results)
  - Adversarial task ('search engine optimization')
- Relational algebra sucks
  - too slow
- Natural language processing sucks
  - too slow
  - doesn't generalize across domains

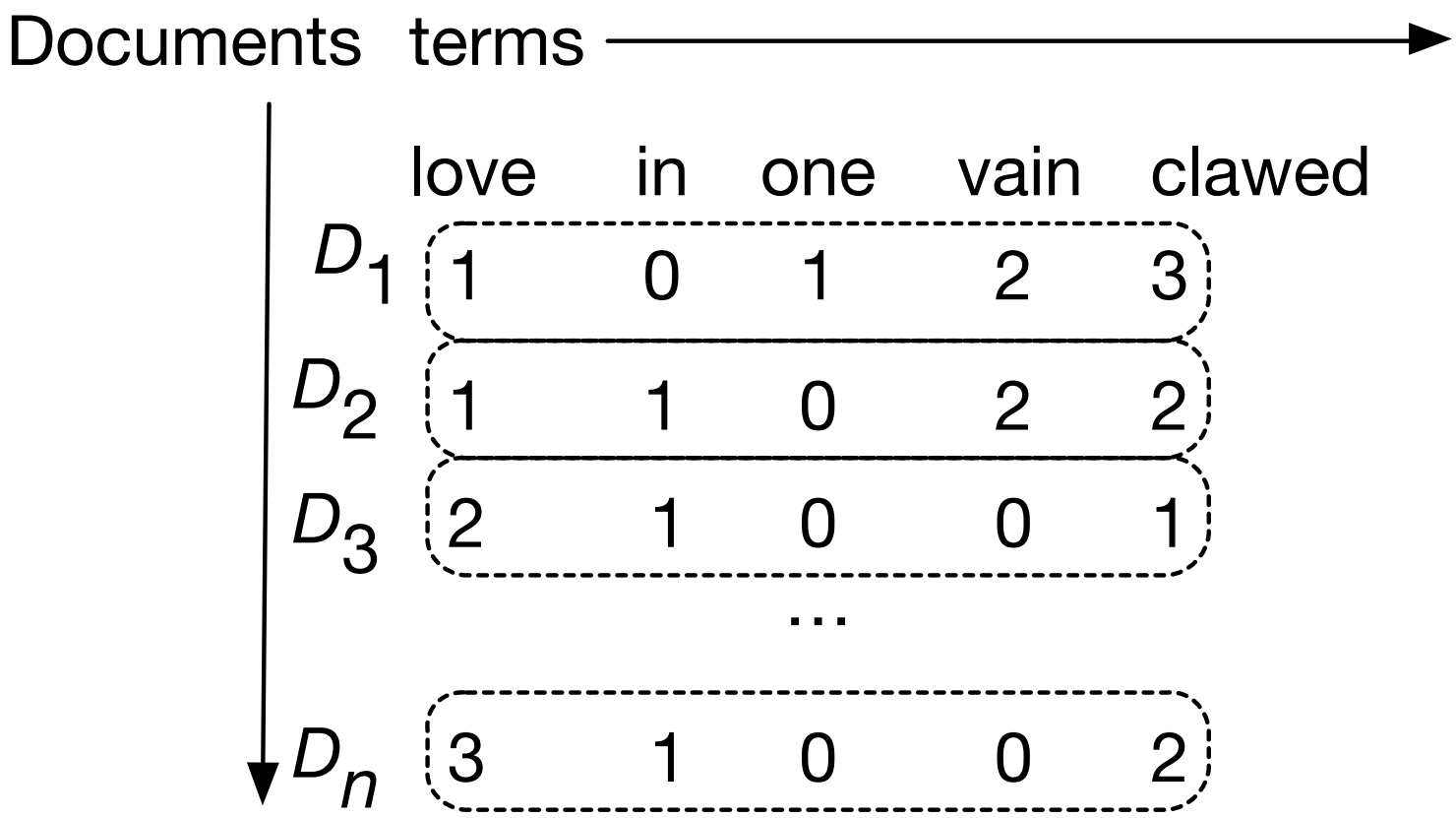
Bag of words

- Revolutionary idea: if it matches, it's similar
- Represent documents and queries as a 'bag of words'
- Word order doesn't make much of a difference
- Example
  - Document: `we clawed we chained our hearts in vain`
  - B-o-w: `{'we': 2, 'clawed': 1, 'chained': 1, 'our': 1, 'hearts': 1, 'vain': 1, 'in': 1}`

Vector space model

- How to compare bags of words?
- Each word ('term') is a dimension
- Each document is a vector
- Frequency of term w in D is its magnitude in dimension w
- Euclidean distance?
- $s(Q, D) = \sqrt{\sum_{i=1}^{n} (Q_i - D_i)^2}$

Vector space model



Cosine similarity

- Euclidean distance sensitive to length
- A poor inductive bias
- Cosine similarity
-

$$s(Q, D) = \frac{\sum_{w \in (Q \cap D)} Q_w D_w}{\sqrt{\sum_{w \in Q} {Q_w}^2} \times \sqrt{\sum_{w \in D} {D_w}^2}}$$

- Best match = D with smallest angle between it and Q
- Problem?

Favouring important terms

- Too much weight given to semantically vacuous terms
- *a*, *the*, *one*, …
- Zipf's law: $f(w) \propto \frac{1}{R(w)}$
- $tf - idf$
  - penalize terms that appear in a lot of documents
  - $tf_{w,D}$ # times term w appears in document D
  - df_w: # documents w appears in

tf-idf weighted sum

- Term frequency, *inverse* document frequency
- $$s(Q, D) = \sum_{w \in V} tf_{w,Q} \times \frac{tf_{w,D}}{tf_{w,D} + \frac{k \times |D|}{avg|d|, d \in C}} \times \log \frac{|C|}{df_w}$$
- $|C|$: # documents in corpus C
- $avg|d|$: average length of a document d in C
- k: free parameter (lets you tune against document length)

Curse of dimensionality

- Each term is a dimension of the vector space
- That's a lot of dimensions
- Lots of dimensions = lots of headache
  - Huge, sparse matrix
  - Inaccurate matches due to vocabulary mismatch

Dimensionality reduction

- Stopword supression: delete semantically vacuous words
- Stemming: map all inflected word forms to their stems
- Clever tokenization: strip punctuation
- Soundex fingerprinting: help overcome badly-spelled queries
- *n*-gram models: suggest previous queries to user before query

Executing queries: first pass

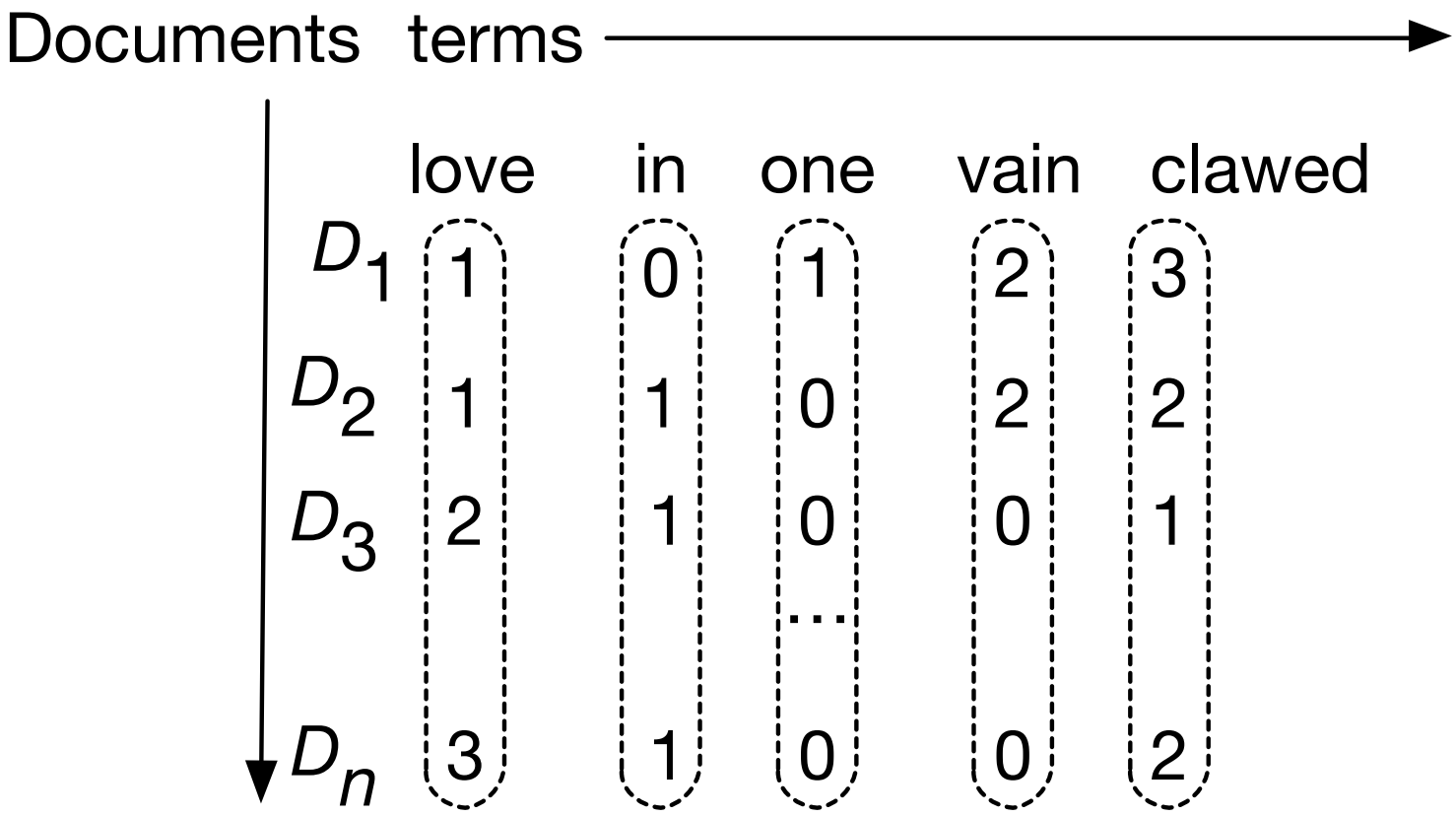
1. Index user's query Q
2. Compute $s(Q, D_i)$ for all $D_i \in C$
3. Toss results into a priority queue with $s(Q, D_i)$ as priority number
4. Dequeue first n results

- Good luck
- There are essentially infinite documents
- …and finite memory and time

Rethinking document representation

- Need a cleverer data structure
- Bag of words: records which *terms* appear in each document
- A *document-centric* representation
- We need a *term-centric* representation
- Instead record which *documents* contain each *term*

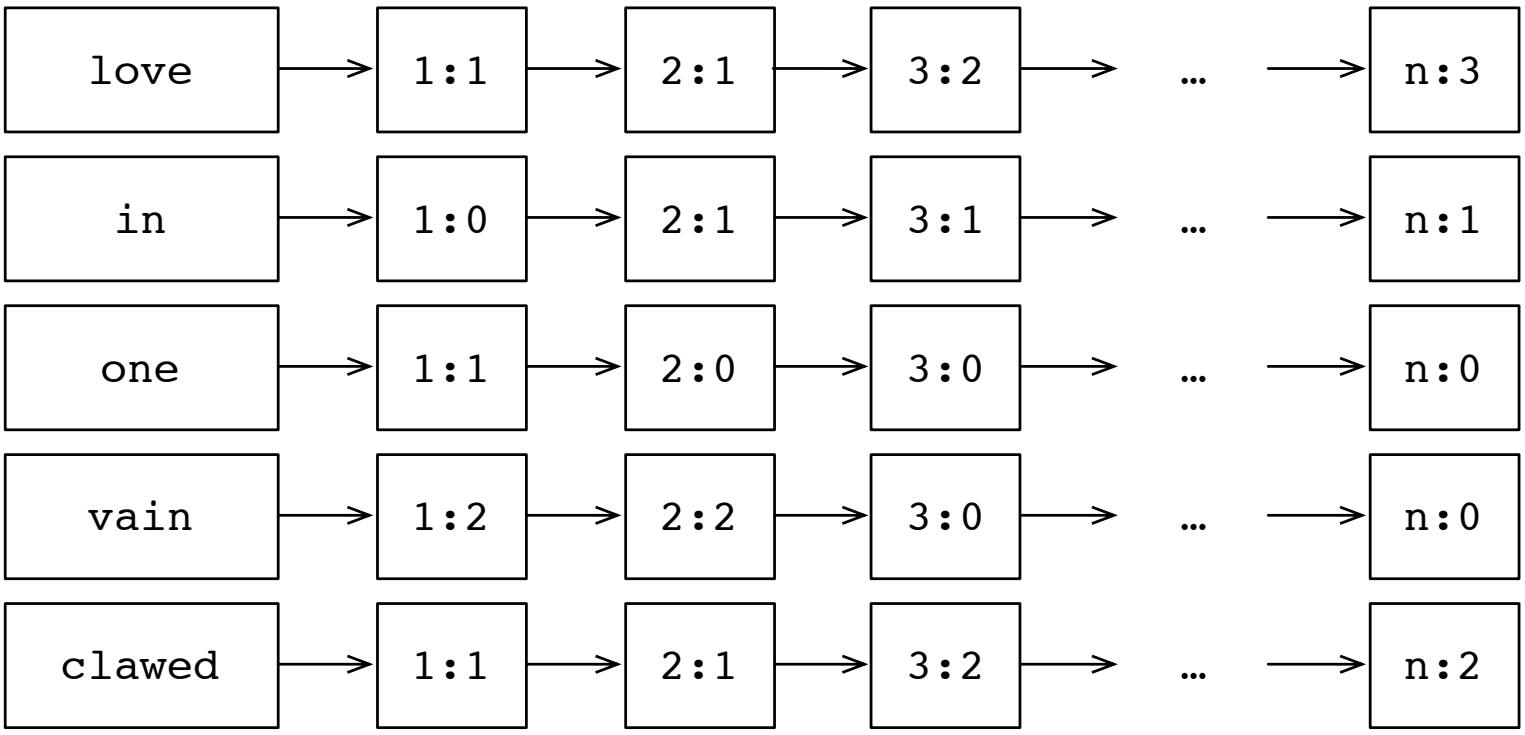
Term-centric vector space



Postings lists

- Representation still very sparse (remember Zipf's law)
- Specialized data structure needed
- Postings lists fit the bill
- key-value structure
- keys: terms
- values: linked list of tuples (document ID, frequency)

Postings lists



Query by merge

- Execute queries by *merging* postings lists
- $Q = \{wrecking, ball\}$
- Look up postings lists for *wrecking* and *ball*
- Set pointers at beginning of each list
- Merge left-to-right to construct new list
- Read off results

doc-at-a-time

- Record the query term postings entries for each document and merge to compute s
- Works well, but can degenerate when documents are of vastly different lengths
- Merging lots of lists is slow: $O(|C| \times |Q|)$
- Memory-expensive: $O(|Q|)$

term-at-a-time

- Initialize a new postings list for each query term
- Look up each term's postings list in turn
- Update aggregate score for each term
- Linear runtime

Construction and storage

- Amenable to distributed computation (e.g. Map-Reduce)
- Postings lists still take up a lot of space
- Usually store them on disk unless you have a stupid amount of RAM
- Clever compression techniques
- Can add additional pointers to speed up linear access

Link analysis

- The Internet is a special kind of corpus
- Documents link to each other
- Forms a directed graph
- Relevant to ranking
- Favour popular pages over unpopular pages when ranking results

Extracting content

- Crawler starts on a random page
- Follows outgoing links
- Statistical, layout-independent methods for extracting page content
- Duplicate-detection important
  - News articles
  - Wikipedia clones

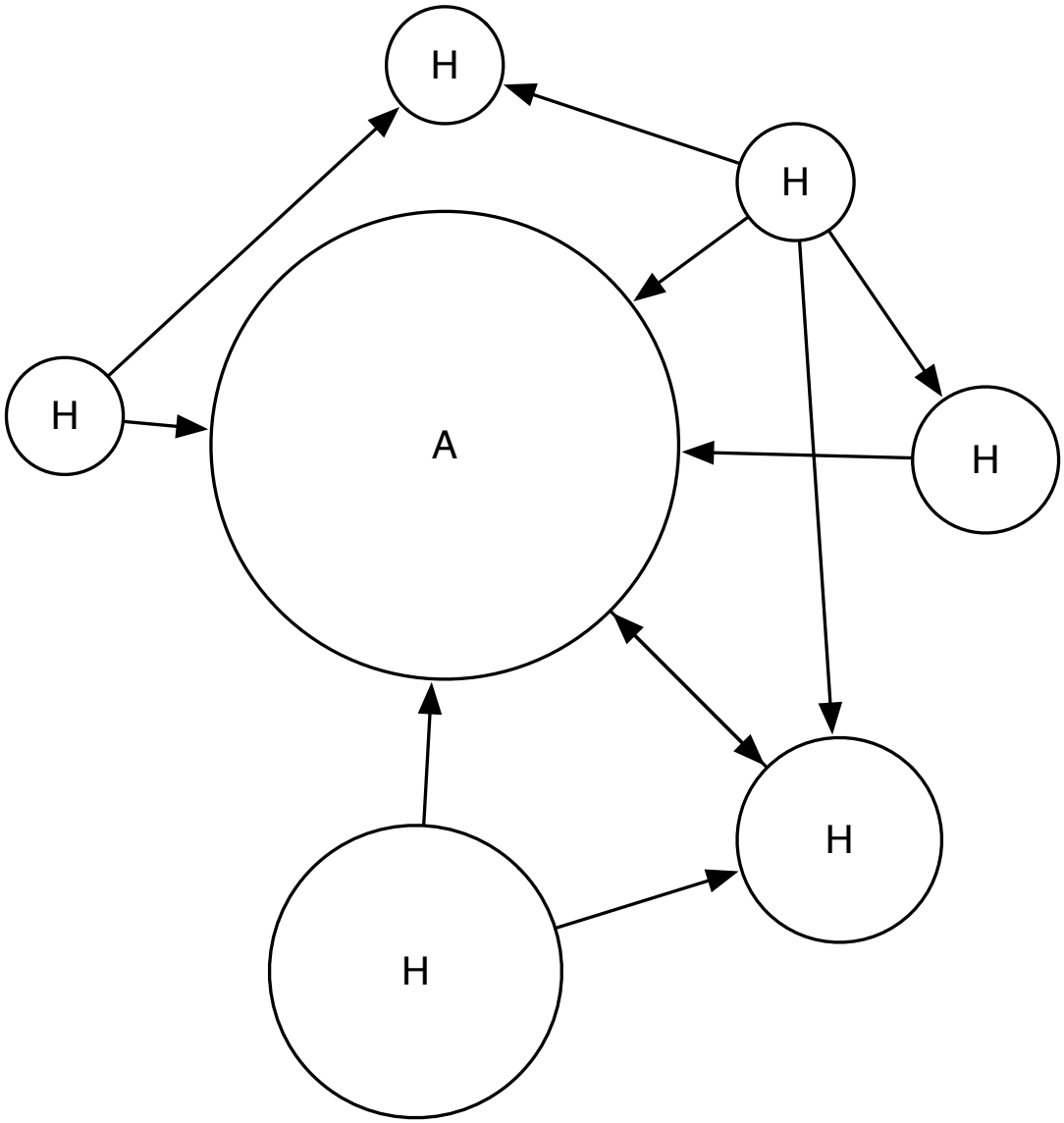
Duplicate detection

- Define a clever hash function
- Cryptographic hashing: collisions = bad
- Duplicate detection hashing: cook up collisions such that duplicates collide

Hubs and authorities

- Hub: page that links to lots of other pages (e.g. Google)
- Authority: page that a lot of pages link to (e.g. Wikipedia)
- HITS algorithm iteratively rates pages as hubs or authorities
- Useful in ranking
- Useful for driving crawlers
- But vulnerable to SEO

Hubs and authorities



PageRank

- 'Rank by consensus'
- If P_1 links to P_2 then P_1 likes P_2

That's it

- Questions