Web Search Engines

Web Search: Challenges & Opportunities

- Challenges
  - Scalability
    → Parallel indexing & searching (MapReduce)
    - How to handle the size of the Web and ensure completeness of coverage?
    - How to serve many user queries quickly?
  - Low quality information and spams
    → Spam detection & robust ranking
  - Dynamics of the Web
    - New pages are constantly created and some pages may be updated very quickly
- Opportunities
  - Many additional heuristics (e.g., links) can be leveraged to improve search accuracy
    → Link analysis & multi-feature ranking

Ranking Algorithms for Web Search

- Standard IR models apply but aren't sufficient
  - Different information needs
  - Documents have additional information
  - Information quality varies a lot
- Major extensions
  - Exploiting links to improve scoring
  - Exploiting clickthroughs for massive implicit feedback
  - In general, rely on machine learning to combine all kinds of features

“Free Text” vs. “Structured Text”

- So far, we’ve assumed “free text”
  - Document = word sequence
  - Query = word sequence
  - Collection = a set of documents
  - Minimal structure …
- But, we may have structures on text (e.g., title, hyperlinks)
  - Can we exploit the structures in retrieval?
  - Sometimes, structures may be more important than the text

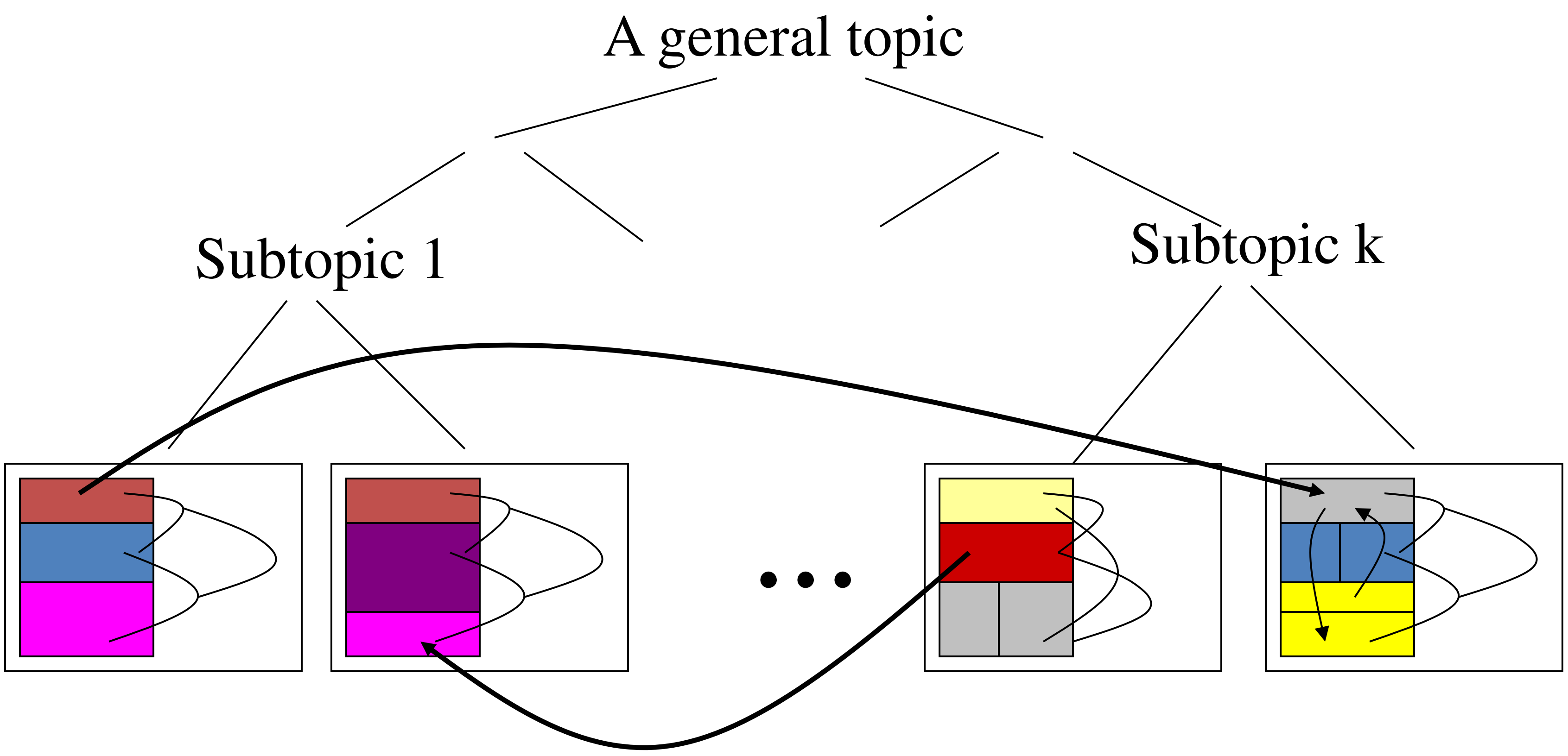
Examples of Document Structures

- Intra-doc structures (=relations of components)
  - Natural components: title, author, abstract, sections, references, …
  - Annotations: named entities, subtopics, markups, …

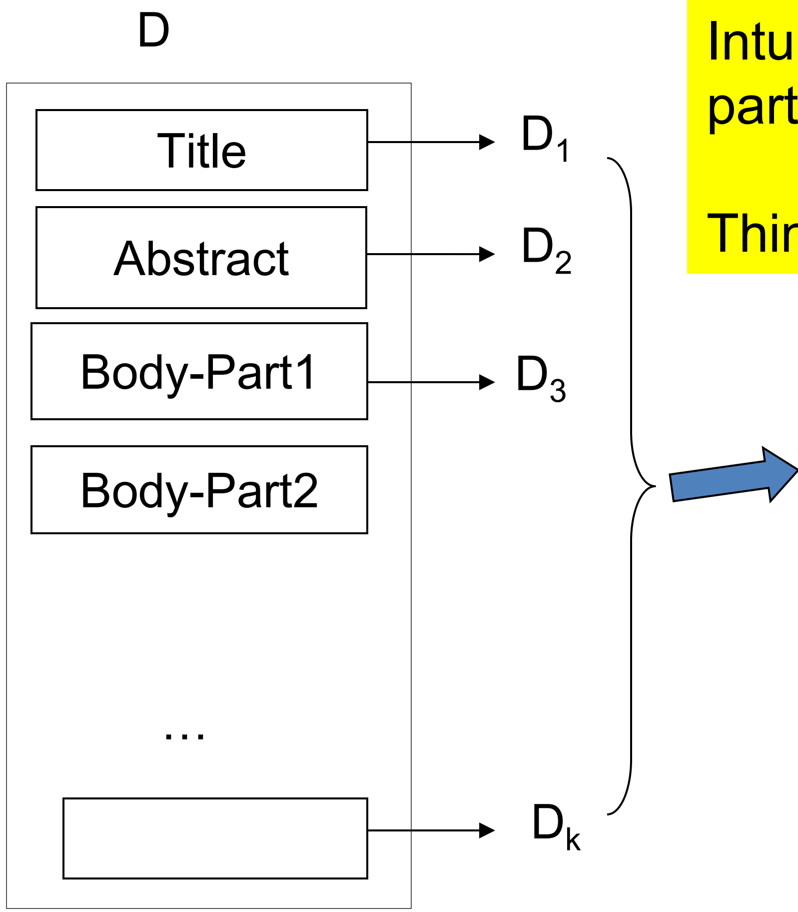
- Inter-doc structures (=relations between documents)
  - Topic hierarchy
  - Hyperlinks/citations (hypertext)

Structured Text Collection

General question: How do we search such a collection?



Exploiting Intra-Document Structures



Intuitively, we want to combine all the parts, but give more weights to some parts

Think about query-likelihood model…

Select D_j and generate a query word using D_j

$$p(w|\theta_D)=\sum_{i=1}^{k}\lambda_i p(w|\theta_{D_i})$$

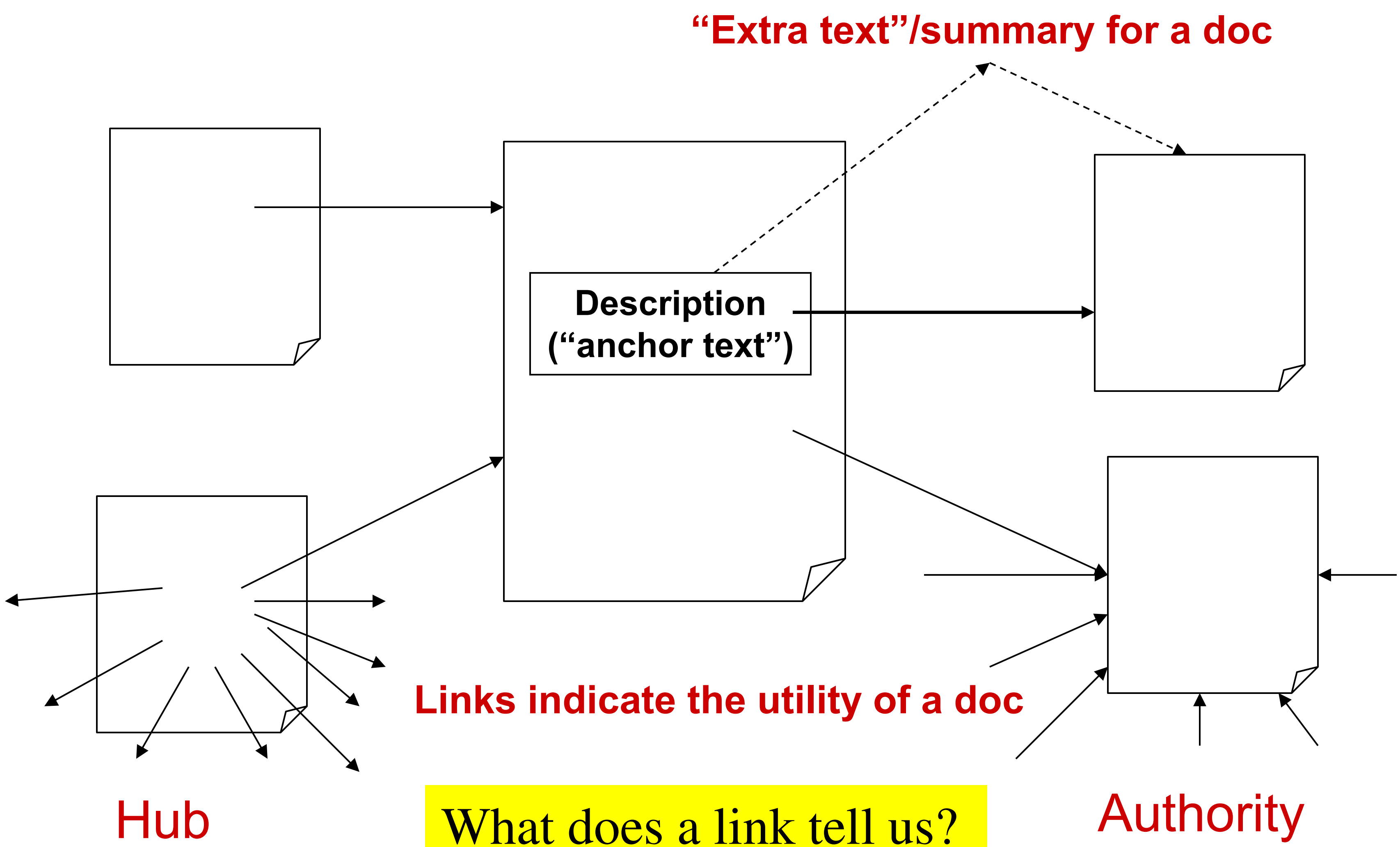
$$\sum_{i=1}^{k}\lambda_i=1, \text{ and for } 1\le i\le k, \lambda_i\ge 0$$

Anchor text can be treated as a “part” of a document

Exploiting Inter-Document Structures

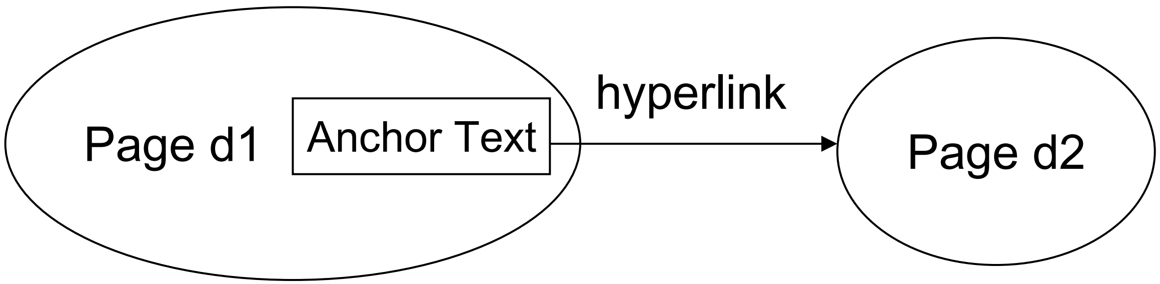
- Document collection has links (e.g., Web, citations of literature)
- Query: text query
- Results: ranked list of documents
- Challenge: how to exploit links to improve ranking?

Exploiting Inter-Document Links



Anchor Text

The Web as a Directed Graph

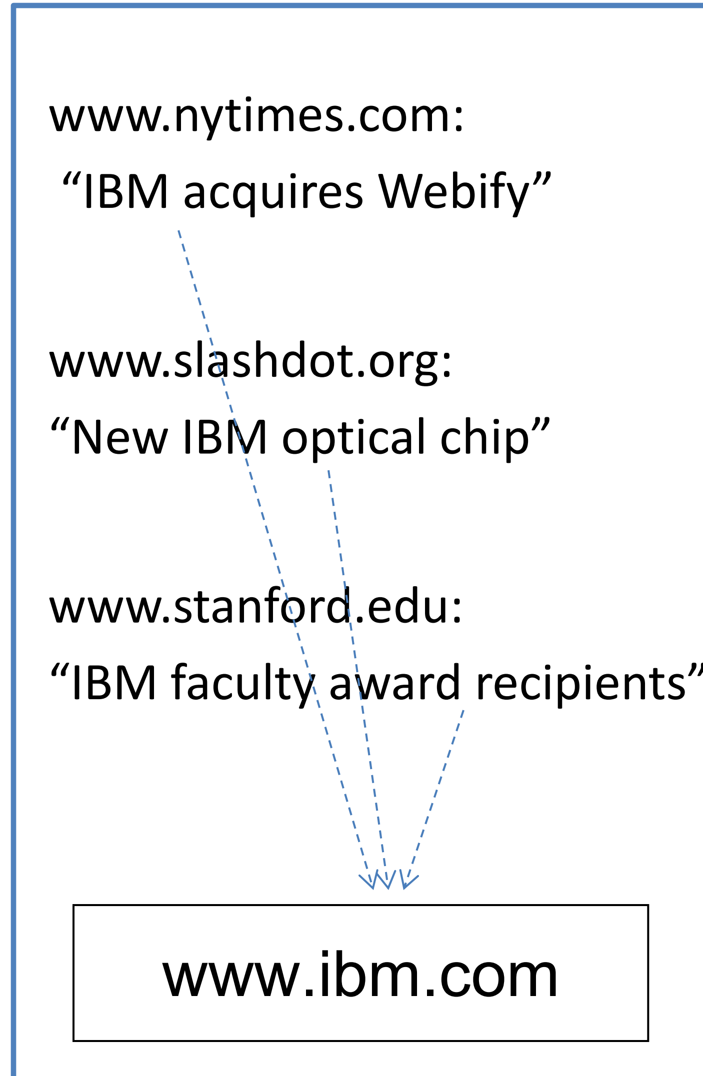


- Assumption 1: A hyperlink is a quality signal.
  - A hyperlink between pages denotes that the author perceived relevance.

- Assumption 2: The anchor text describes the target page.
  - We use anchor text somewhat loosely here: the text surrounding the hyperlink. E.g., “You can find cheap cars here.”

[Document Text Only] vs. [Document Text + Anchor Text]

- Searching on [document text + anchor text] is often more effective than searching on [document text only].
- Example: Query IBM
  - Matches IBM's copyright page
  - Matches many spam pages
  - Matches IBM wikipedia article
  - May not match IBM home page! (if IBM homepage is mostly graphical)
- Searching on anchor text is better for the query IBM.
- Represent each page by all the anchor text pointing to it.
- In this representation, the page with the most occurrences of IBM is www.ibm.com.



PageRank

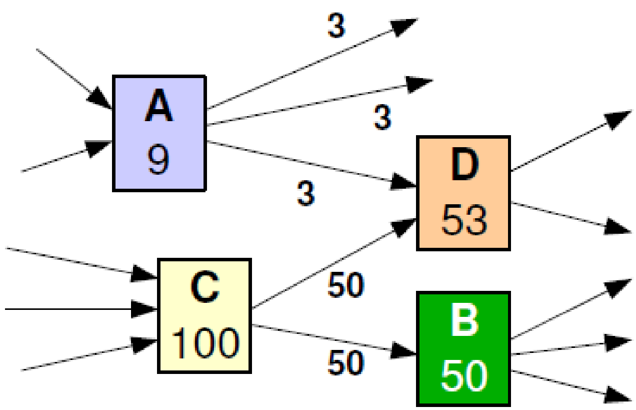
PageRank: Capturing Page "Popularity"

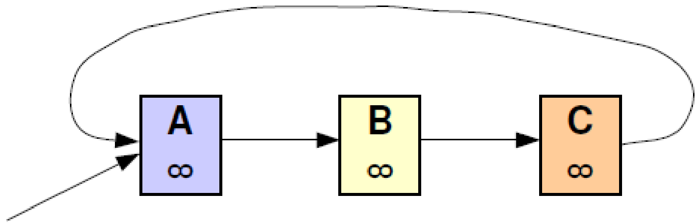
[Page & Brin 98]

- Intuitions
  - Links are like citations in literature
  - A page that is cited often can be expected to be more useful in general
- PageRank is essentially "citation counting", but improves over simple counting
  - Consider "indirect citations" (being cited by a highly cited paper counts a lot…)
  - Smoothing of citations (every page is assumed to have a non-zero citation count)
- PageRank can also be interpreted as random surfing (thus capturing popularity)

PageRank – An Intuitive Description

- *A page has a high rank if the sum of the ranks of its backlinks is high.*
- Rank of each page is divided among its forward links evenly to contribute to ranks of the pages it points to.
- Problem: If some pages point to each other, but no other pages, during iterations, the loop will accumulate rank and will never distribute any rank.
- The loop forms a trap which is called a rank sink. To overcome the problem, rank source is introduced.
- If a real Web surfer gets into a small loop of pages, it is unlikely that it will continue in the loop forever and jumps to some other page.





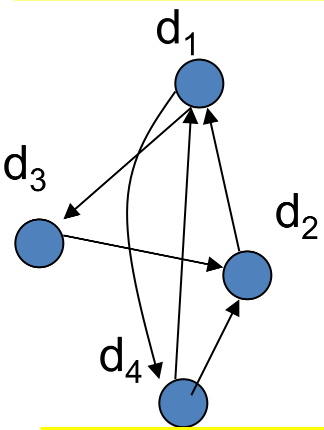
The PageRank Algorithm [Page et al. 98]

Random surfing model: At any page,

With prob. α, randomly jumping to a page

With prob. (1-α), randomly picking a link to follow.

p(di): PageRank score of di = average probability of visiting page di



"Transition matrix"

$$M = \begin{bmatrix} 0 & 0 & 1/2 & 1/2 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \end{bmatrix}$$

M_{ij} = probability of going from d_i to d_j

$$\sum_{j=1}^{N} M_{ij} = 1$$

Probability of visiting page dj at time t+1

Probability of visiting page di at time t

Equilibrium Equation:
N= # pages

$$p_{t+1}(d_j) = \underbrace{(1-\alpha)\sum_{i=1}^{N} M_{ij} p_t(d_i)}_{\text{Reach dj via following a link}} + \underbrace{\alpha \sum_{i=1}^{N} \tfrac{1}{N} p_t(d_i)}_{\text{Reach dj via random jumping}}$$

Reach dj via following a link

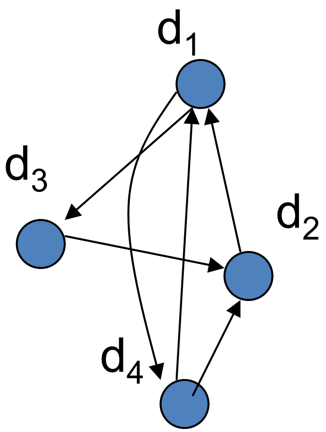
Reach dj via random jumping

Dropping the time index

$$p(d_j) = \sum_{i=1}^{N} [\tfrac{1}{N}\alpha + (1-\alpha) M_{ij}] p(d_i) \quad \Rightarrow \quad \vec{p} = (\alpha I + (1-\alpha) M)^T \vec{p}$$

I_{ij} = 1/N

PageRank: Example



$$p(d_j) = \sum_{i=1}^{N} [\tfrac{1}{N}\alpha + (1-\alpha)M_{ij}] p(d_i)$$

$$\vec{p} = (\alpha I + (1-\alpha)M)^T \vec{p}$$

$$A = (1-0.2)M + 0.2I = 0.8 \times \begin{bmatrix} 0 & 0 & 1/2 & 1/2 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \end{bmatrix} + 0.2 \times \begin{bmatrix} 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 \end{bmatrix}$$

$$\begin{bmatrix} p^{n+1}(d_1) \\ p^{n+1}(d_2) \\ p^{n+1}(d_3) \\ p^{n+1}(d_4) \end{bmatrix} = A^T \begin{bmatrix} p^{n}(d_1) \\ p^{n}(d_2) \\ p^{n}(d_3) \\ p^{n}(d_4) \end{bmatrix} = \begin{bmatrix} 0.05 & 0.85 & 0.05 & 0.45 \\ 0.05 & 0.05 & 0.85 & 0.45 \\ 0.45 & 0.05 & 0.05 & 0.05 \\ 0.45 & 0.05 & 0.05 & 0.05 \end{bmatrix} \times \begin{bmatrix} p^{n}(d_1) \\ p^{n}(d_2) \\ p^{n}(d_3) \\ p^{n}(d_4) \end{bmatrix}$$

Initial value p(d)=1/N, iterate until converge

PageRank in Practice

- Computation can be quite efficient since M is usually sparse
- Interpretation of the damping factor α ($\approx$0.15):
  - Probability of a random jump
  - Smoothing the transition matrix (avoid zero's)

$$p(d_i) = (1-\alpha)\sum_{k=1}^{N} m_{ki}\, p(d_k) + \alpha \sum_{k=1}^{N} \frac{1}{N} p(d_k) \qquad \vec{p} = (\alpha I + (1-\alpha)M)^T \vec{p}$$

- The zero-outlink problem: p(di)'s don't sum to 1
  - One possible solution = page-specific damping factor (α=1.0 for a page with no outlink)
- Many extensions (e.g. Topic Sensitive PageRank)
- Many other applications (e.g., social network analysis)

PageRank Issues

- Real surfers are not random surfers
- Simple PageRank ranking produces bad results for many pages.
  - Consider the query video service
  - The Yahoo home page (i) has a very high PageRank and (ii) contains both words.
  - If we rank all Boolean hits according to PageRank, then the Yahoo home page would be top-ranked.
  - Clearly not desirable
- In practice: rank according to weighted combination of many factors, including raw text match, anchor text match, PageRank and many other factors

What Google Says About PageRank

- Through the years…
  - 2020: “Yes, we do use PageRank internally, among many, many other signals. It’s not quite the same as the original paper, there are lots of quirks (e.g., disavowed links, ignored links, etc.), and, again, we use a lot of other signals that can be much stronger [John Mueller, Google]
  - 2017: “DYK that after 18 years we’re still using PageRank (and 100s of other signals) in our ranking? [Gary Illyes, Google]
  - 2011: “We use more than 200 signals, including our patented PageRank algorithm, to examine the entire link structure of the web and determine which pages are most important. We then conduct hypertext-matching analysis to determine which pages are relevant to the specific search being conducted” [http://www.google.com/technology]
  - 2007: “The heart of our software is PageRank, a system for ranking web pages developed by our founders… And while we have dozens of engineers working to improve every aspect of Google on daily basis, PageRank continues to play a central role in many of our web search tools.”
  - 2002: “… PageRank continues to provide the basis for all our web search tools.”

HITS

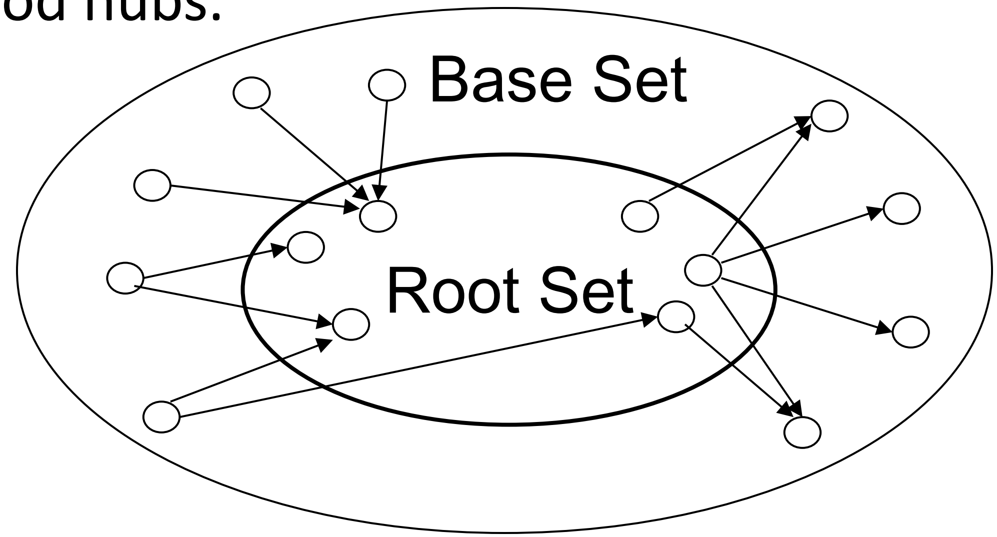
HITS: Capturing Authorities & Hubs

[Kleinberg 98]

- Intuitions
  - Pages that are widely cited are good authorities
  - Pages that cite many other pages are good hubs
- The key idea of HITS (Hypertext-Induced Topic Search)
  - Good authorities are cited by good hubs
  - Good hubs point to good authorities
  - Iterative reinforcement…
- Many applications in graph/network analysis

Root Set and Base Set

- Good authority pages may not contain the query term.
- If the text query manages to capture a good hub page in the root set, then the inclusion of pages linked to by this page will capture good authorities linked to by this page.
- If the text query manages to capture a good authority page in the root set, then inclusion of pages linking to this page will capture good hubs.



Hub and Authority Scores

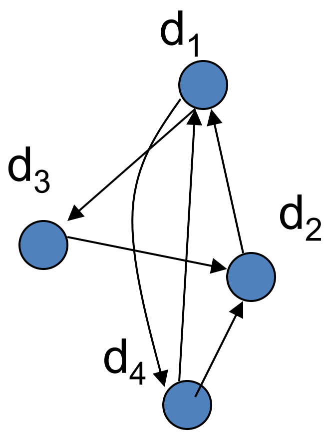
- Compute for each page d in the base set a hub score h(d) and an authority score a(d)
- Initialization: for all d: h(d) = 1, a(d) = 1
- Iteratively update all h(d), a(d)

  For all d: $h(d) = \sum_{d \rightarrow y} a(y)$

  For all d: $a(d) = \sum_{y \rightarrow d} h(y)$

- Iterate these two steps until convergence

The HITS Algorithm [Kleinberg 98]



$$A = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

"Adjacency matrix"

Initial values: $a(d_i)=h(d_i)=1$

$$h(d_i) = \sum_{d_j \in OUT(d_i)} a(d_j)$$

$$a(d_i) = \sum_{d_j \in IN(d_i)} h(d_j)$$

Iterate

Normalize:

PageRank vs. HITS: Discussion

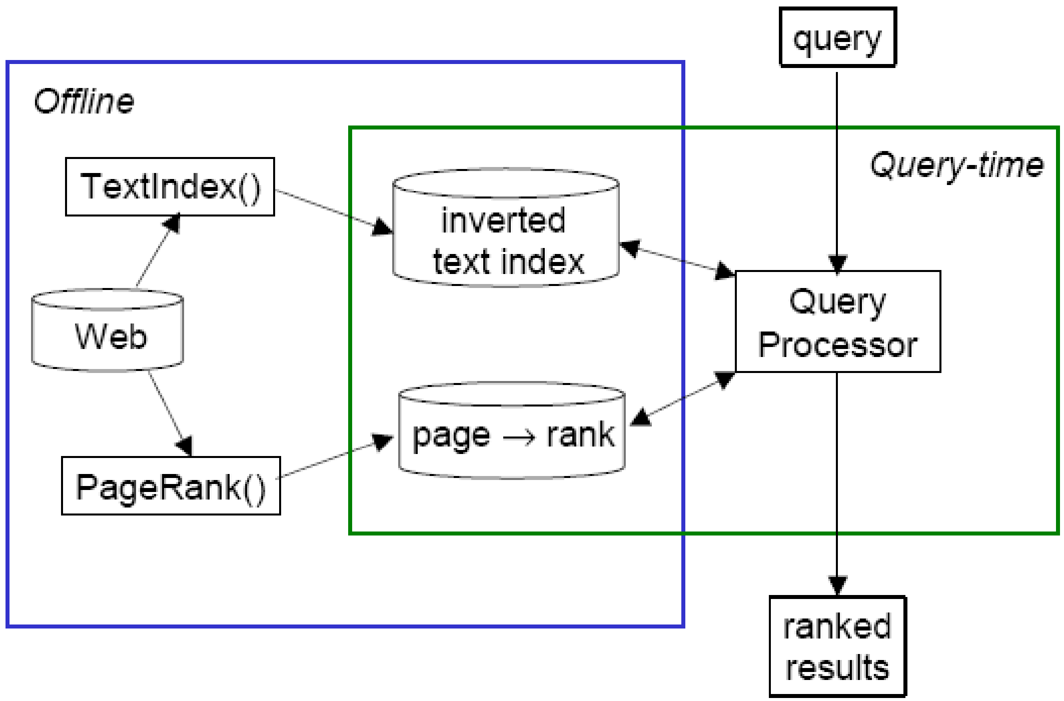
- PageRank can be precomputed, HITS has to be computed at query time.
  - HITS is too expensive in most application scenarios.
- The PageRank and HITS make two different design choices concerning (i) the eigen problem formalization (ii) the set of pages to apply the formalization to.
- These two are orthogonal.
  - We could also apply HITS to the entire web and PageRank to a small base set.

PageRank Extension: Topic-Sensitive PageRank

Topic Sensitive PageRank

- Computes a set of PageRank vectors biased upon a set of representative topics to yield more accurate search results.
- Goals:
  - Allow query to influence link-based score
    (Like HITS)
  - Requires Minimal query-time processing
    (Like PageRank)

PageRank – A Review

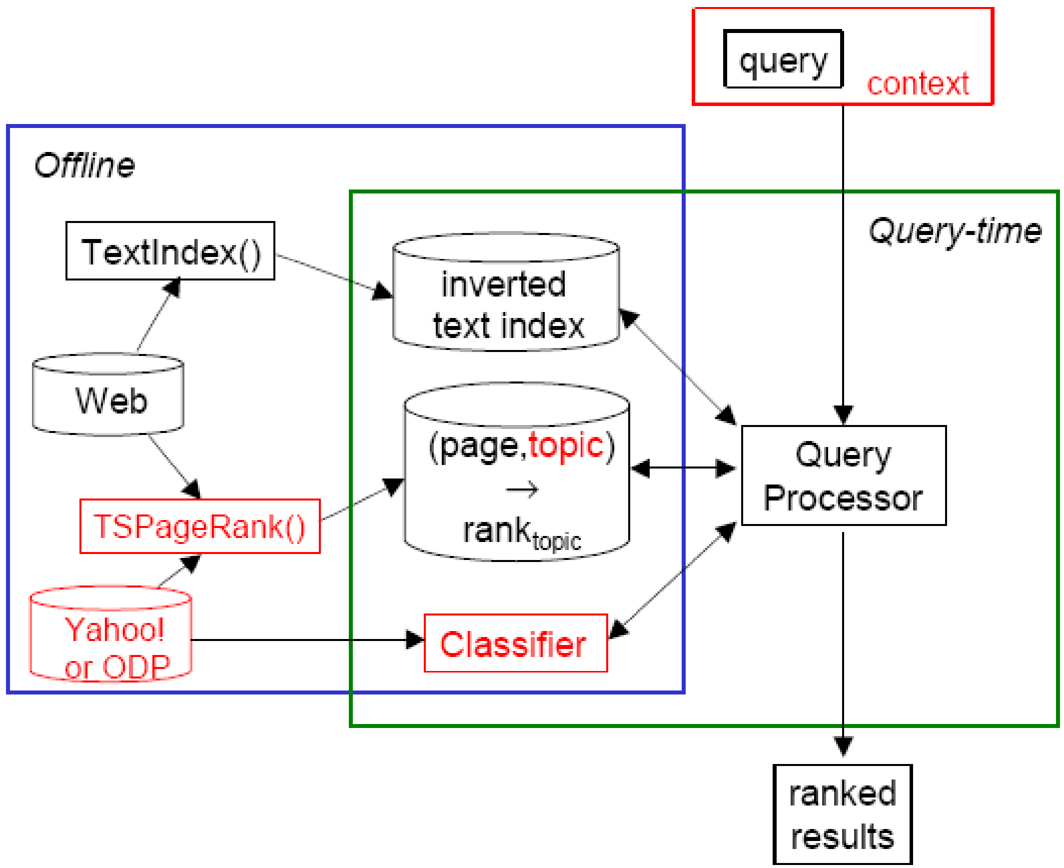


Topic-Sensitive PageRank

- Assigns multiple a-priori “importance” estimates to pages

- One PageRank score per basis topic
  – Query specific rank score
  – Inexpensive at run-time

Topic-Sensitive PageRank



Topic-Sensitive PageRank

- Step 1 – ODP-Biasing
  - Generate set of biased PageRank vectors using a set of basis topics
    - Create 16 different biased PageRank vectors
  - Performed offline, during preprocessing of crawled data
  - URLs in various categories in Open Directory Project (ODP)

Topic-Sensitive PageRank

- For each category c_j , compute: $\vec{p} = \vec{v}_j$

$$v_{ji} = \begin{cases} \dfrac{1}{|T_j|} & i \in T_j \\ 0 & i \notin T_j \end{cases}$$

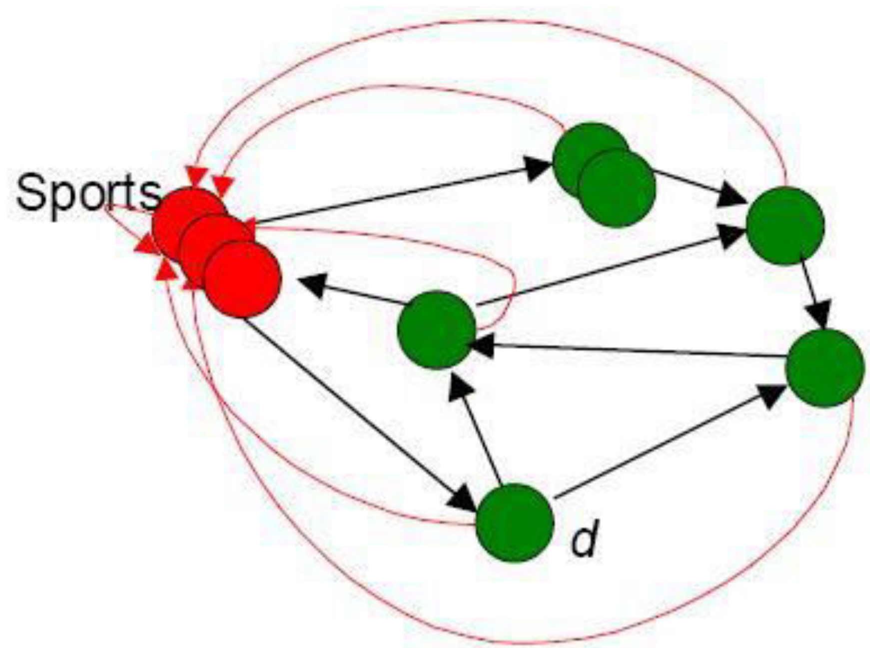
With T_j the set of URL for the ODP category c_j

$\overrightarrow{PR}(\alpha, \vec{v}_j)$ will be PageRank vector for c_j

- Also compute the 16 class term vectors $\vec{D}_j$

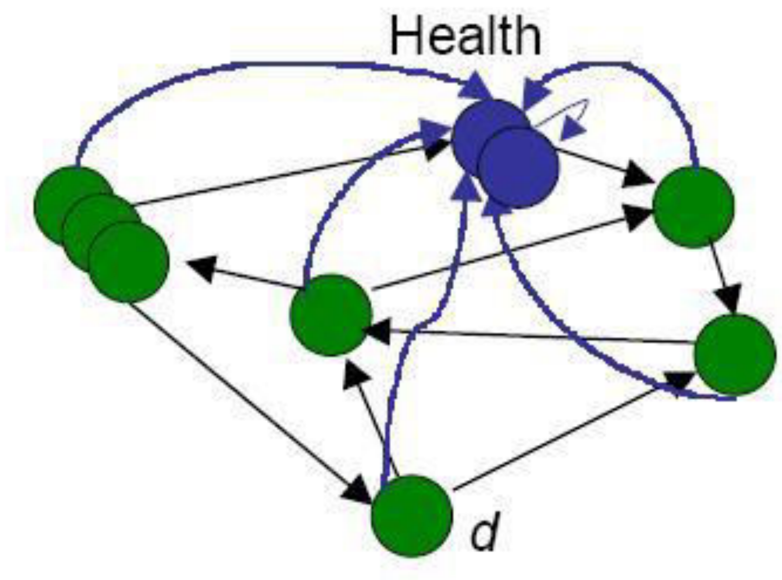
D_{jt} gives the number of occurrences of t in documents of class c_j

Graphical depiction of part I



Select set of pages on a topic, calculate PageRank for all pages.
For example $r_{sports}[d] = 0.05$

Graphical depiction of part I



Select set of pages on a topic, calculate PageRank for all pages.
For example $r_{health}[d] = 0.01$

Topic-Sensitive PageRank

- Step 2 – Query-Time Importance Score
  - Performed at query time.
  - Compute class probabilities for each of the 16 top-level ODP classes.

$$P(c_j|q) = \frac{P(c_j).P(q|c_j)}{P(q)} \propto P(c_j).\prod_i P(q_i|c_j)$$

  - Retrieve URLs for all documents containing the original query terms (r_{1d}, r_{2d}, …, r_{16d}).
  - Compute query-sensitive importance score for each retrieved URL.

$$S_{qd} = \sum_j P(c_j|q).rank_{jd}$$

$rank_{jd}$ is the rank of the document d according to $\overrightarrow{PR}(\alpha, \overrightarrow{v_j})$

HITS Extension: Topic distillation

Topic Distillation

- HITS problems:
  - Mutually Reinforcing Relationships Between Hosts

    certain arrangements of documents “conspire” to dominate the computation
  - Automatically Generated Links

    no human opinion is expressed by the link
  - Non-relevant Nodes

    the graph contains documents not relevant to the query topic

Topic Distillation – Proposed Solution

- Improved Connectivity Analysis

$$A[n] := \sum\nolimits_{(n',n)\in N} H[n'] \times auth\_wt(n',n)$$

$$H[n] := \sum\nolimits_{(n,n')\in N} A[n'] \times hub\_wt(n,n')$$

- Combine Connectivity and Content Analysis
  - Compute Relevance Weights for Nodes
  - Prune Nodes from the Neighborhood Graph
  - Regulate the Influence of a Node

Combining Ideas from PageRank and HITS: SALSA - Stochastic Approach for Link-Structure Analysis

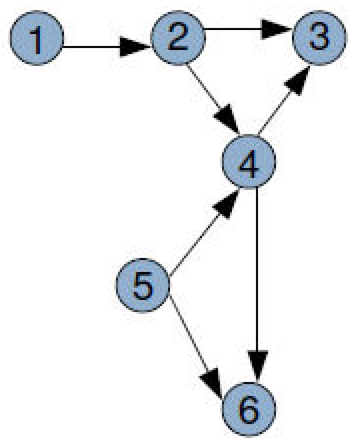
SALSA – Walk on a Bipartite Graph

- An alternative algorithm, that combines ideas from both PageRank and HITS, was proposed in 2001 by Lempel and Moran.

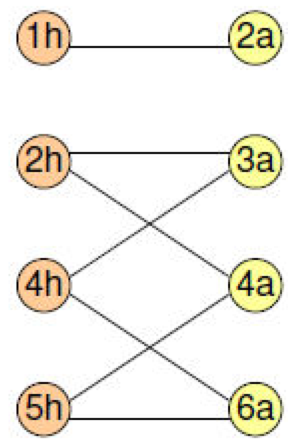
- The SALSA algorithm splits the set of nodes into a bipartite graph, and then performs a random walk alternating between the hub and authority sides.

SALSA – Construction of the Graph

- Each non-isolated page is represented in the bipartite graph with one or two nodes.



(standard collection)



(bipartite graph)

- The authority walk starts from an authority node selected at random and then proceeds alternating backwards and forwards steps.

SALSA – A Variation of HITS

- The probability to move from authority i to authority j is then

$$\sum_{k:k\in B(i)\cap B(j)} \frac{1}{|B(i)|}\frac{1}{|F(k)|}$$

- Instead of simply broadcasting its weight, each node divides its hub/authority weight equally among the authorities/hubs connected to it.

$$a_i \leftarrow \sum_{j:j\in B(i)} \frac{1}{|F(j)|} h_j \qquad\qquad h_i \leftarrow \sum_{j:j\in F(i)} \frac{1}{|B(j)|} a_j$$

(*I* - operation) (*O* - operation)

Web Spamming

Defining web spam

- Working Definition
  - Spam web page: A page created for the sole purpose of attracting search engine referrals (to this page or some other "target" page)

- Ultimately a judgment call
  - Some web pages are borderline useless
  - Sometimes a page might look fine by itself, but in context it clearly is "spam"

Why web spam is bad

- Bad for users
  - Makes it harder to satisfy information need
  - Leads to frustrating search experience

- Bad for search engines
  - Burns crawling bandwidth
  - Pollutes corpus (infinite number of spam pages!)
  - Distorts ranking of results

Taxonomy of web spam techniques

- "Keyword stuffing"
- "Cloaking"
- "Link spam"

Link Spam

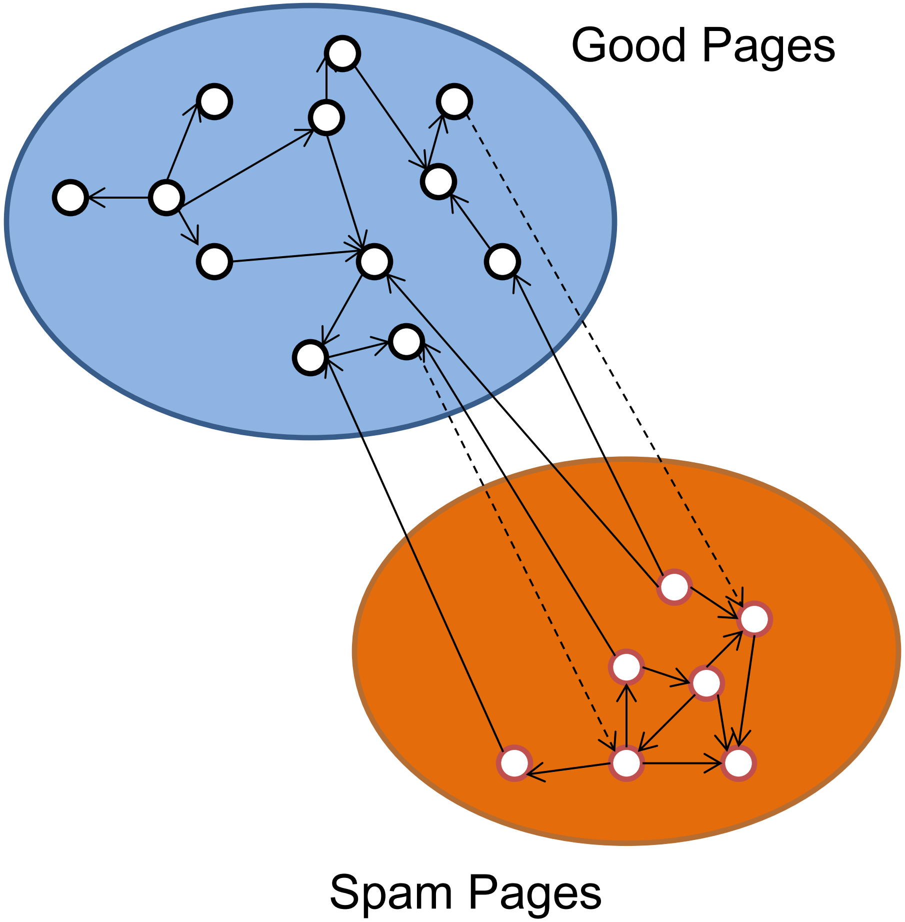
- Since link analysis has played an important role in search engines, it has large commercial values
- Improving one's PageRank, can directly increase one's clicks thus earn more money.
- Link Spam is something trying to unfairly gain a high ranking on a search engine for a web page without improving the user experience, by mean of tricky modification / manipulation of the link graph.

Link Spamming Technologies

- Adding outlinks
  - Replicate hub pages
- Adding inlinks
  - Create a honey pot
  - Infiltrate a web directory
  - Post links on blog, wiki, etc
  - Participate in-link exchange
  - Buy expired domains
  - Create own spam farm (Clique Attack)

Anti-Spam: TrustRank

- Basic assumption
  - Good pages seldom point to spam pages, but spam pages may very likely point to good pages.
- Use TrustRank to denote the goodness of a webpage, and use Trust Propagation to label all the web pages starting from a small human-labeled seed set.



TrustRank – Computing Trust

- Initialization

$$T_O(p) = \begin{cases} O(p) & \text{if } \mathrm{p} \in \mathrm{S} \\ \dfrac{1}{2} & \text{otherwise} \end{cases}$$

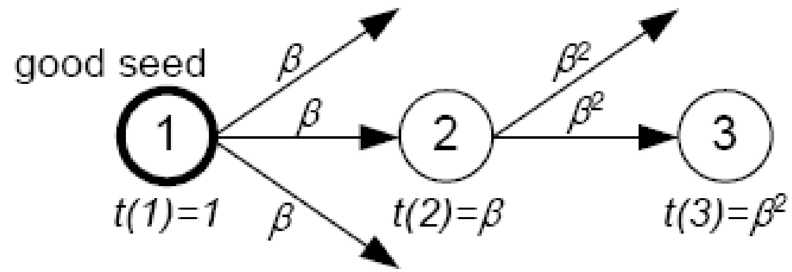
- Propagation

M-Step Trust Function

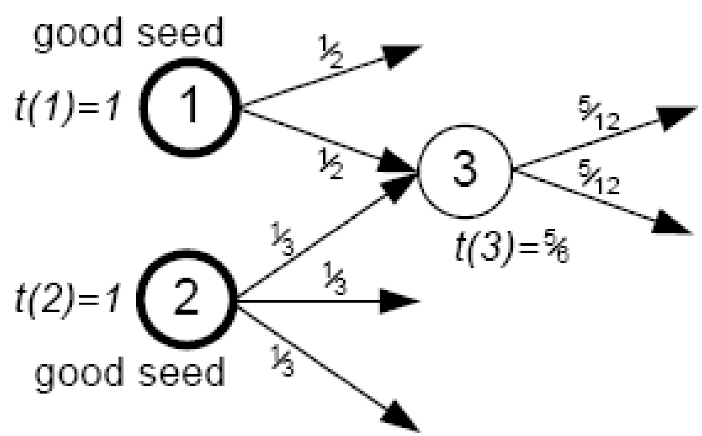
$$T_M(p) = \begin{cases} O(p) & \text{if } \mathrm{p} \in \mathrm{S} \\ 1 & \text{if } \mathrm{p} \notin \mathrm{S} \text{ and } \exists \mathrm{q} \in \mathrm{S}^+ : q \xrightarrow{M} p \\ \dfrac{1}{2} & \text{otherwise} \end{cases}$$

TrustRank – Trust Attenuation

- Trust Dampening



- Trust Splitting



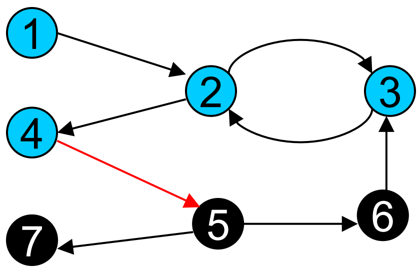
TrustRank - Algorithm

1) **s** = SelectSeed(…) //evaluate seed desirability of pages
2) $\boldsymbol{\sigma}$ = Rank({1, …, N}, **s**) // select good seeds
3) $\mathbf{d} = \mathbf{0}_N$
4) <u>for</u> i = 1 <u>to</u> L <u>do</u>
   <u>if</u> (O($\boldsymbol{\sigma}$(i)) == 1) then
      $\mathbf{d}(\boldsymbol{\sigma}(i)) = 1$
5) $\mathbf{d} = \mathbf{d} / |\mathbf{d}|$ // normalize static score distribution vector
6) $\mathbf{t}^* = \mathbf{d}$ // compute TrustRank scores
7) <u>for</u> i = 1 <u>to</u> M_B <u>do</u>
   $\mathbf{t}^* = \alpha_B.\mathbf{T}.\mathbf{t}^* + (1 - \alpha_B).\mathbf{d}$
8) return $\mathbf{t}^*$

TrustRank – Selecting Seeds

- Inverse PageRank
  - Hub pages, since they have more influence
- High PageRank
  - Important pages are more important to search applications

TrustRank - Example



Select seed

S = [0.08,0.13,0.08,0.10,0.09,0.06,0.02]

Order = [2,4,5,1,3,6,7]

Assume L = 3

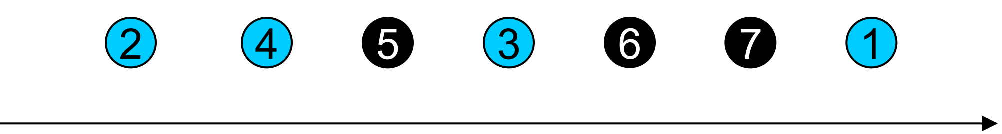
selected seed set = {2,4,5}

d = [0,1/2,0,1/2,0,0,0]

Assume α_B=0.85 and

M_B = 20

T*=[0,0.18,0.12,0.15,0.13,0.05,0.05]



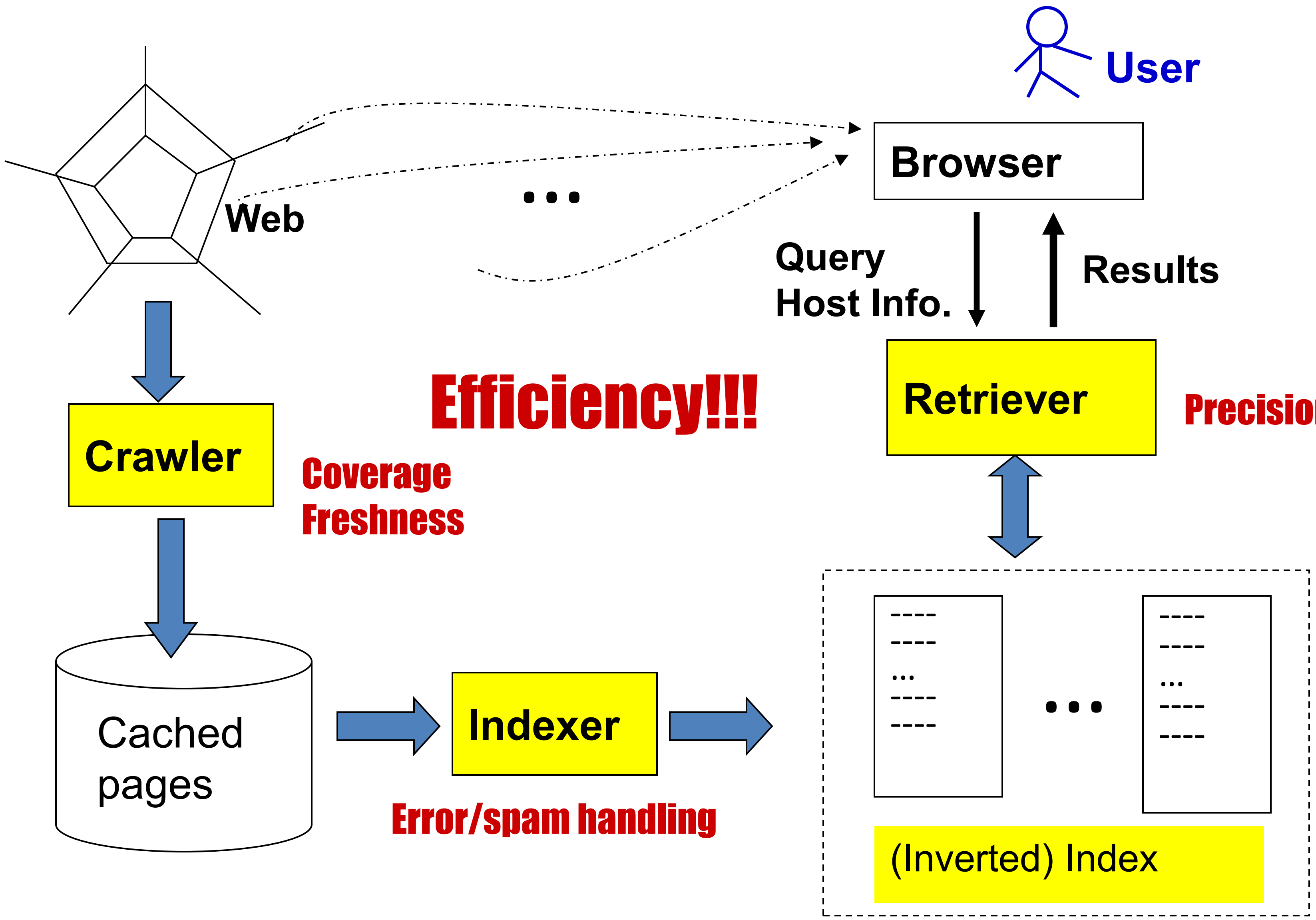
Ordered by trust score

Experiment results show that in spite of errors like these, on a real web graph the algorithm is able to correctly identify a significant number of good pages

Questions?

Search Engine Technologies

Basic Search Engine Technologies



Component I: Crawler/Spider/Robot

- Building a “toy crawler” is easy
  - Start with a set of “seed pages” in a priority queue
  - Fetch pages from the web
  - Parse fetched pages for hyperlinks; add them to the queue
  - Follow the hyperlinks in the queue
- A real crawler is much more complicated…
  - Robustness
  - Politeness
  - Distributed
  - Scalable
  - Performance and efficiency
  - Quality
  - Freshness
  - Extensible

Major Crawling Strategies

- Breadth-First is common (balance server load)
- Parallel crawling is natural
- Variation: Focused crawling
  - Targeting at a subset of pages (e.g., all pages about "automobiles")
  - Typically given a query
- How to find new pages (they may not be linked to an old page!)
- Incremental/repeated crawling
  - Need to minimize resource overhead
  - Can learn from the past experience (updated daily vs. monthly)
  - Target at: 1) frequently updated pages; 2) frequently accessed pages

Component II: Indexer

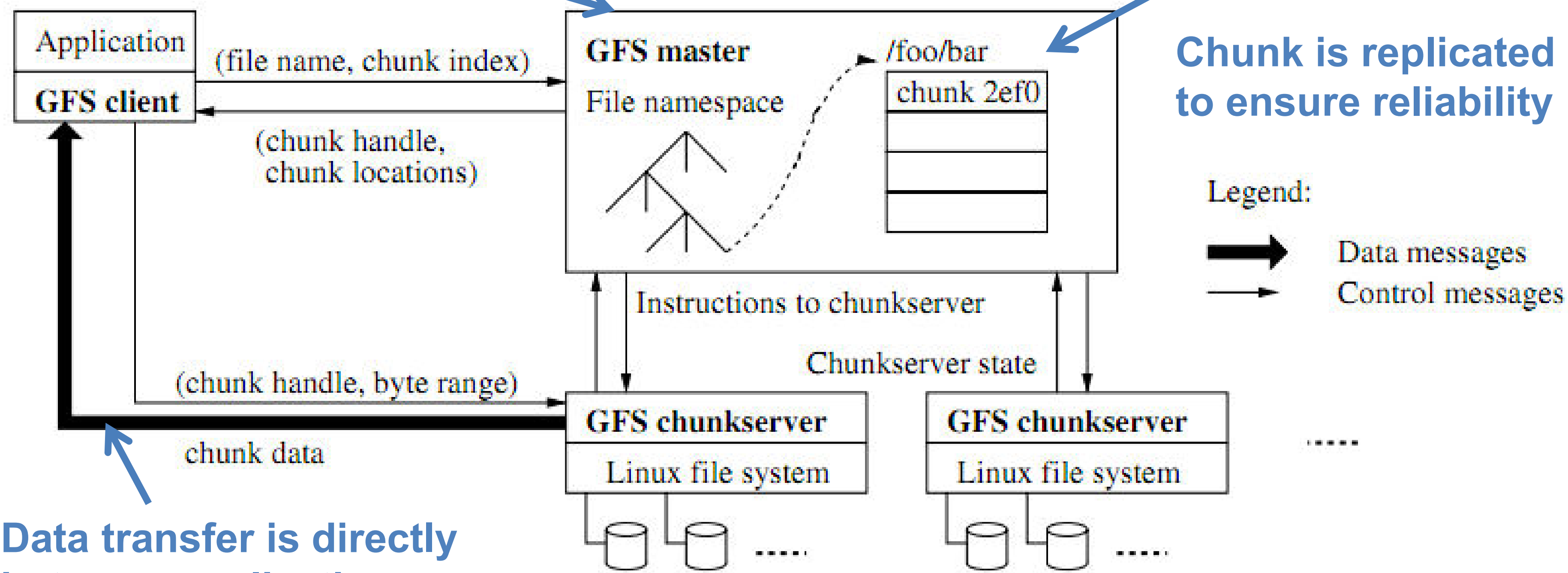
- Standard IR techniques are the basis, but insufficient
  - Scalability
  - Efficiency
- Google's contributions:
  - Google file system (GFS): distributed file system
  - MapReduce: Software framework for parallel computation
  - Hadoop: Open source implementation of MapReduce

GFS Architecture

Simple centralized management

Fixed chunk size (64MB)

Chunk is replicated to ensure reliability



Data transfer is directly between application and chunk servers

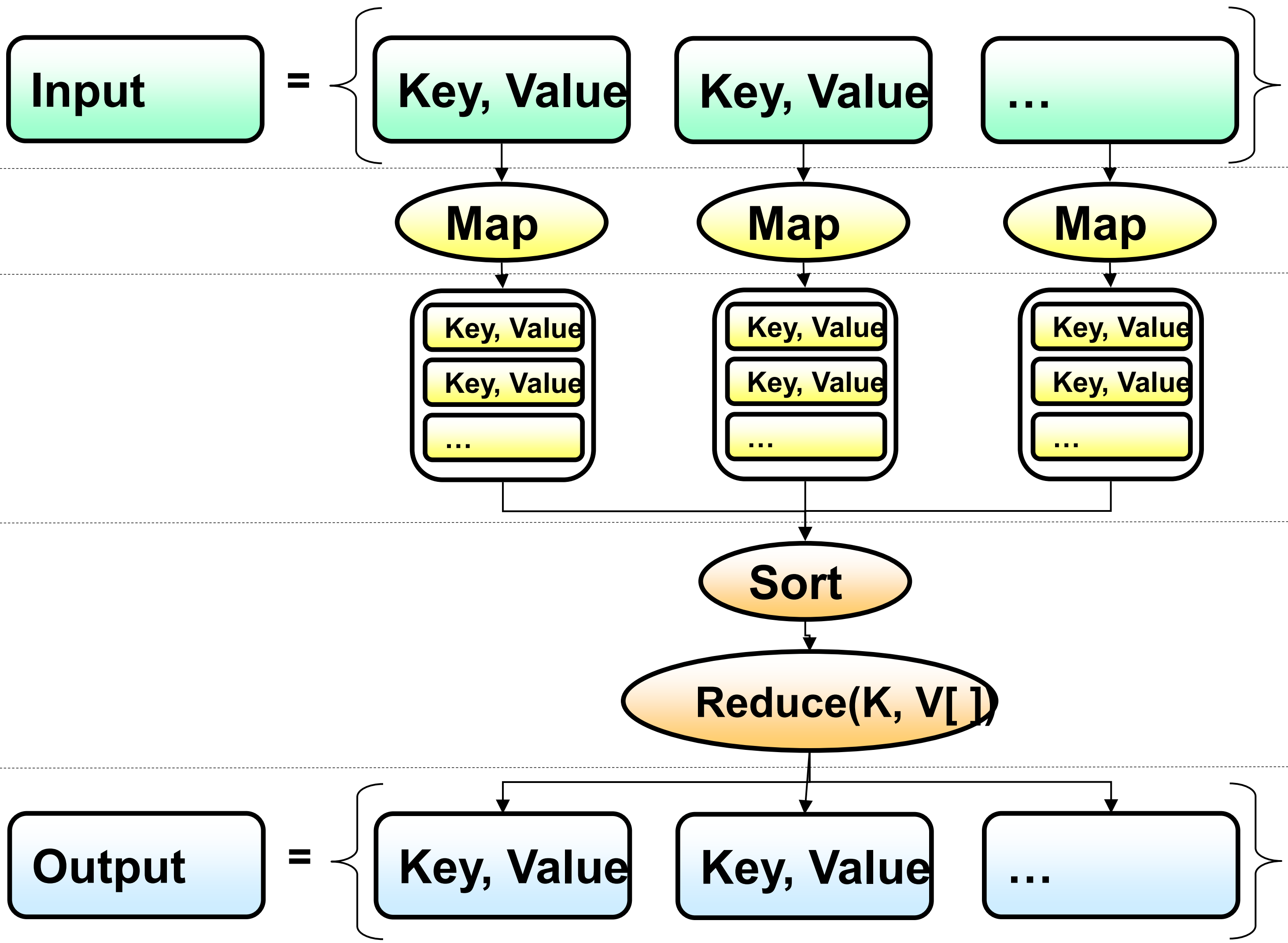
Figure 1: GFS Architecture

http://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf

MapReduce: A Framework for Parallel Programming

- Minimize effort of programmer for simple parallel processing tasks
- Features
  - Hide many low-level details (network, storage)
  - Built-in fault tolerance
  - Automatic load balancing

MapReduce Flow



Split Input into Key-Value pairs.

For each K-V pair call Map.

Each Map produces new set of K-V pairs.

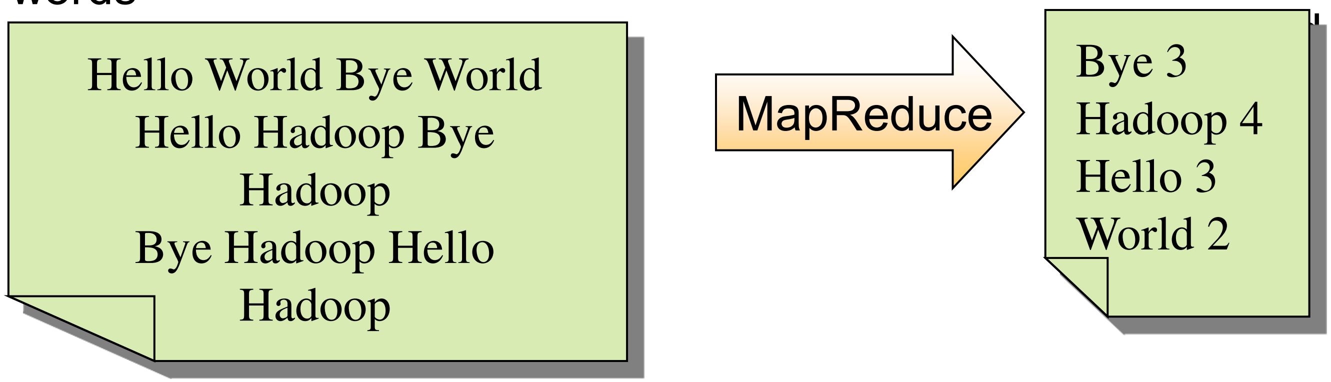
For each distinct key, call reduce. Produces one K-V pair for each distinct key.

Output as a set of Key Value Pairs.

MapReduce WordCount Example

Input:
File containing words

Output:
Number of occurrences



How can we do this within the MapReduce framework?

Basic idea: parallelize on lines in input file!

MapReduce WordCount Example

Input

1, “Hello World Bye World”

2, “Hello Hadoop Bye Hadoop”

3, “Bye Hadoop Hello Hadoop”

Map

Map

Map

Map Output

<Hello,1>
<World,1>
<Bye,1>
<World,1>

<Hello,1>
<Hadoop,1>
<Bye,1>
<Hadoop,1>

<Bye,1>
<Hadoop,1>
<Hello,1>
<Hadoop,1>

```
Map(K, V) {
  For each word w in V
    Collect(w, 1);
}
```

MapReduce WordCount Example

```
Reduce(K, V[ ]) {
  Int count = 0;
  For each v in V
    count += v;
  Collect(K, count);
}
```

Map Output

<Hello,1>
<World,1>
<Bye,1>
<World,1>

<Hello,1>
<Hadoop,1>
<Bye,1>
<Hadoop,1>

<Bye,1>
<Hadoop,1>
<Hello,1>
<Hadoop,1>

Internal Grouping

<Bye → 1, 1, 1>

<Hadoop → 1, 1, 1, 1>

<Hello → 1, 1, 1>

<World → 1, 1>

Reduce

Reduce

Reduce

Reduce

Reduce Output

<Bye, 3>
<Hadoop, 4>
<Hello, 3>
<World, 2>

Inverted Indexing with MapReduce

D1: java resource java class | **D2: java travel resource** | **D3: …**

Map

Key	**Value**
java	**(D1, 2)**
resource	**(D1, 1)**
class	**(D1,1)**

Key	**Value**
java	**(D2, 1)**
travel	**(D2,1)**
resource	**(D2,1)**

Built-In Shuffle and Sort: aggregate values by keys

Reduce

Key	**Value**
java	**{(D1, 2), (D2, 1)}**
resource	**{(D1, 1), (D2,1)}**
class	**{(D1, 1)}**
travel	**{(D2, 1)}**
…	

Slide adapted from Jimmy Lin's presentation

Component III: Retriever

- Standard IR models apply but aren't sufficient
  - Different information need (home page finding vs. topic-driven)
  - Documents have additional information (hyperlinks, markups, URL)
  - Information is often redundant and the quality varies a lot
  - Server-side feedback is often not feasible
- Major extensions
  - Exploiting links (anchor text, link-based scoring)
  - Exploiting layout/markups (font, title field, etc.)
  - Spelling correction
  - Spam filtering
  - Redundancy elimination
- In general, rely on machine learning to combine all kinds of features

Effective Web Retrieval Heuristics

- High accuracy in home page finding can be achieved by
  - Matching query with the title
  - Matching query with the anchor text
  - Plus URL-based or link-based scoring (e.g. PageRank)
- Imposing a conjunctive ("and") interpretation of the query is often appropriate
  - Queries are generally very short (all words are necessary)
  - The size of the Web makes it likely that at least a page would match all the query words
- Combine multiple features using machine learning

Home/Entry Page Finding Evaluation Results (TREC 2001)

Runid	Baseline	Group	Struct.	URLtext	Links	MRR	%top10	%fail
tnout10epCAU	tnout10epCU	tno/utwente	-	Y	Y	0.774	88.3	4.8
tnout10epCU		tno/utwente	-	Y	-	0.772	87.6	4.8
jscbtawep2		Justsystem	Y	Y	Y	0.769	83.4	9.0
jscbtawep1		Justsystem	Y	Y	Y	0.754	83.4	9.0
jscbtawep4		Justsystem	Y	Y	Y	0.752	83.4	8.3
jscbtawep3		Justsystem	Y	Y	Y	0.746	83.4	9.0
yehp01	yehpb01	Yonsei	Y	Y	Y	0.669	76.6	22.1
yehpb01		Yonsei	Y	Y	-	0.659	75.9	22.8
UniNEep1		Neuchatel	-	Y	-	0.637	69.0	8.3
UniNEep2		Neuchatel	-	Y	-	0.637	69.0	7.6
IBMHOMER	IBMHOMENR	ibm-web	Y	-	Y	0.611	77.9	10.3
flabxeall		Fujitsu	-	-	Y	0.599	80.7	9.7
csiro0awh2		CSIRO	-	-	Y	0.593	71.7	21.4
iit01stb	iit01st	IIT	Y	Y	Y	0.578	66.9	24.8
iit01st		IIT	Y	Y	-	0.559	62.8	29.7
UniNEep3		Neuchatel	-	Y	-	0.530	68.3	6.9
VTEP	VTBASE	VT	-	Y	Y	0.506	68.3	15.9
msrcnp2	msrcnp1	microsoft-china	Y	Y	Y	0.505	69.0	15.2
csiro0awh1	csiro0awh3	CSIRO	Y	Y	Y	0.498	72.4	11.0
UniNEep4		Neuchatel	-	Y	-	0.477	68.3	11.0
msrcnp1		microsoft-china	Y	Y	-	0.424	65.5	13.1
flabxe75a		Fujitsu	Y	Y	Y	0.399	55.9	37.9
ok10wahd1	ok10whd1	microsoft	-	Y	Y	0.387	64.1	13.1
IBMHOMENR		ibm-web	Y	-	-	0.382	62.1	11.7
flabxemerge		Fujitsu	Y	Y	Y	0.365	51.0	33.8
flabxet256		Fujitsu	Y	-	Y	0.363	50.3	33.8
ok10wahd0	ok10whd0	microsoft	-	Y	Y	0.362	62.1	13.1
ok10whd1		microsoft	-	Y	-	0.340	60.7	15.9
tnout10epC		tno/utwente	-	-	-	0.338	58.6	18.6
tnout10epA		tno/utwente	-	-	Y	0.331	48.3	35.9
ok10whd0		microsoft	-	Y	-	0.312	58.6	15.2
apl10ha		apl-jhu	-	-	-	0.238	44.8	22.1
ichp2		imperial	-	-	-	0.237	44.8	29.7
apl10hb		apl-jhu	-	-	-	0.220	42.8	21.4
ichp1	ichp2	imperial	-	-	Y	0.208	33.8	37.2
kuhpf2001		kasetsart	-	-	-	0.191	36.6	42.1
PDWTEPDR		padova	-	-	-	0.189	33.8	42.8
PDWTEPWL	PDWTEPDR	padova	-	-	Y	0.178	30.3	42.8
VTBASE		VT	-	-	-	0.126	24.1	45.5
ajouai0102		ajou	-	-	-	0.101	23.4	49.7
ajouai0104		ajou	-	-	Y	0.100	23.4	49.7
PDWTEPTL	PDWTEPDR	padova	-	-	Y	0.099	20.0	42.8
PDWTEPPR		padova	-	-	-	0.054	13.1	44.8

MRR	%top10	%fail
0.774	88.3	4.8
0.772	87.6	4.8

Unigram Query Likelihood
+ Link/URL prior
i.e., p(Q|D) p(D)

[Kraaij et al. SIGIR 2002]

Exploiting anchor text or links or both

0.382 62.1 11.7

0.338 58.6 18.6

Query example:
Haas Business School

How can we Combine Many Features? (Learning to Rank)

- General idea:
  - Given a query-doc pair (Q,D), define various kinds of features Xi(Q,D)
  - Examples of features: the number of overlapping terms, BM25 score of Q and D, p(Q|D), PageRank of D, p(Q|Di), where Di may be anchor text or big font text, "does the URL contain '~'?"…
  - Hypothesize $p(R=1|Q,D)=s(X1(Q,D),\ldots,Xn(Q,D), \lambda)$ where λ is a set of parameters
  - Learn λ by fitting function s with training data (i.e., 3-tuples like (D, Q, 1) (D is relevant to Q) or (D, Q, 0) (D is non-relevant to Q))

Regression-Based Approaches

Logistic Regression: Xi(Q,D) is feature; β's are parameters

$$log \frac{p(R=1|Q,D)}{1-p(R=1|Q,D)} = \beta_0 + \sum_{i=1}^{n} \beta_i X_i$$

$$p(R=1|Q,D) = \frac{1}{1+\exp(-\beta_0 - \sum_{i=1}^{n} \beta_i X_i)}$$

Estimate β's by maximizing the likelihood of training data

	X1(Q,D)	X2(Q,D)	X3(Q,D)
	BM25	PageRank	BM25Anchor
D1 (R = 1)	0.7	0.11	0.65
D2 (R = 0)	0.3	0.05	0.4

$$\mathrm{p}(\{(Q,D_1,1),(Q,D_2,0)\}) = \frac{1}{1+\exp(-\beta_0 - 0.7\beta_1 - 0.11\,\beta_2 - 0.65\,\beta_3)} * (1 - \frac{1}{1+\exp(-\beta_0 - 0.3\,\beta_1 - 0.05\,\beta_2 - 0.4\beta_3)})$$

$$\overrightarrow{\beta^*} = \arg max_{\vec{\beta}}\, p(\{(Q_1,D_{11},R_{11}),(Q_1,D_{12},R_{12}),\ldots,(Q_m,D_{m1},R_{m1}),\ldots\})$$

Once β's are known, we can take Xi(Q,D) computed based on a new query and a new document to generate a score for D w.r.t. Q.

More Advanced Learning Algorithms

- Attempt to directly optimize a retrieval measure (e.g. MAP, nDCG)
  - More difficult as an optimization problem
  - Many solutions were proposed [Liu 09]
- Can be applied to many other ranking problems beyond search
  - Recommender systems
  - Computational advertising
  - Summarization

Machine Learning in Text Retrieval

- Machine learning has been applied to text retrieval since many decades ago
- Recent use of machine learning is driven by
  - Large-scale training data available
  - Need for combining many features
  - Need for robust ranking (again spams)
- Modern Web search engines all use some kind of ML technique to combine many features to optimize ranking
- Learning to rank is still an active research topic

Questions?

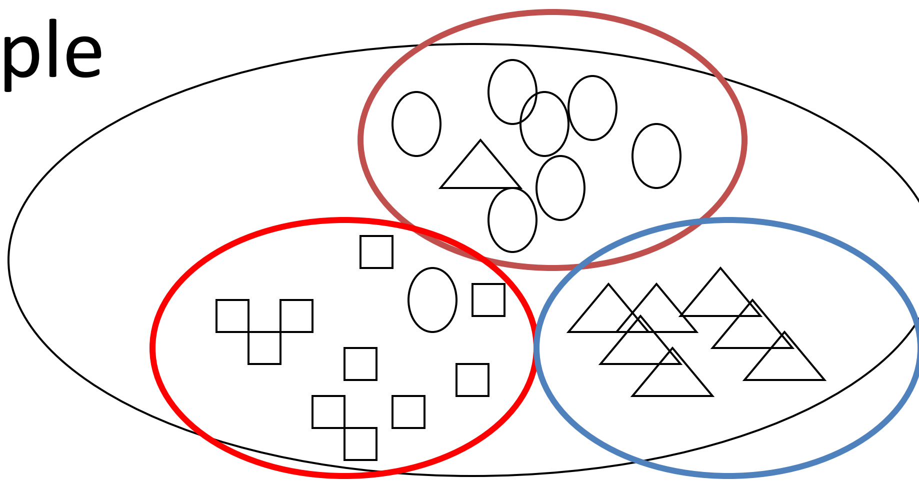
Clustering

Overview

- What is text clustering?
- Why text clustering?
- How to do text clustering?
- How to evaluate clustering results?

The Clustering Problem

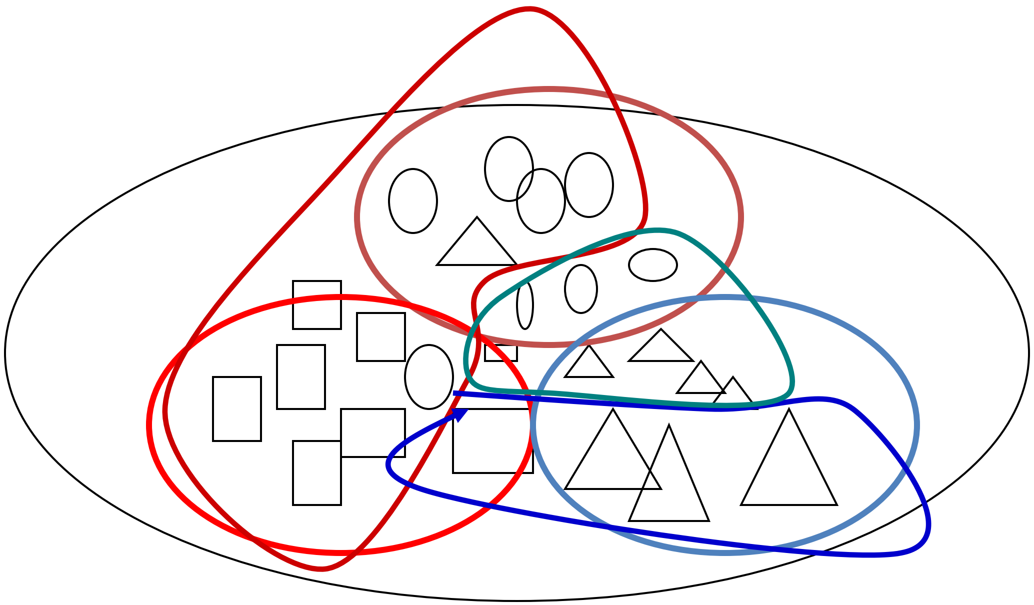
- Discover “natural structure”
- Group similar objects together
- Objects can be documents, terms, passages, websites, …
- Example



Not well defined!   What does similar mean?

The "Clustering Bias"

- Any two objects can be similar, depending on how you look at them!
- Are "car" and "horse" similar?
- A user must define the **perspective** (i.e., a **bias**) for assessing similarity!



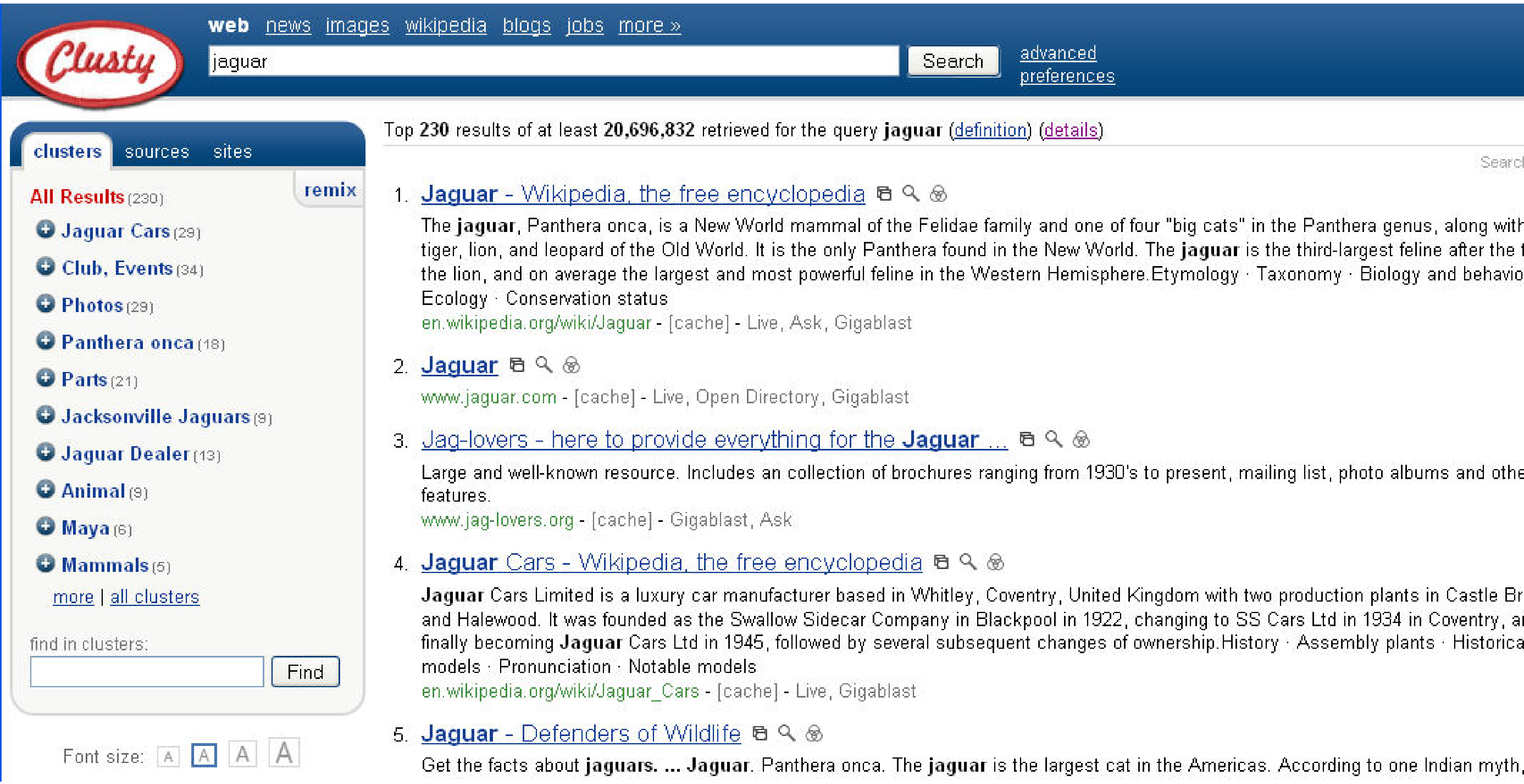
Examples of Text Clustering

- Clustering of documents in the whole collection
- Term clustering to define “concept” or “theme”
- Clustering of passages/sentences or any selected text segments from larger text objects
- Clustering of websites (text objects has multiple documents)
- Text clusters can be further clustered to generate a hierarchy

Why Text Clustering?

- In general, very useful for text mining and exploratory text analysis
  - Get a sense about the overall content of a collection (e.g., what are some of the "typical"/representative documents in a collection?)
  - Link (similar) text objects (e.g., removing duplicated content)
  - Create a structure on the text data (e.g., for browsing)
  - As a way to induce additional features (i.e., clusters) for classification of text objects
- Examples of applications:
  - Clustering of search results
  - Understanding major complaints in emails from customers

Clustering search results



Overview

- What is text clustering?
- Why text clustering?
- How to do text clustering?
- How to evaluate clustering results?

Similarity-based Clustering: General Idea

- Explicitly define a similarity function to measure similarity between two text objects (i.e., providing “clustering bias”)
- Find an optimal partitioning of data to
  - maximize intra-cluster similarity
  - minimize inter-cluster similarity
- Two strategies of obtaining optimal clustering
  - Progressively construct a hierarchy of clusters (hierarchical clustering)
    - Bottom-up (agglomerative): gradually group similar objects into larger clusters
    - Top-down (divisive): gradually partition the data into smaller clusters
  - Starting with an initial tentative clustering and iteratively improve it (“flat” clustering, e.g., K-means)

Similarity-based Clustering Methods

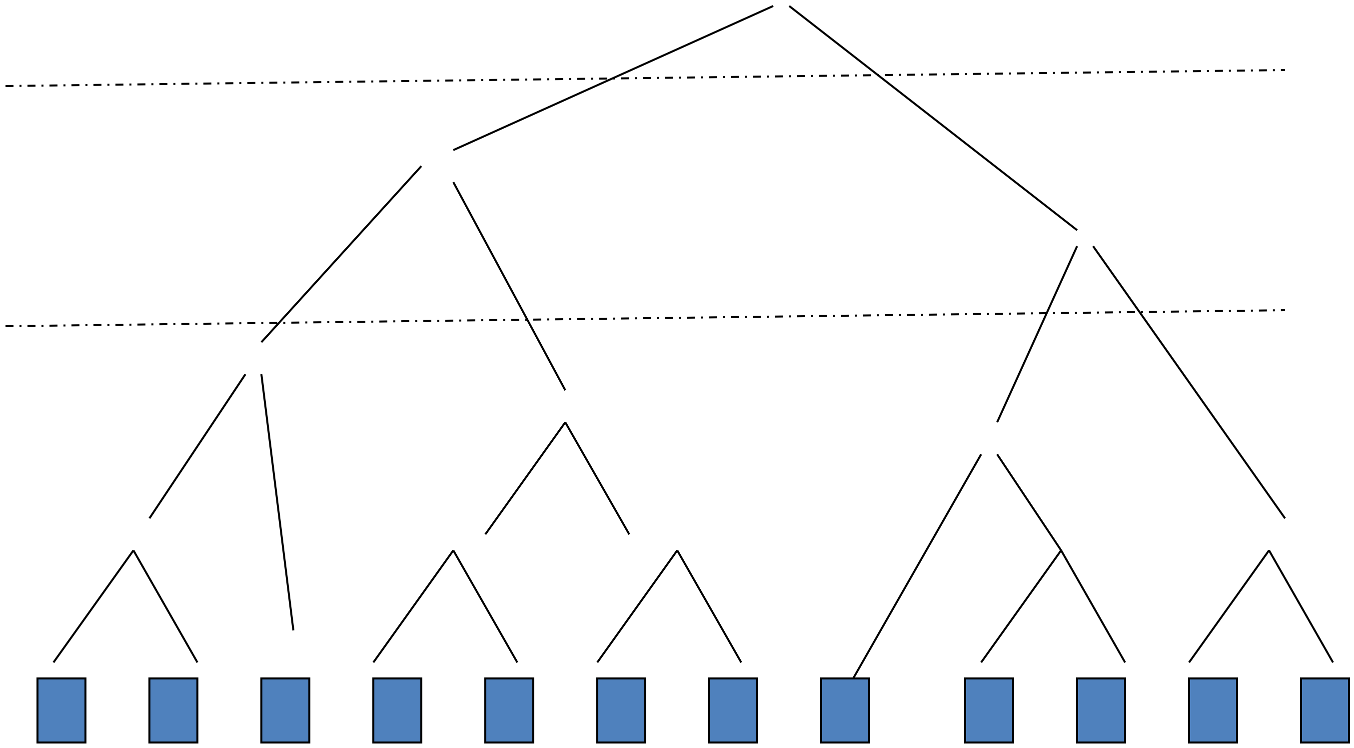
- Many general clustering methods are available

- Two representative methods
  - Hierarchical Agglomerative Cluster (HAC)
  - k-means

Hierarchical Agglomerative Clustering

- Given a similarity function to measure similarity between two objects
- Gradually group similar objects together in a bottom-up fashion to form a hierarchy
- Stop when some stopping criterion is met
- Variations: different ways to compute group similarity based on individual object similarity

Similarity-induced Structure



How to Compute Group Similarity?

Three Popular Methods:

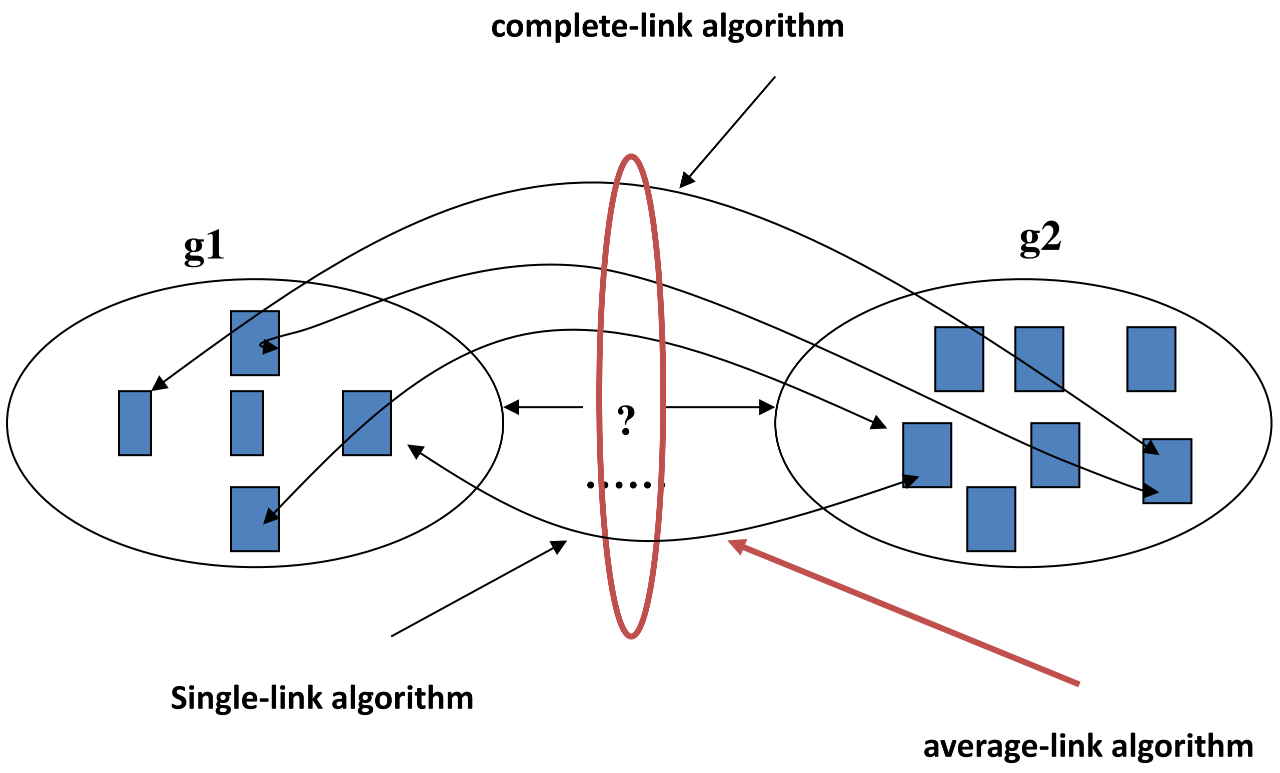
Given two groups g1 and g2,

Single-link algorithm: s(g1,g2)= similarity of the closest pair

Complete-link algorithm: s(g1,g2)= similarity of the farthest pair

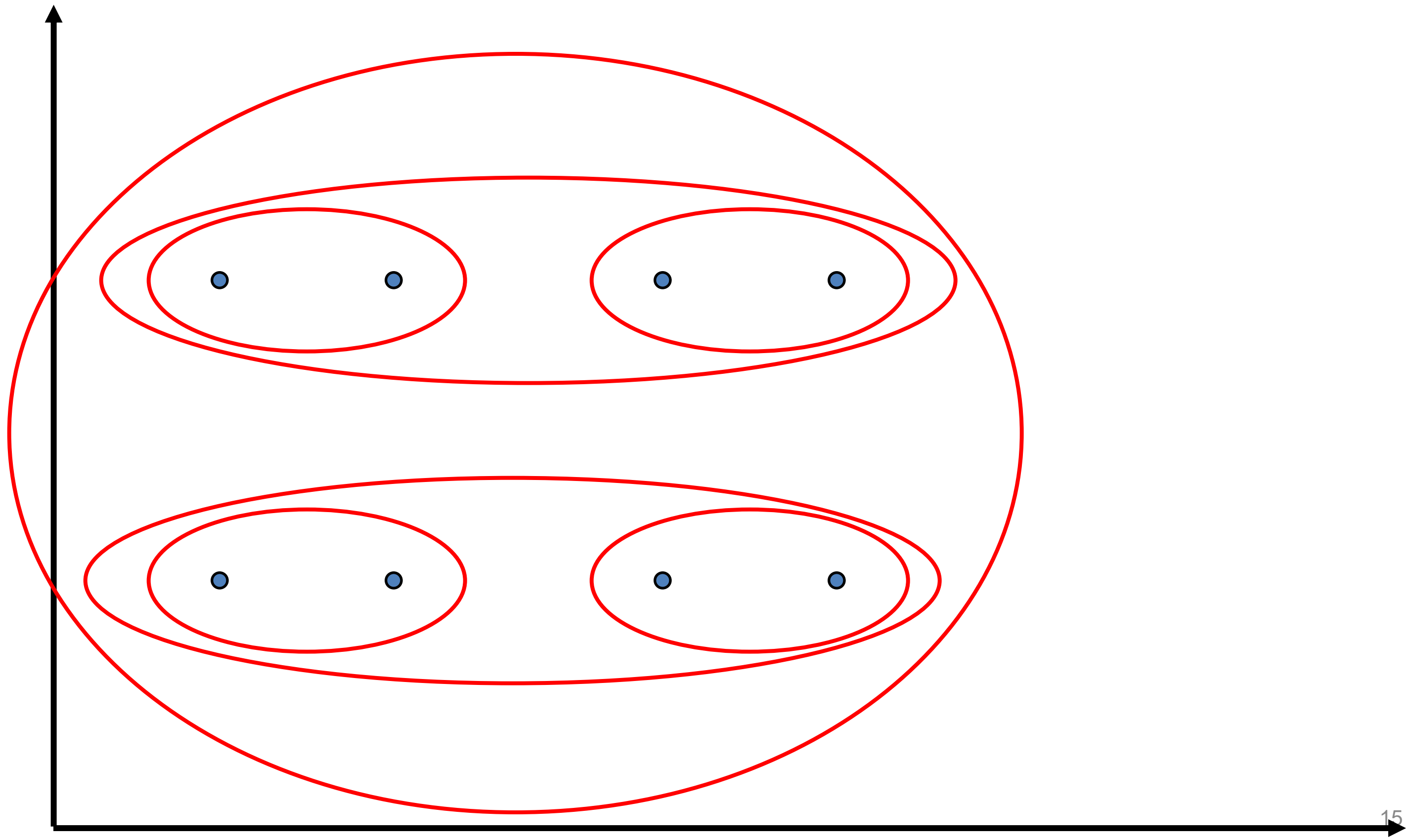
Average-link algorithm: s(g1,g2)= average of similarity of all pairs

Three Methods Illustrated



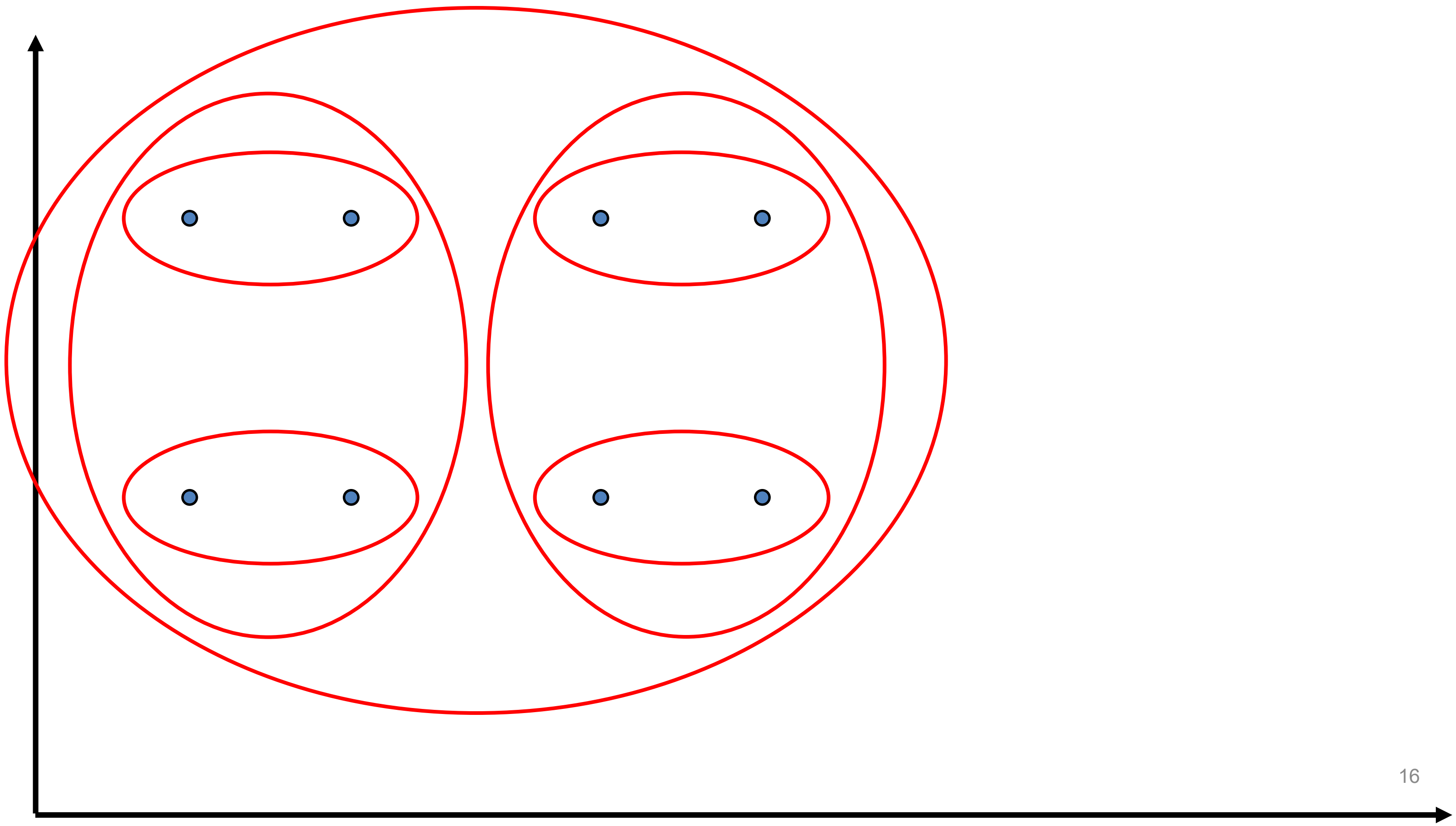
Single Link Algorithm

- Use maximum similarity of pairs: $sim(c_i, c_j) = \max_{x \in c_i, y \in c_j} sim(x, y)$
- Can result in "straggly" (long and thin) clusters due to chaining effect.



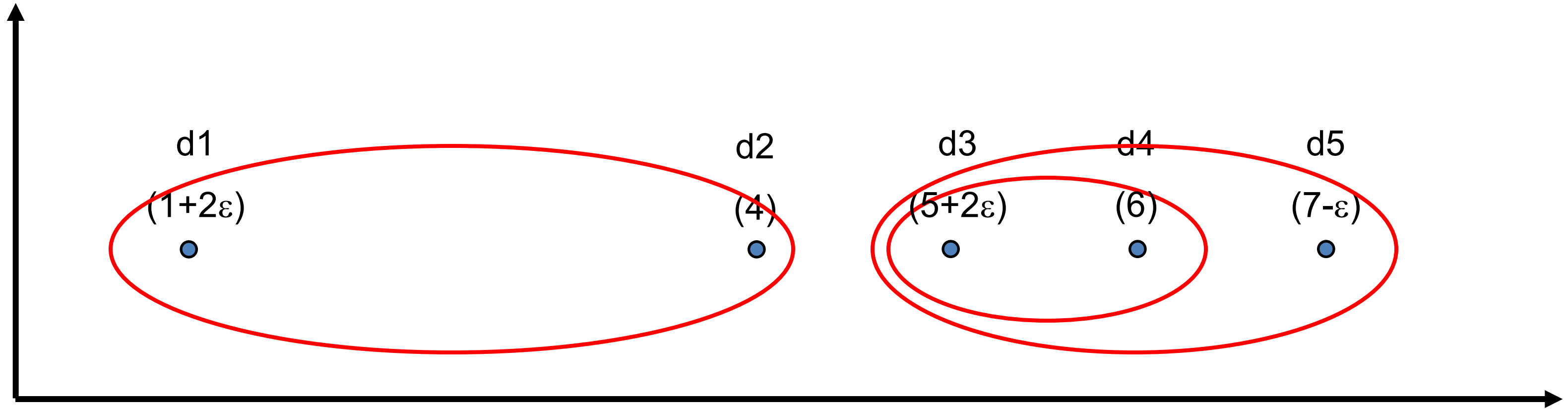
Complete Link Algorithm

- Use minimum similarity of pairs: $sim(c_i, c_j) = \min_{x \in c_i, y \in c_j} sim(x, y)$
- Makes "tighter," spherical clusters that are typically preferable.



Complete Link and Outliers

	d1	d2	d3	d4	d5
d1	**0**	**3-2ε**	**4**	**5-2ε**	**6-3ε**
d2	**3-2ε**	**0**	**1+2ε**	**2**	**3-ε**
d3	**4**	**1+2ε**	**0**	**1-2ε**	**2-3ε**
d4	**5-2ε**	**2**	**1-2ε**	**0**	**1-ε**
d5	**6-3ε**	**3-ε**	**2-3ε**	**1-ε**	**0**



Average Link Algorithm

- Similarity of two clusters = average similarity of all pairs within merged cluster.

$$sim(c_i, c_j) = \frac{1}{\left|c_i \cup c_j\right|\left(\left|c_i \cup c_j\right| - 1\right)} \sum_{\vec{x} \in (c_i \cup c_j)} \sum_{\vec{y} \in (c_i \cup c_j): \vec{y} \neq \vec{x}} sim(\vec{x}, \vec{y})$$

- Compromise between single and complete link.
- Two options:
  - Averaged across all ordered pairs in the merged cluster
  - Averaged over all pairs *between* the two original clusters
- No clear difference in efficacy

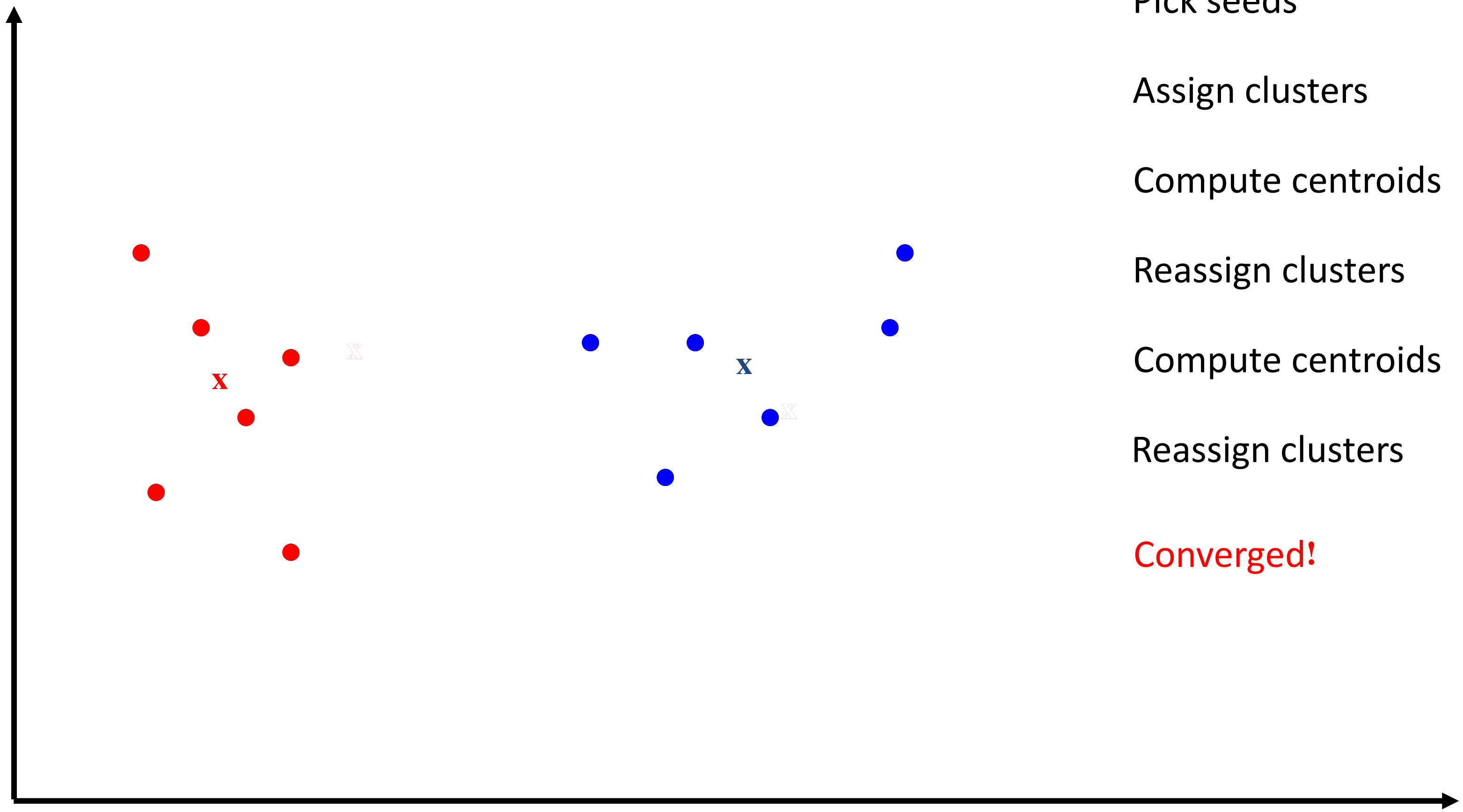
Comparison of the Three Methods

- Single-link
  - "Loose" clusters
  - Individual decision, sensitive to outliers
- Complete-link
  - "Tight" clusters
  - Individual decision, sensitive to outliers
- Average-link
  - "In between"
  - Group decision, insensitive to outliers
- Which one is the best?
  - Depends on what you need!

K-Means Clustering

- Represent each text object as a term vector and assume a similarity function defined on two objects
- Start with k randomly selected vectors and assume they are the centroids of k clusters (initial tentative clustering)
- Assign every vector to a cluster whose centroid is the closest to the vector
- Re-compute the centroid for each cluster based on the newly assigned vectors in the cluster
- Repeat the process until the similarity-based objective function (i.e., within cluster sum of squares) converges (to a local minimum)

K Means Example
(*K*=2)



Pick seeds

Assign clusters

Compute centroids

Reassign clusters

Compute centroids

Reassign clusters

Converged!

Termination conditions

- Several possibilities, e.g.,
  - A fixed number of iterations
  - Doc partition unchanged
  - Centroid positions don’t change

Convergence of *K*-Means

- Why should the *K*-means algorithm ever reach a *fixed point*?
- Define goodness measure of cluster *k* as sum of squared distances from cluster centroid:

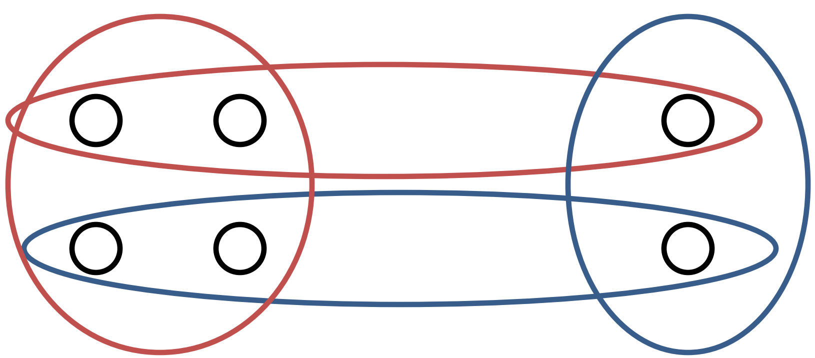
$$RSS_k = \sum_{\vec{x} \in \omega_k} |\vec{x} - \vec{\mu}(\omega_k)|^2$$

$$RSS = \sum_{k=1}^{K} RSS_k$$

- Reassignment monotonically decreases RSS since each vector is assigned to the closest centroid.
- Recomputation monotonically decreases each RSS_k
- *K*-means typically converges quickly

Seed Choice

- Results can vary based on random seed selection.
- Some seeds can result in poor convergence rate, or convergence to sub-optimal clusterings.
  - Exclude outliers from the seed set
  - Try out multiple starting points
  - Initialize with the results of another method.

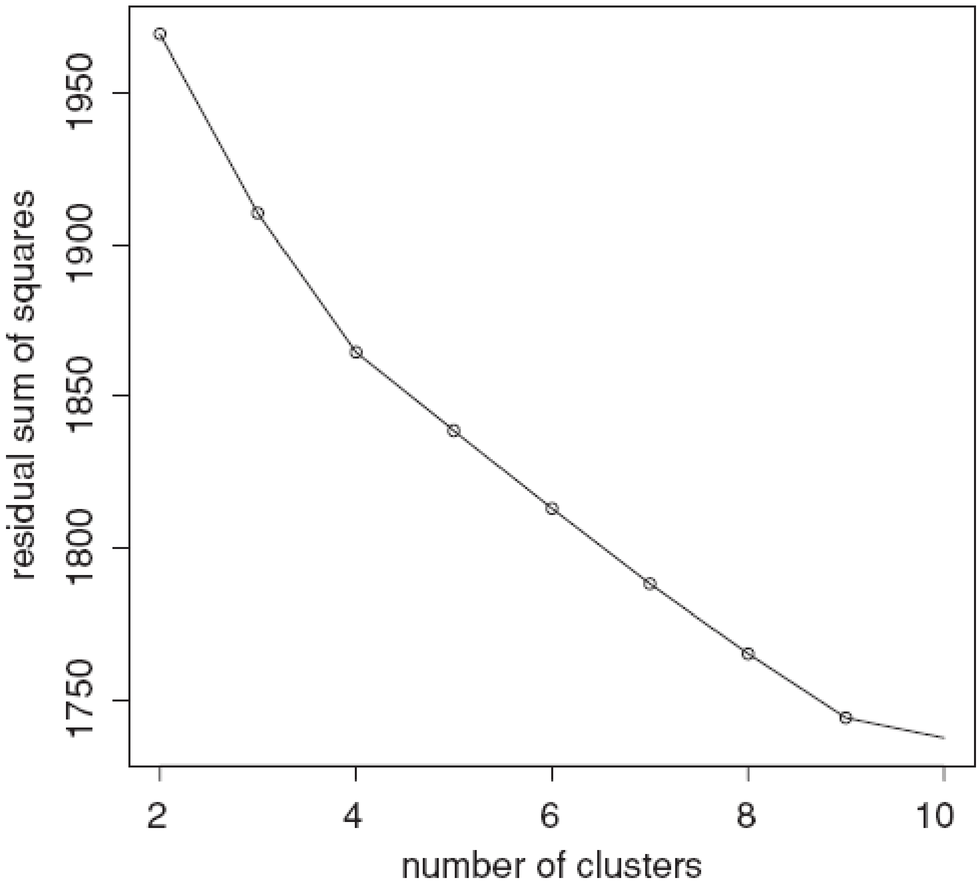


Cluster Cardinality

- Either: Number of clusters K is given.
  - Then partition into K clusters
  - K might be given because there is some external constraint.
- Or: Finding the “right” number of clusters is part of the problem.
  - Given docs, find K for which an optimum is reached.
  - How to define “optimum”?
  - Can we use RSS or average squared distance from centroid?

Cluster Cardinality

- Pick the number of clusters where curve "flattens". Here: 4 or 9.

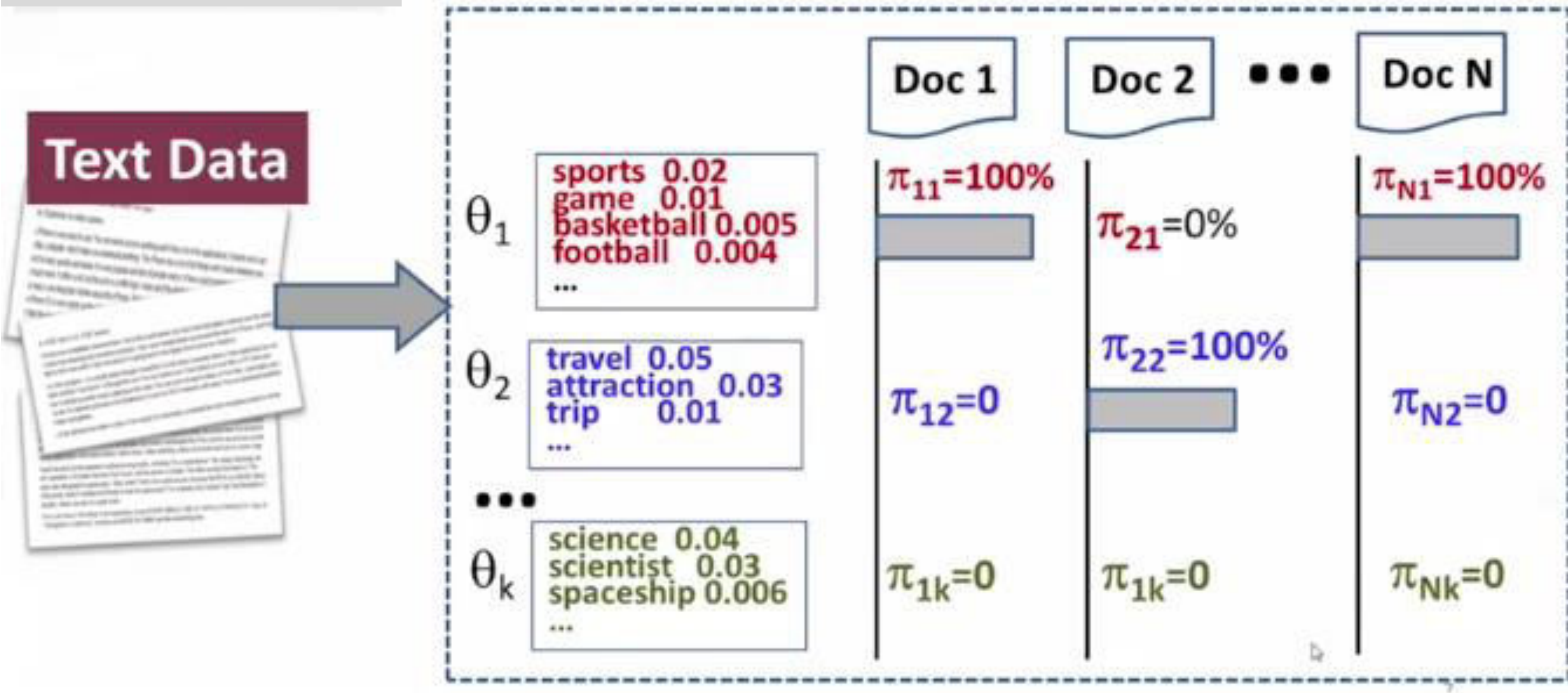


A Generative Model for Clustering

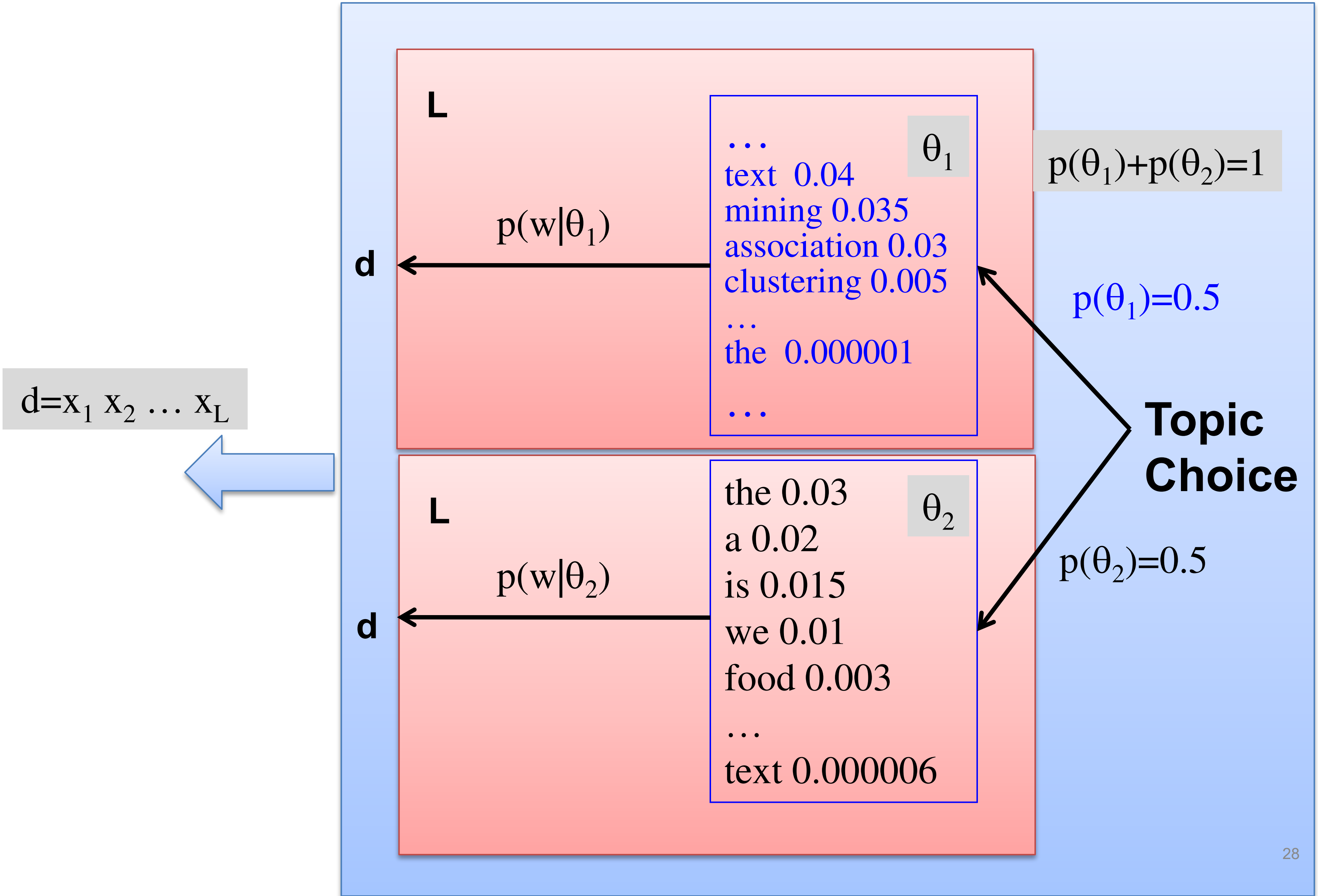
INPUT: C, k, V

OUTPUT: $\{\theta_1, \ldots, \theta_k\}$

$\{c_1, \ldots, c_n\}\ c_i \in [1, k]$



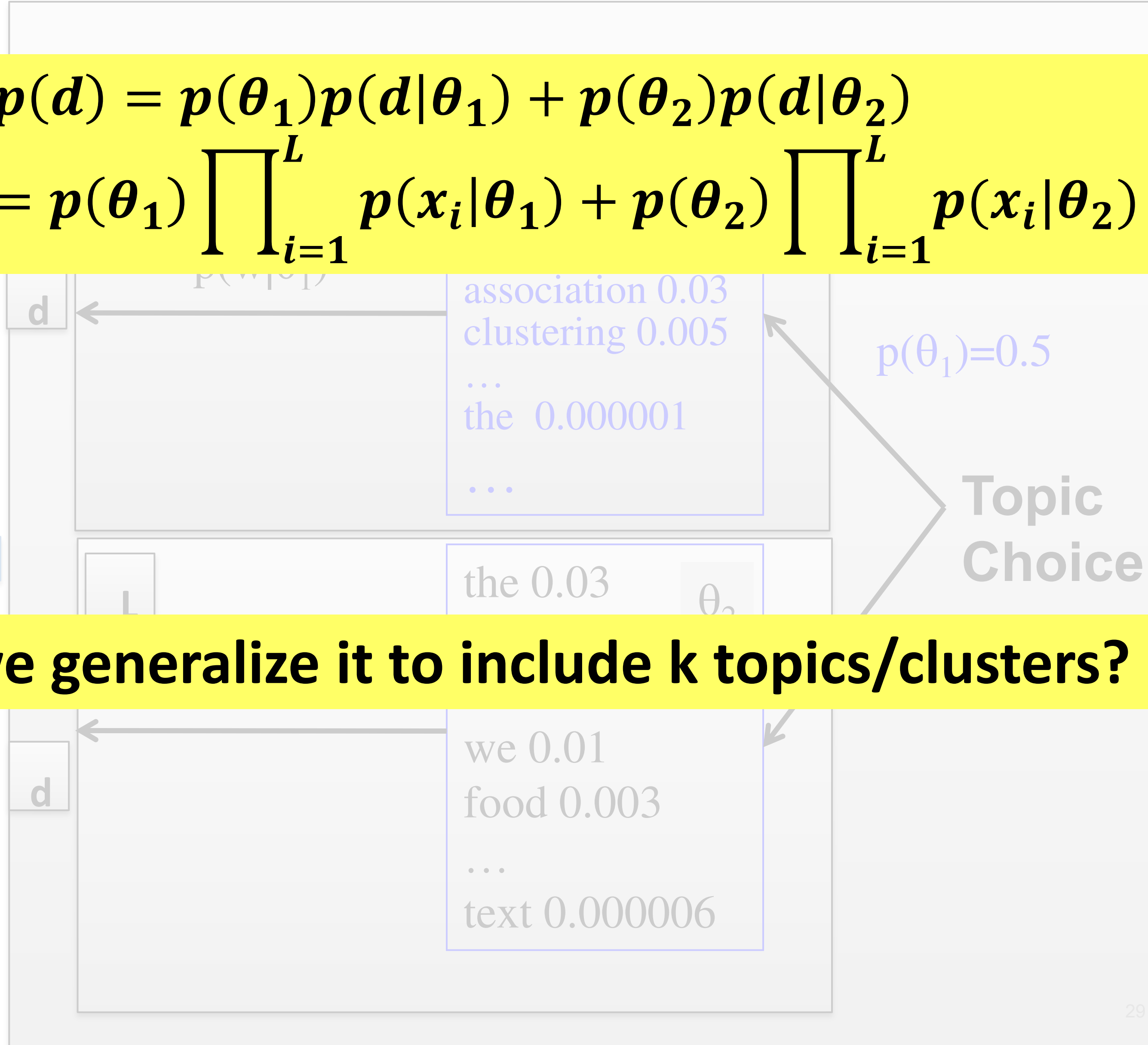
Mixture Model for Document Clustering



Likelihood Function: p(d)=?

$$p(d) = p(\theta_1)p(d|\theta_1) + p(\theta_2)p(d|\theta_2)$$
$$= p(\theta_1)\prod_{i=1}^{L} p(x_i|\theta_1) + p(\theta_2)\prod_{i=1}^{L} p(x_i|\theta_2)$$

$d = x_1\ x_2\ \ldots\ x_L$



How can we generalize it to include k topics/clusters?

Mixture Model for Document Clustering

- Data: a collection of documents C={d_1, …, d_N}
- Model: mixture of k unigram LMs:

$$\Lambda = (\{\theta_i\}; \{p(\theta_i)\}), i \in [1, k]$$

  – To generate a document, first **choose a $\boldsymbol{\theta_i}$** according to $p(\theta_i)$, and then generate **all** words in the document using $p(w|\theta_i)$

- Likelihood:

$$p(d|\Lambda) = \sum_{i=1}^{k} [p(\theta_i) \prod_{j=1}^{|d|} p(x_j|\theta_i)]$$

$$= \sum_{i=1}^{k} [p(\theta_i) \prod_{w \in V} p(w|\theta_i)^{c(w,d)}]$$

- Maximum likelihood estimate

$$\Lambda^* = \arg\max_{\Lambda} \ \mathrm{p}(\mathrm{d}|\Lambda)$$

Cluster Allocation After Parameter Estimation

- **Parameters** of the mixture model:
$$\boldsymbol{\Lambda} = (\{\boldsymbol{\theta_i}\}; \{\boldsymbol{p}(\boldsymbol{\theta_i})\}), \boldsymbol{i} \in [\mathbf{1}, \boldsymbol{k}]$$
  - Each θ_i represents the **content of cluster i: $\boldsymbol{p}(\boldsymbol{w}|\boldsymbol{\theta}_{\mathbf{i}})$**
  - $\boldsymbol{p}(\boldsymbol{\theta_i})$ indicates the **size of cluster i**
- Which cluster should document d belong to? $c_d =?$
  - **Likelihood only**: assign d to the cluster corresponding to the topic θ_i that most likely has been used to generate d
$$c_d = \arg\max_{\mathrm{i}} p(d|\theta_i)$$
  - **Likelihood + prior $\boldsymbol{p}(\boldsymbol{\theta_i})$**: favor large clusters
$$c_d = \arg\max_{\mathrm{i}} p(d|\theta_i)p(\theta_i)$$

How Can We Compute the ML Estimate?

- Data: a collection of documents $C = \{d_1, \dots, d_N\}$
- Model: mixture of k unigram LMs:
$$\Lambda = (\{\theta_i\}; \{p(\theta_i)\}), i \in [1, k]$$
  - To generate a document, first choose a θ_i according to $p(\theta_i)$, and then generate all words in the document using $p(w|\theta_i)$
- Likelihood:
$$p(d|\Lambda) = \sum_{i=1}^{k} [p(\theta_i) \prod_{w \in V} p(w|\theta_i)^{c(w,d)}]$$
$$p(C|\Lambda) = \prod_{j=1}^{N} p(d_j|\Lambda)$$
- Maximum likelihood estimate: $\Lambda^* = \arg\max_{\Lambda} p(C|\Lambda)$

EM Algorithm for Document Clustering

- Initialization: Randomly set
$$\boldsymbol{\Lambda} = (\{\boldsymbol{\theta_i}\}; \{\boldsymbol{p}(\boldsymbol{\theta_i})\}), \boldsymbol{i} \in [\mathbf{1}, \boldsymbol{k}]$$
- Repeat until **likelihood** $\boldsymbol{p}(\boldsymbol{C}|\boldsymbol{\Lambda})$ converges
  - **E-step**: infer which distribution has been used to generate document d: hidden variable $Z_d \in [1, k]$

$p^{(n)}(Z_d = i|d) \propto p^{(n)}(\theta_i) \prod_{w \in V} p^{(n)}(w|\theta_i)^{c(w,d)}$ $\quad \sum_{i=1}^{k} p^{(n)}(Z_d = i|d) = 1$

  - **M-step**: Re-estimation of all parameters

$p^{(n+1)}(\theta_i) \propto \sum_{j=1}^{N} p^{(n)}(Z_{d_j} = i|d_j)$ $\quad \sum_{i=1}^{k} p^{(n+1)}(\theta_i) = 1$

$p^{(n+1)}(w|\theta_i) \propto \sum_{j=1}^{N} c(w, d_j) p^{(n)}(Z_{d_j} = i|d_j)$ $\quad \sum_{i=1}^{k} p^{(n+1)}(w|\theta_i) = 1, \forall i \in [1, k]$

An Example of 2 Clusters

Random Initialization

$$p(\theta_1) = (\theta_2) = 0.5$$

	$p(w\|\theta_1)$	$p(w\|\theta_2)$
text	0.5	0.1
mining	0.2	0.1
medical	0.2	0.75
health	0.1	0.05

E-step **Document d**

	c(w,d)
text	2
mining	2
medical	0
Health	0

Hidden variables:

$$Z_d \in \{1, 2\}$$

$$p(Z_d = 1|d) = \frac{p(\theta_1)p('text'|\theta_1)^2 p('mining'|\theta_1)^2}{p(\theta_1)p('text'|\theta_1)^2 p('mining'|\theta_1)^2 + p(\theta_2)p('text'|\theta_2)^2 p('mining'|\theta_2)^2}$$

$$= \frac{0.5 \times 0.5^2 \times 0.2^2}{0.5 \times 0.5^2 \times 0.2^2 + 0.5 \times 0.1^2 \times 0.1^2} = \frac{100}{101}$$

$$p(Z_d = 2|d) = ?$$

Normalization to Avoid Underflow

	$p(w\|\theta_1)$	$p(w\|\theta_2)$	$p(w\|\bar{\theta})$
text	0.5	0.1	(0.5+0.1)/2
mining	0.2	0.1	(0.2+0.1)/2
medical	0.2	0.75	(0.2+0.75)/2
health	0.1	0.05	(0.1+0.05)/2

Average of $p(w|\theta_i)$ as a possible normalizer

$$p(Z_d = 1|d) = \frac{\dfrac{p(\theta_1)p('text'|\theta_1)^2 p('mining'|\theta_1)^2}{\boxed{p('text'|\bar{\theta})^2 p('mining'|\bar{\theta})^2}}}{\dfrac{p(\theta_1)p('text'|\theta_1)^2 p('mining'|\theta_1)^2}{\boxed{p('text'|\bar{\theta})^2 p('mining'|\bar{\theta})^2}} + \dfrac{p(\theta_1)p('text'|\theta_2)^2 p('mining'|\theta_2)^2}{\boxed{p('text'|\bar{\theta})^2 p('mining'|\bar{\theta})^2}}}$$

An Example of 2 Clusters (cont.)

From E-Step

	$p(Z_d = 1\|d)$
d1	0.9
d2	0.1
d3	0.8

	c('text')	c('mining')
d1	2	3
d2	1	2
d3	4	3

M-Step $p(\theta_1) = ?\quad p(\theta_2) = ?$

	$p(w\|\theta_1)$	$p(w\|\theta_2)$
text	?	?
mining	?	?
medical	?	?
health	?	?

$$p(\theta_1) = \frac{p(Z_{d_1} = 1|d_1) + p(Z_{d_2} = 1|d_2) + p(Z_{d_3} = 1|d_3)}{3}$$

$$= \frac{0.9 + 0.1 + 0.8}{3} = 0.6$$

$$p('text'|\theta_1)$$
$$\propto c('text', d_1) \times p(Z_{d_1} = 1|d_1) + \cdots$$
$$+ c('text', d_3) \times p(Z_{d_3} = 1|d_3)$$
$$= 2 \times 0.9 + 1 \times 0.1 + 4 \times 0.8$$

$$p('mining'|\theta_1) \propto 3 \times 0.9 + 2 \times 0.1 + 3 \times 0.8$$

$$p('text'|\theta_1) + p('mining'|\theta_1) + p('medical'|\theta_1) + p('health'|\theta_1) = 1$$

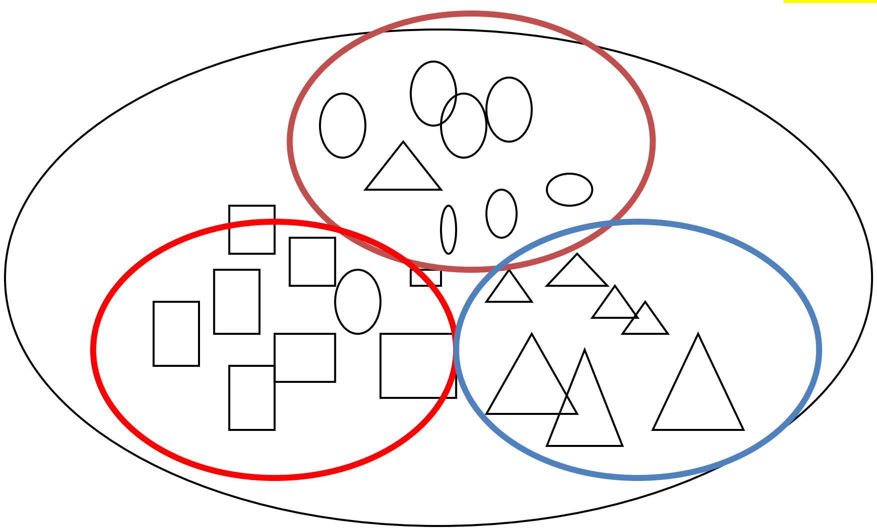
Overview

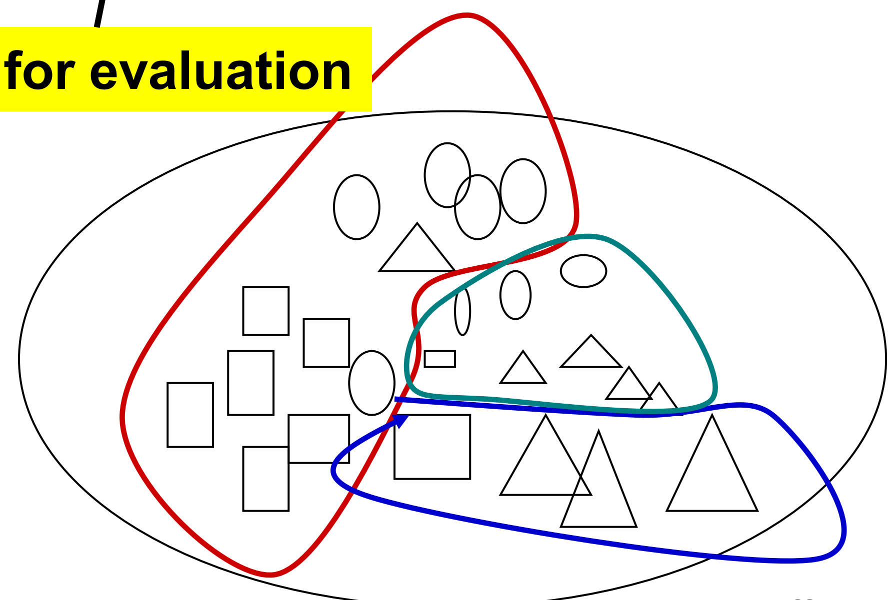
- What is text clustering?
- Why text clustering?
- How to do text clustering?
- How to evaluate clustering results?

The "Clustering Bias"

- Any two objects can be similar, depending on how you look at them!
- Are "car" and "horse" similar?
- A user must define the **perspective** (i.e., a **bias**) for assessing similarity!

Basis for evaluation





Direct Evaluation of Text Clusters

- Question to answer: How close are the system-generated clusters to the ideal clusters (generated by humans)?
  - "Closeness" can be assessed from multiple perspectives
  - "Closeness" can be quantified
  - "Clustering bias" is imposed by the human assessors
- Evaluation procedure:
  - Given a test set, have humans to create an ideal clustering results (i.e., an ideal partitioning of text objects or "gold standard")
  - Use a system to produce clusters from the same test data
  - Quantify the similarity between the system-generated clusters and the gold standard clusters
  - Similarity can be measured from multiple perspectives

Purity

- Purity: each cluster is assigned to the class which is more frequent in the cluster

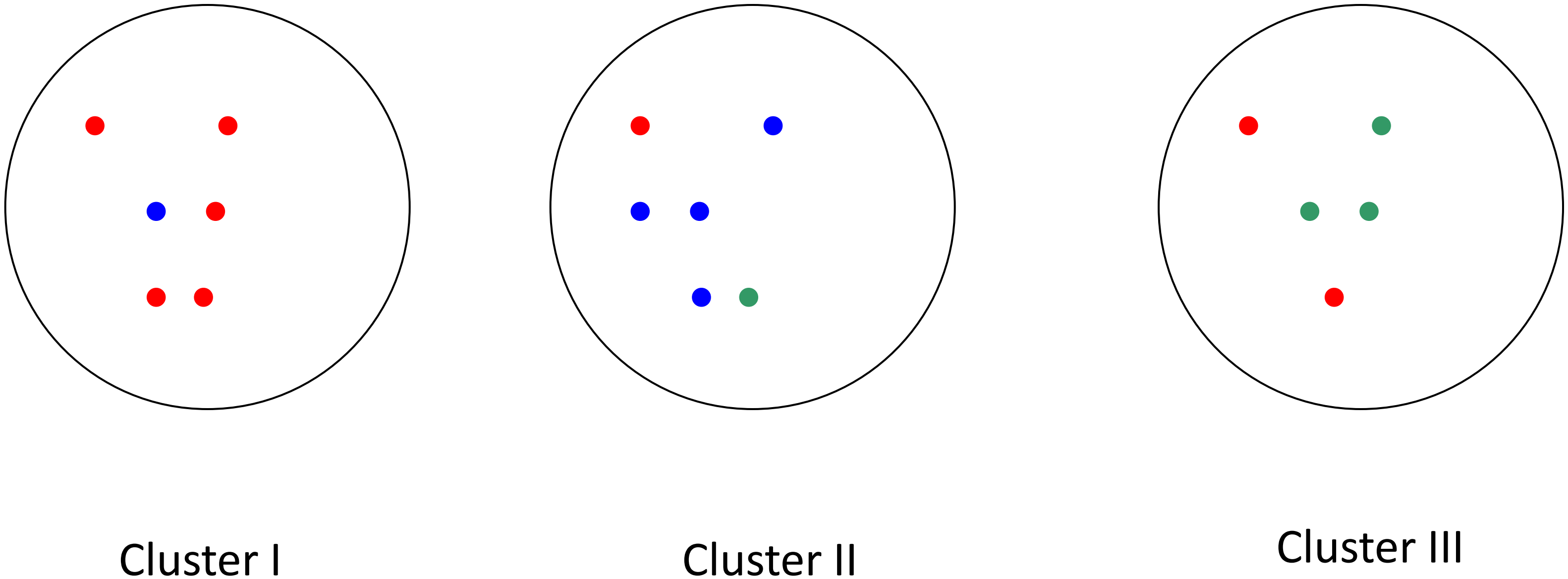
$$Purity(\Omega, C) = \frac{1}{N} \sum_{k} \max_{j} |\omega_k \cap c_j|$$

$$\Omega = \{\omega_1, \omega_2, \ldots, \omega_K\} \text{ set of clusters}$$

$$\mathrm{C} = \{c_1, c_2, \ldots, c_J\} \text{ set of classes}$$

- Biased because having *n* clusters maximizes purity

Purity example



Cluster I: Purity = 1/6 (max(5, 1, 0)) = 5/6

Cluster II: Purity = 1/6 (max(1, 4, 1)) = 4/6

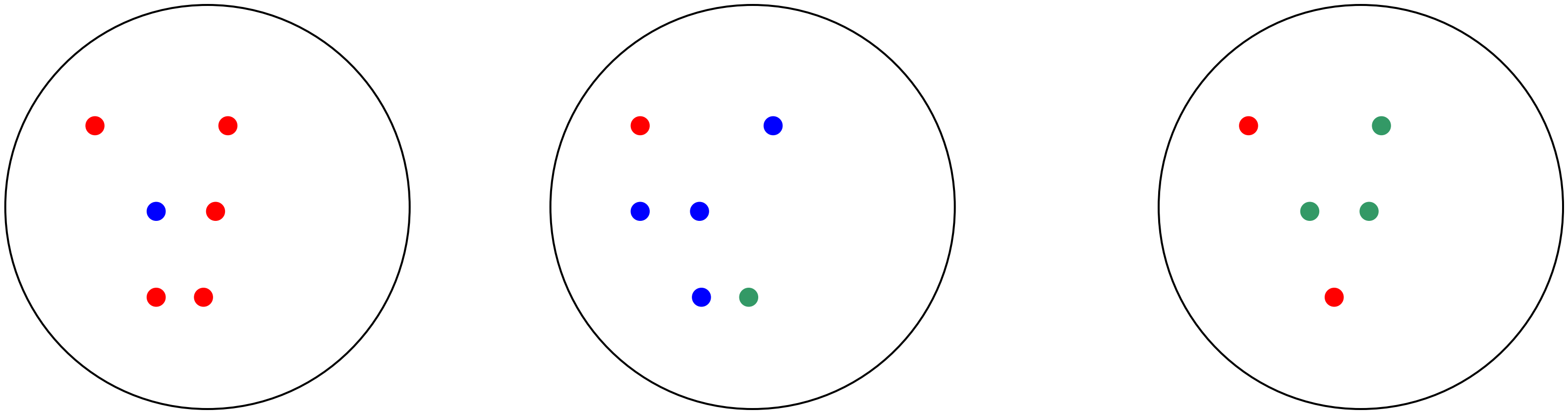
Cluster III: Purity = 1/5 (max(2, 0, 3)) = 3/5

Overall Purity = 1/17 (5 + 4 + 3) = 12/17

Matching-based F-Measure

- Precision: The fraction of points in ω_i from the majority class c_{ji}, (i.e., the same as purity), where ji is the class that contains the maximum # of points from ω_i
- Recall: the fraction of the points in class c_{ji} shared in common with cluster ω_i
- F-measure for ω_i : The harmonic means of precision and recall
- F-measure for clustering C: average of F-measures of all clusters

F-measure example



Cluster I   Cluster II   Cluster III

Cluster I: Precision = 5/6, Recall = 5/8, F1 = 0.714

Cluster II: Precision = 4/6, Recall = 4/5, F1 = 0.727

Cluster III: Precision = 3/5, Recall = 3/4, F1 = 0.667

Overall F1 = 0.703

Normalized Mutual Information (NMI)

- Mutual Information: How much information does the clustering contain about the classification

$$I(\Omega;C) = \sum_k \sum_j p(\omega_k \cap c_j) \log \frac{p(\omega_k \cap c_j)}{p(\omega_k)\,p(c_j)}$$

$$= \sum_k \sum_j \frac{|\omega_k \cap c_j|}{N} \log \frac{N\,|\omega_k \cap c_j|}{|\omega_k|\,|c_j|}$$

- Entropy: Information contained in a distribution

$$H(\Omega) = -\sum_k P(\omega_k) \log P(\omega_k) = -\sum_k \frac{|\omega_k|}{N} \log \frac{|\omega_k|}{N}$$

- Normalized Mutual Information (NMI):

$$NMI(\Omega;C) = \frac{I(\Omega;C)}{[H(\Omega)+H(C)]/2}$$

Rand Index measures between pair decisions. Here RI = 0.68

Number of points	Same Cluster in clustering	Different Clusters in clustering
Same class in ground truth	20	24
Different classes in ground truth	20	72

Four Possibilities for Truth Assignment

- Four possibilities based on the agreement between cluster label and partition label
  - TP: true positive – Two points x_i and x_j belong to the same class c, and they also are in the same cluster ω

    $$TP = \left|\left\{\left(x_i, x_j\right): y_i = y_j \; and \; \widehat{y_i} = \widehat{y_j}\right\}\right|$$

    y_i : the true class label

    $\widehat{y_i}$: the cluster label for point x_i
  - FN: false negative: $FN = \left|\left\{\left(x_i, x_j\right): y_i = y_j \; and \; \widehat{y_i} \neq \widehat{y_j}\right\}\right|$
  - FP: false positive: $FP = \left|\left\{\left(x_i, x_j\right): y_i \neq y_j \; and \; \widehat{y_i} = \widehat{y_j}\right\}\right|$
  - TN: true negative: $TN = \left|\left\{\left(x_i, x_j\right): y_i \neq y_j \; and \; \widehat{y_i} \neq \widehat{y_j}\right\}\right|$

Rand index and Cluster Pairwise F-measure

$$RI = \frac{TP + TN}{TP + FP + TN + FN}$$

Compare with standard Precision and Recall:

$$P = \frac{TP}{TP + FP} \qquad R = \frac{TP}{TP + FN}$$

People also define and use a cluster F-measure, which is probably a better measure.

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

Indirect Evaluation of Text Clusters

- Question to answer: How useful are the clustering results for the intended applications?
  - "usefulness" is inevitably application specific
  - "clustering bias" is imposed by the intended application
- Evaluation procedure:
  - Create a test set for the intended application to quantify the performance of any system for this application
  - Choose a baseline system to compare with
  - Add a clustering algorithm to the baseline system → "clustering system"
  - Compare the performance of the clustering system and the baseline in terms of any performance measure for the application

Questions?