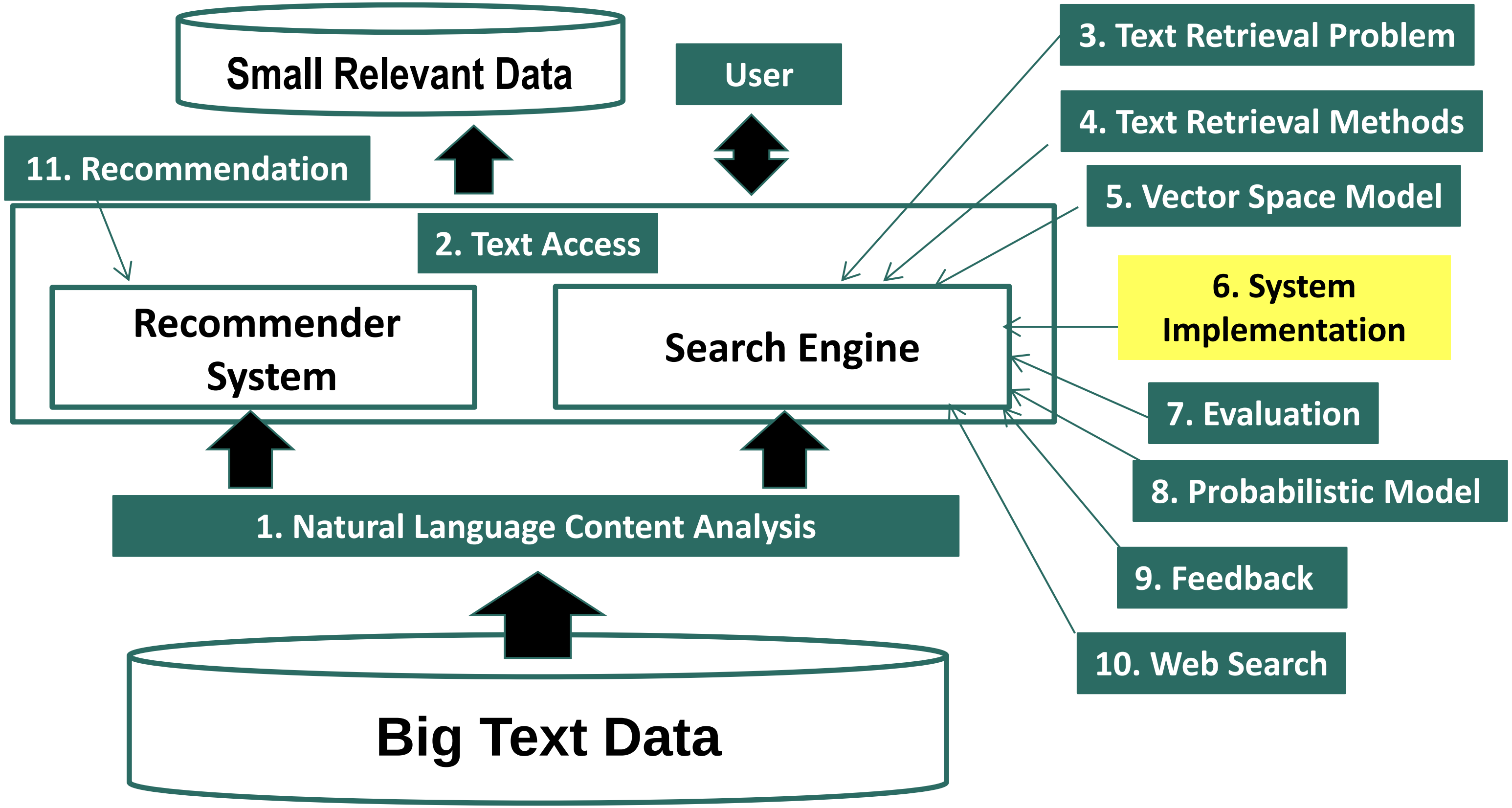
Text Retrieval and Search Engines

System Implementation: Fast Search

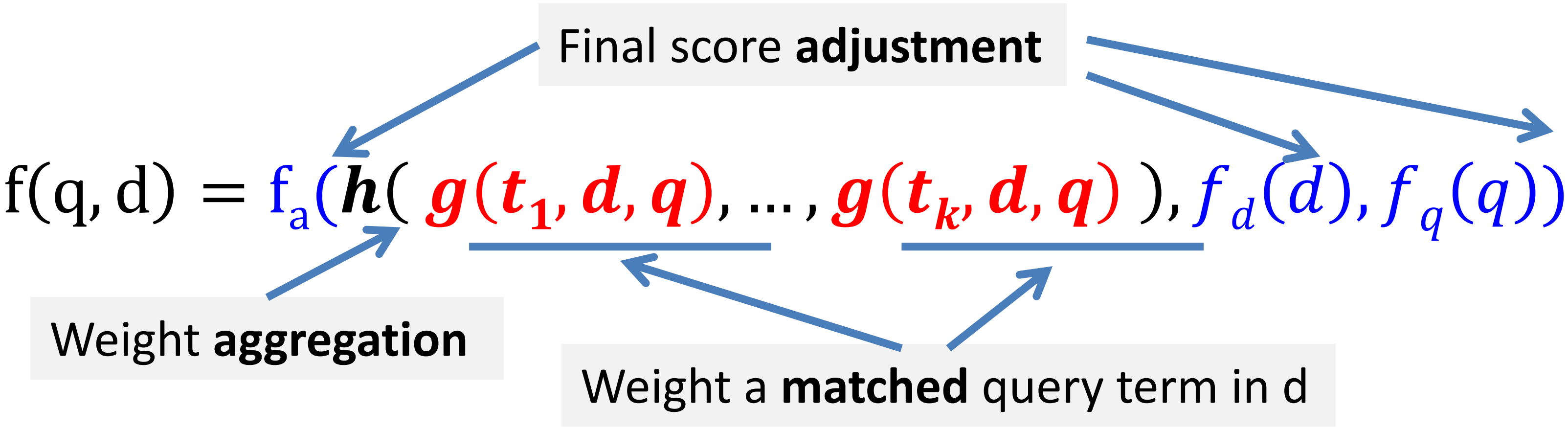
ChengXiang "Cheng" Zhai
Department of Computer Science
University of Illinois at Urbana-Champaign

System Implementation: Fast Search



How to Score Documents Quickly

General Form of Scoring Function



Final score **adjustment**

$$f(q,d) = f_a(\boldsymbol{h}(\ \boldsymbol{g}(\boldsymbol{t_1},\boldsymbol{d},\boldsymbol{q}), \ldots, \boldsymbol{g}(\boldsymbol{t_k},\boldsymbol{d},\boldsymbol{q})\ ), f_d(d), f_q(q))$$

Weight **aggregation**

Weight a **matched** query term in d

A General Algorithm for Ranking Documents

$$\mathrm{f}(\mathrm{q},\mathrm{d}) = \mathrm{f}_{\mathrm{a}}(\boldsymbol{h}(\,\boldsymbol{g}(\boldsymbol{t}_{\mathbf{1}},\boldsymbol{d},\boldsymbol{q}),\ldots,\boldsymbol{g}(\boldsymbol{t}_{\boldsymbol{k}},\boldsymbol{d},\boldsymbol{q})\,),f_{d}(d),f_{q}(q))$$

- $f_d(d)$ and $f_q(q)$ are pre-computed
- Maintain a score accumulator for each **d** to compute ***h***
- For each query term $\boldsymbol{t_i}$
  - Fetch the inverted list $\{(d_1,f_1),\ldots,(d_n,f_n)\}$
  - For each entry (d_j,f_j), compute **$g(t_i,d_j,q)$**, and update score accumulator for doc d_i to incrementally compute ***h***
- Adjust the score to compute **f_a**, and sort

An Example: Ranking Based on TF Sum

$\mathbf{f(d,q)=g(t_1,d,q)+\ldots+ g(t_k,d,q)}$ where $g(t_i,d,q) = c(t_i,d)$

Query = **“info security”**

Info: (d1, 3), (d2, 4), (d3, 1), (d4, 5)
Security: (d2, 3), (d4,1), (d5, 3)

		Accumulators:	**d1**	**d2**	**d3**	**d4**	**d5**
			0	0	0	0	0
info	(d1,3)	=>	**3**	0	0	0	0
	(d2,4)	=>	3	**4**	0	0	0
	(d3,1)	=>	3	4	**1**	0	0
	(d4,5)	=>	3	4	1	**5**	0
security	(d2,3)	=>	3	**7**	1	5	0
	(d4,1)	=>	3	7	1	**6**	0
	(d5,3)	=>	3	7	1	6	**3**

Further Improving Efficiency

- Caching (e.g., query results, list of inverted index)
- Keep only the most promising accumulators
- Scaling up to the Web-scale? (need parallel processing)

Some Text Retrieval Toolkits

- Lucene: http://lucene.apache.org/
- Lemur/Indri: http://www.lemurproject.org/
- Terrier: http://terrier.org/
- MeTA: http://meta-toolkit.github.io/meta/
- More can be found at http://timan.cs.uiuc.edu/resources

Summary of System Implementation

- Inverted index and its construction
  - Preprocess data as much as we can
  - Compression when appropriate
- Fast search using inverted index
  - Exploit inverted index to accumulate scores for documents matching a query term
  - Exploit Zipf’s law to avoid touching many documents not matching any query term
  - Can support a wide range of ranking algorithms
- Great potential for further scaling up using distributed file system, parallel processing, and caching

Additional Readings

- Ian H. Witten, Alistair Moffat, Timothy C. Bell: Managing Gigabytes: Compressing and Indexing Documents and Images, Second Edition. Morgan Kaufmann, 1999.
- Stefan Büttcher, Charles L. A. Clarke, Gordon V. Cormack: Information Retrieval - Implementing and Evaluating Search Engines. MIT Press, 2010.