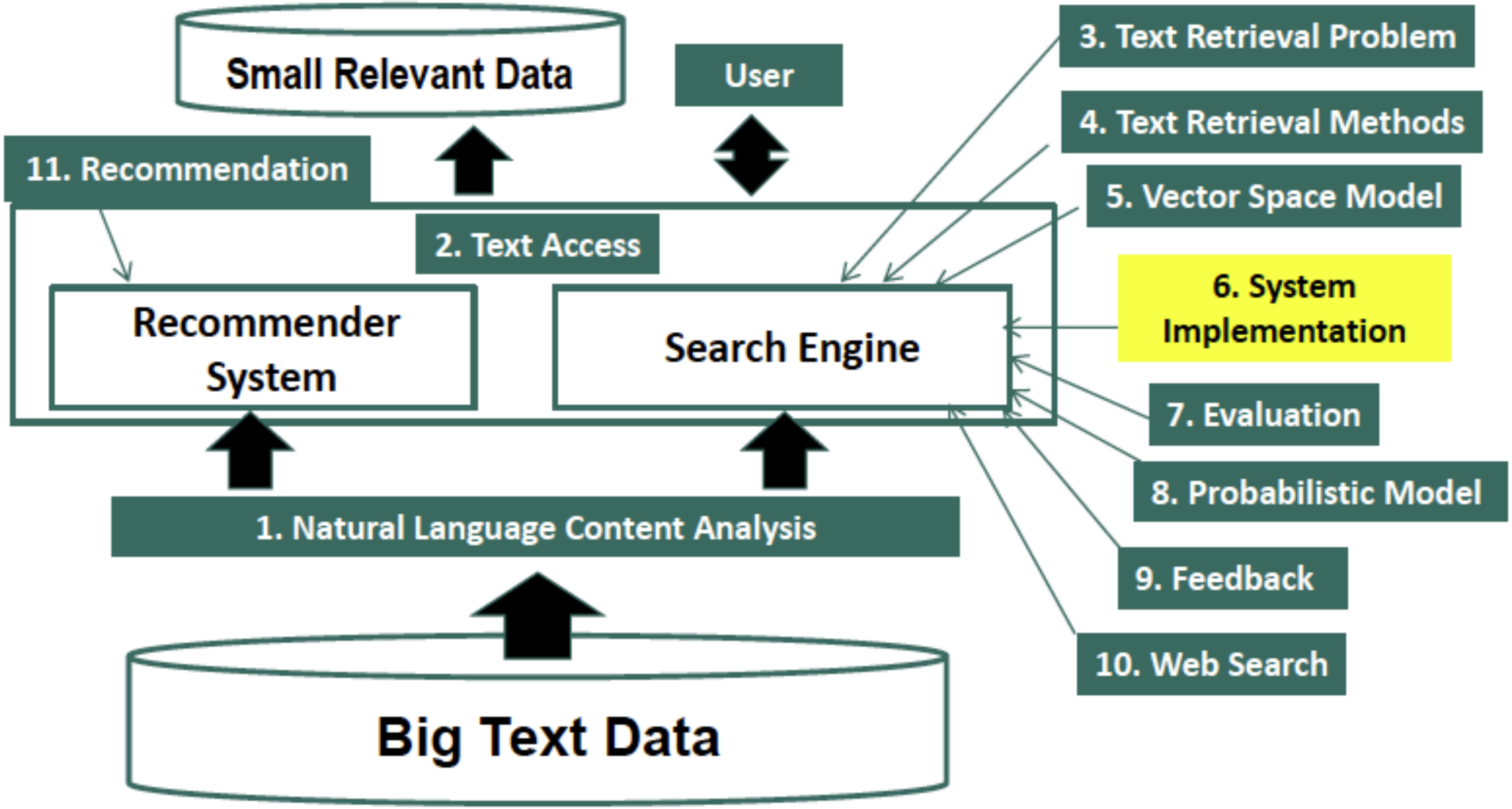
Information Retrieval & Text Mining

System Implementation

Inverted Index Construction

Dr. Saeed Ul Hassan
Information Technology University

Implementation of Text Retrieval Systems



Constructing Inverted Index

- The main difficulty is to build a huge index with limited memory
- Memory-based methods: not usable for large collections
- Sort-based methods:
  - Step 1: Collect local (termID, docID, freq) tuples
  - Step 2: Sort local tuples (to make “runs”)
  - Step 3: Pair-wise merge runs
  - Step 4: Output inverted file

Sort-based Inversion

doc1

doc2

• • •

doc300

Mapping Strings to integer>>

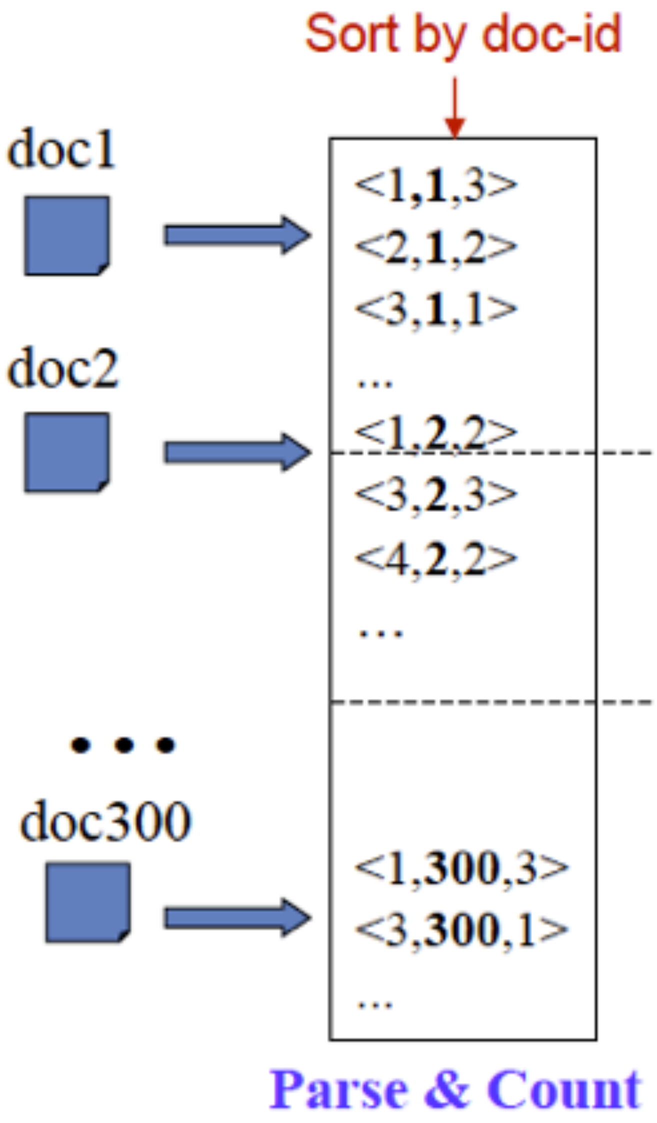
Term Lexicon:

the	1
campaign	2
news	3
a	4
…	

DocID Lexicon:

doc1	1
doc2	2
doc3	3
…	

Sort-based Inversion



Sort by doc-id

doc1

doc2

• • •

doc300

<1,1,3>
<2,1,2>
<3,1,1>
...
<1,2,2>
<3,2,3>
<4,2,2>
…

<1,300,3>
<3,300,1>
...

Parse & Count

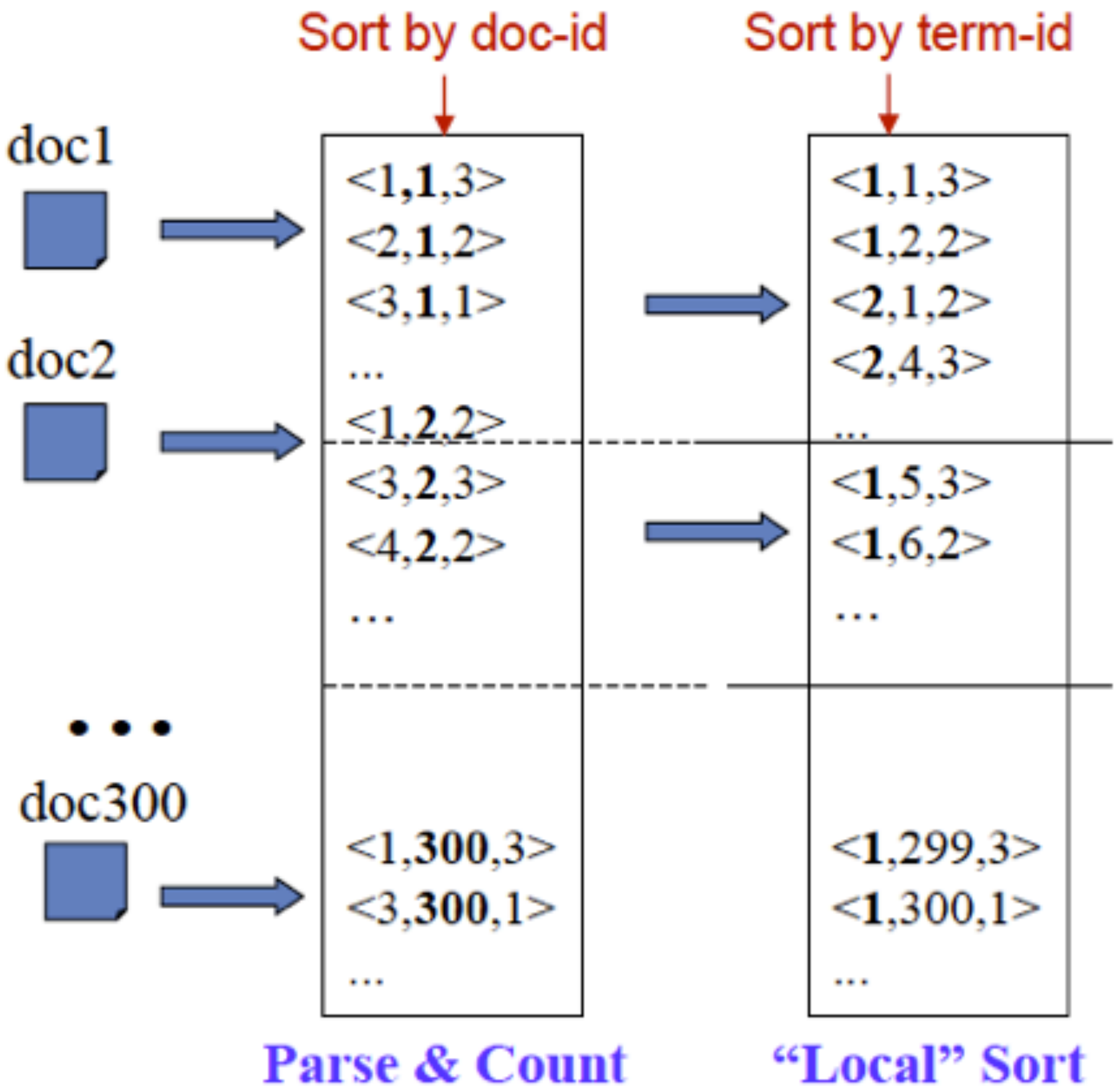
Term Lexicon:

the 1
campaign 2
news 3
a 4
…

DocID Lexicon:

doc1 1
doc2 2
doc3 3
…

Sort-based Inversion



Sort by doc-id

Sort by term-id

doc1

doc2

• • •

doc300

<1,1,3>
<2,1,2>
<3,1,1>
...
<1,2,2>
<3,2,3>
<4,2,2>
…
<1,300,3>
<3,300,1>
...

<1,1,3>
<1,2,2>
<2,1,2>
<2,4,3>
...
<1,5,3>
<1,6,2>
…
<1,299,3>
<1,300,1>
...

Parse & Count

"Local" Sort

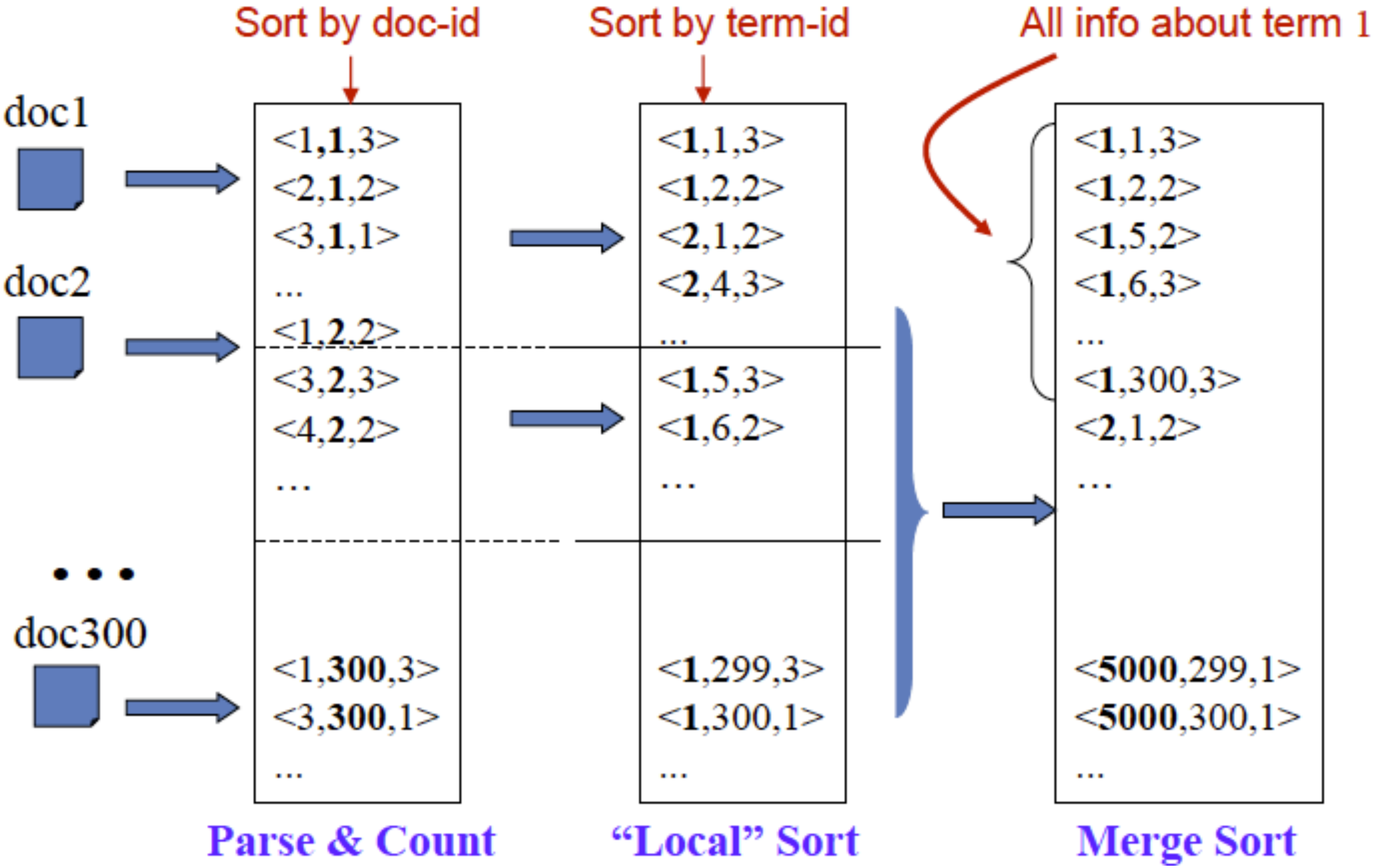
Term Lexicon:

the 1
campaign 2
news 3
a 4
…

DocID Lexicon:

doc1 1
doc2 2
doc3 3
…

Sort-based Inversion



Sort by doc-id

Sort by term-id

All info about term 1

doc1

doc2

• • •

doc300

<1,1,3>
<2,1,2>
<3,1,1>
...
<1,2,2>
<3,2,3>
<4,2,2>
…
<1,300,3>
<3,300,1>
...

<1,1,3>
<1,2,2>
<2,1,2>
<2,4,3>
...
<1,5,3>
<1,6,2>
…
<1,299,3>
<1,300,1>
...

<1,1,3>
<1,2,2>
<1,5,2>
<1,6,3>
...
<1,300,3>
<2,1,2>
…
<5000,299,1>
<5000,300,1>
...

Parse & Count

“Local” Sort

Merge Sort

Term Lexicon:

the 1
campaign 2
news 3
a 4
…

DocID Lexicon:

doc1 1
doc2 2
doc3 3
…

Inverted Index Compression

- In general, leverage skewed distribution of values and use variable-length encoding
- TF compression
  - Small numbers tend to occur far more frequently than large numbers (why?)
  - Fewer bits for small (high frequency) integers at the cost of more bits for large integers
- Doc ID compression
  - “d-gap” (store difference): d1, d2-d1, d3-d2,...
  - Feasible due to sequential access

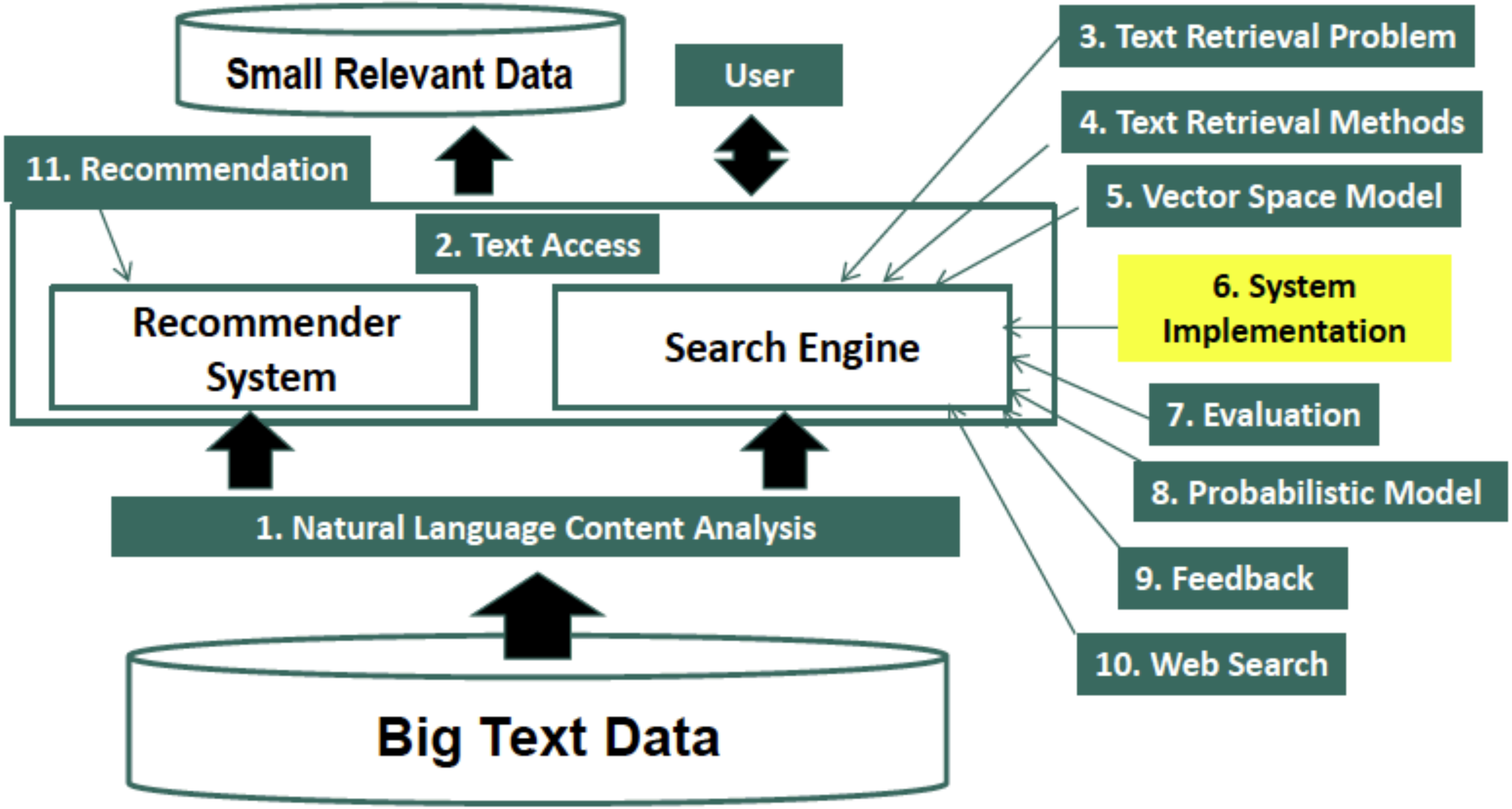
Information Retrieval & Text Mining

System Implementation

Fast Search

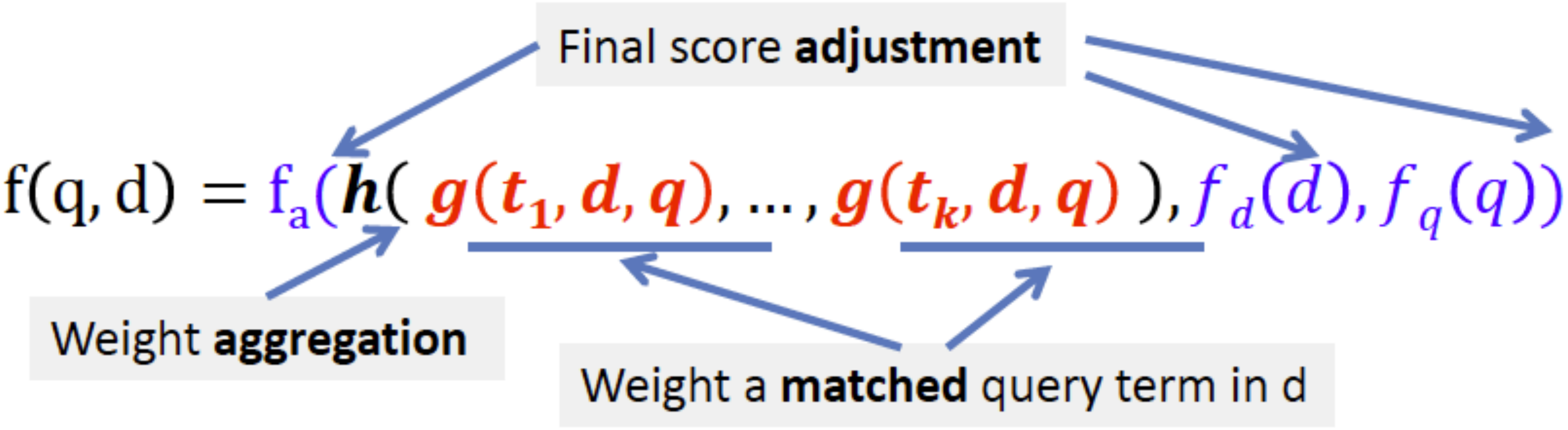
Dr. Saeed Ul Hassan
Information Technology University

Implementation of Text Retrieval Systems



How to Score Documents Quickly

General Form of Scoring Function



Final score **adjustment**

$$f(q, d) = f_a(\boldsymbol{h}(\underline{\boldsymbol{g}(\boldsymbol{t}_1, \boldsymbol{d}, \boldsymbol{q})}, \ldots, \underline{\boldsymbol{g}(\boldsymbol{t}_k, \boldsymbol{d}, \boldsymbol{q})}), f_d(d), f_q(q))$$

Weight **aggregation**

Weight a **matched** query term in d

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

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
(d1,3) =>	3	0	0	0	0

An Example: Ranking Based on TF Sum

$f(d,q)=g(t_1,d,q)+\ldots+g(t_k,d,q)$ 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
(d1,3) =>	**3**	0	0	0	0
(d2,4) =>	3	**4**	0	0	0

An Example: Ranking Based on TF Sum

$f(d,q)=g(t_1,d,q)+\ldots+g(t_k,d,q)$ 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
(d1,3) =>	**3**	0	0	0	0
(d2,4) =>	3	**4**	0	0	0
(d3,1) =>	3	4	**1**	0	0

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
(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

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
(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
(d2,3) =>	3	**7**	1	5	0

An Example: Ranking Based on TF Sum

$f(d,q)=g(t_1,d,q)+\ldots+g(t_k,d,q)$ 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
(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
(d2,3) =>	3	**7**	1	5	0
(d4,1) =>	3	7	1	**6**	0

An Example: Ranking Based on TF Sum

$f(d,q)=g(t_1,d,q)+\ldots+ g(t_k,d,q)$ 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
(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
(d2,3) =>	3	**7**	1	5	0
(d4,1) =>	3	7	1	**6**	0
(d5,3) =>	3	7	1	6	**3**

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.