Distributed data processing on the Cloud – Lecture 4

MapReduce in Information Retrieval

Satish Srirama

Some material adapted from slides by Jimmy Lin, 2008 (licensed under Creation Commons Attribution 3.0 License)
Outline

• Looked at several algorithms
• Let us try to apply them in a particular domain
• Introduction to IR
• Boolean retrieval
• Ranked retrieval
The Information Retrieval Cycle

Source Selection

Query Formulation

Search

Selection

Examination

Delivery

Resource

Query

Results

Documents

Information

System discovery
Vocabulary discovery
Concept discovery
Document discovery

source reselection
The Central Problem in Search

Searcher

Author

Concepts

Concepts

Query Terms

Document Terms

“tragic love story”

“fateful star-crossed romance”

Do these represent the same concepts?
Architecture of IR Systems

Query Representation

Comparison Function

Query Representation

Representation Function

Documents

Representation Function

Document Representation

Index

online offline

Hits
How do we represent text?

• Remember: computers don’t “understand” anything!
• “Bag of words”
  – Treat all the words in a document as index terms for that document
  – Assign a “weight” to each term based on “importance”
  – Disregard order, structure, meaning, etc. of the words
  – Simple, yet effective!
• Assumptions
  – Term occurrence is independent
  – Document relevance is independent
  – “Words” are well-defined
What’s a word?

天主教教宗若望保禄二世因感冒再度住進醫院。這是他今年第二度因同樣的病因住院。 وقال مارك ريجيف - الناطق باسم الخارجية الإسرائيلية - إن شارون قبل الدعوة وسيقوم للمرة الأولى بزيارة تونس، التي كانت لفترة طويلة المقر الرسمي لمنظمة التحرير الفلسطينية بعد خروجها من لبنان عام 1982.

Выступая в Мещанском суде Москвы экс-глава ЮКОСа заявил не совершал ничего противозаконного, в чем обвиняет его генпрокуратура России.

भारत सरकार ने आर्थिक सर्वेक्षण में वित्तीय वर्ष 2005-06 में सात फीसदी विकास दर हासिल करने का आकलन किया है और कर सुधार पर ज़ोर दिया है।

日米連合で台頭中国に対処…アーミテージ前副長官提言

28.09.2018 Satish Srirama 7/23
McDonald's slims down spuds

Fast-food chain to reduce certain types of fat in its french fries with new cooking oil.

NEW YORK (CNN/Money) - McDonald's Corp. is cutting the amount of "bad" fat in its french fries nearly in half, the fast-food chain said Tuesday as it moves to make all its fried menu items healthier.

But does that mean the popular shoestring fries won't taste the same? The company says no. "It's a win-win for our customers because they are getting the same great french-fry taste along with an even healthier nutrition profile," said Mike Roberts, president of McDonald's USA.

But others are not so sure. McDonald's will not specifically discuss the kind of oil it plans to use, but at least one nutrition expert says playing with the formula could mean a different taste.

Shares of Oak Brook, Ill.-based McDonald's (MCD: down $0.54 to $23.22, Research, Estimates) were lower Tuesday afternoon. It was unclear Tuesday whether competitors Burger King and Wendy's International (WEN: down $0.80 to $34.91, Research, Estimates) would follow suit. Neither company could immediately be reached for comment.

...
Boolean Retrieval

• Users express queries as a Boolean expression
  – AND, OR, NOT
  – Can be arbitrarily nested

• Retrieval is based on the notion of sets
  – Any given query divides the collection into two sets:
    retrieved, not-retrieved
  – Pure Boolean systems do not define an ordering of the results
Representing Documents

Document 1
The quick brown fox jumped over the lazy dog’s back.

Document 2
Now is the time for all good men to come to the aid of their party.

Term

<table>
<thead>
<tr>
<th>Term</th>
<th>Document 1</th>
<th>Document 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>aid</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>all</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>back</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>brown</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>come</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>fox</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>good</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>jump</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lazy</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>men</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>now</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>over</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>party</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>quick</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>their</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>time</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Stopword List
- for
- is
- of
- the
- to
## Inverted Index

<table>
<thead>
<tr>
<th>Term</th>
<th>Doc 1</th>
<th>Doc 2</th>
<th>Doc 3</th>
<th>Doc 4</th>
<th>Doc 5</th>
<th>Doc 6</th>
<th>Doc 7</th>
<th>Doc 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>aid</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>all</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>back</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>brown</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>come</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>dog</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fox</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>good</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>jump</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lazy</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>men</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>now</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>over</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>party</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>quick</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>their</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>time</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Term Postings

<table>
<thead>
<tr>
<th>Term</th>
<th>Postings</th>
</tr>
</thead>
<tbody>
<tr>
<td>aid</td>
<td>4 → 8</td>
</tr>
<tr>
<td>all</td>
<td>2 → 4 → 6</td>
</tr>
<tr>
<td>back</td>
<td>1 → 3 → 7</td>
</tr>
<tr>
<td>brown</td>
<td>1 → 3 → 5 → 7</td>
</tr>
<tr>
<td>come</td>
<td>2 → 4 → 6 → 8</td>
</tr>
<tr>
<td>dog</td>
<td>3 → 5</td>
</tr>
<tr>
<td>fox</td>
<td>3 → 5 → 7</td>
</tr>
<tr>
<td>good</td>
<td>2 → 4 → 6 → 8</td>
</tr>
<tr>
<td>jump</td>
<td>3</td>
</tr>
<tr>
<td>lazy</td>
<td>1 → 3 → 5 → 7</td>
</tr>
<tr>
<td>men</td>
<td>2 → 4 → 8</td>
</tr>
<tr>
<td>now</td>
<td>2 → 6 → 8</td>
</tr>
<tr>
<td>over</td>
<td>1 → 3 → 5 → 7 → 8</td>
</tr>
<tr>
<td>party</td>
<td>6 → 8</td>
</tr>
<tr>
<td>quick</td>
<td>1 → 3</td>
</tr>
<tr>
<td>their</td>
<td>1 → 5 → 7</td>
</tr>
<tr>
<td>time</td>
<td>2 → 4 → 6</td>
</tr>
</tbody>
</table>
Boolean Retrieval

• To execute a Boolean query:
  – Build query syntax tree
  (fox or dog) and quick
  – For each clause, look up postings
    
    --
    |  dog   |
    |  3 -> 5 |
    --
    |  fox   |
    |  3 -> 5 -> 7 |
  – Traverse postings and apply Boolean operator
    
    --
    |  dog   |
    |  3 -> 5 |
    --
    |  fox   |
    |  3 -> 5 -> 7 |
    OR = union

    --
    |  3 -> 5 -> 7 |

• Efficiency analysis
  – Postings traversal is linear (assuming sorted postings)
  – Start with shortest posting first
Extensions

• Implementing proximity operators
  – Store word offset in postings

• Handling term variations
  – Stem words: love, loving, loves … \(\rightarrow\) lov
Strengths and Weaknesses

- **Strengths**
  - Precise, if you know the right strategies
  - Precise, if you have an idea of what you’re looking for
  - Implementations are fast and efficient

- **Weaknesses**
  - Users must learn Boolean logic
  - Boolean logic insufficient to capture the richness of language
  - No control over size of result set: either too many hits or none
  - **When do you stop reading?** All documents in the result set are considered “equally good”
  - **What about partial matches?** Documents that “don’t quite match” the query may be useful also
Ranked Retrieval

• Order documents by how likely they are to be relevant to the information need
  – Estimate relevance \( q, d_i \)
  – Sort documents by relevance
  – Display sorted results

• User model
  – Present hits one screen at a time, best results first
  – At any point, users can decide to stop looking

• How do we estimate relevance?
  – Assume document is relevant if it has a lot of query terms
  – Replace relevance \( (q, d_i) \) with \( \text{sim}(q, d_i) \)
  – Compute similarity of vector representations
Assumption: Documents that are “close together” in vector space “talk about” the same things

Therefore, retrieve documents based on how close the document is to the query (i.e., similarity ~ “closeness”)
Similarity Metric

• How about $|d_1 - d_2|$?
• Instead of Euclidean distance, use “angle” between the vectors
  – It all boils down to the inner product (dot product) of vectors

$$\cos(\theta) = \frac{\vec{d}_j \cdot \vec{d}_k}{\|\vec{d}_j\| \|\vec{d}_k\|}$$

$$sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\|\vec{d}_j\| \|\vec{d}_k\|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$$
Term Weighting

• Term weights consist of two components
  – Local: how important is the term in this document?
  – Global: how important is the term in the collection?

• Here’s the intuition:
  – Terms that appear often in a document should get high weights
  – Terms that appear in many documents should get low weights

• How do we capture this mathematically?
  – Term frequency (local)
  – Inverse document frequency (global)
TF.IDF Term Weighting

\[ w_{i,j} = \text{tf}_{i,j} \cdot \log \frac{N}{n_i} \]

- \( w_{i,j} \) weight assigned to term \( i \) in document \( j \)
- \( \text{tf}_{i,j} \) number of occurrence of term \( i \) in document \( j \)
- \( N \) number of documents in entire collection
- \( n_i \) number of documents with term \( i \)

TF-IDF : Term frequency – Inverse Document Frequency
TF.IDF Example

\[
\begin{array}{c|cccc}
 & 1 & 2 & 3 & 4 \\
\hline
\text{complicated} &  &  & 5 & 2 \\
\text{contaminated} & 4 & 1 & 3 & \\
\text{fallout} & 5 & 4 & 3 & \\
\text{information} & 6 & 3 & 3 & 2 \\
\text{interesting} & 1 &  &  & \\
\text{nuclear} & 3 &  & 7 & \\
\text{retrieval} & 6 & 1 & 4 & \\
\text{siberia} & 2 &  &  & \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
 & \text{tf} & \text{idf} & \text{complicated} & \text{contaminated} \\
\hline
\text{complicated} & 0.301 & & 3,5 & 4,2 \\
\text{contaminated} & 0.125 & & 1,4 & 2,1 & 3,3 \\
\text{fallout} & 0.125 & & 1,5 & 3,4 & 4,3 \\
\text{information} & 0.000 & & 1,6 & 2,3 & 3,3 & 4,2 \\
\text{interesting} & 0.602 & & 2,1 & \\
\text{nuclear} & 0.301 & & 1,3 & 3,7 \\
\text{retrieval} & 0.125 & & 2,6 & 3,1 & 4,4 \\
\text{siberia} & 0.602 & & 1,2 & \\
\end{array}
\]
Sketch: Scoring Algorithm

• Initialize *accumulators* to hold document scores

• For each query term $t$ in the user’s query
  – Fetch $t$’s postings
  – For each document, $score_{doc} += w_{t,d} \times w_{t,q}$

• Apply length normalization to the scores at end

• Return top $N$ documents
Tasks to MapReduce

- TF-IDF calculation
- Index construction – Leaving it for those who are interested
  - Can be taken as thesis or research topic
TF-IDF : Information we need

• Number of times term X appears in a given document
• Number of terms in each document
• Number of documents X appears in
• Total number of documents
Job 1: Word Frequency in Doc

• Mapper
  – Input: (docname, contents)
  – Output: ((word, docname), 1)

• Reducer
  – Sums counts for word in document
  – Outputs ((word, docname), n)

• What is an ideal Combiner?
  – Combiner is same as Reducer
Job 2: Word Counts For Docs

- Mapper
  - Input: ((word, docname), n)
  - Output: (docname, (word, n))

- Reducer
  - Sums frequency of individual n’s in same doc
  - Feeds original data through
  - Outputs ((word, docname), (n, N))
Job 3: Word Frequency In Corpus

• Mapper
  – Input: ((word, docname), (n, N))
  – Output: (word, (docname, n, N, 1))

• Reducer
  – Sums counts for word in corpus
  – Outputs ((word, docname), (n, N, m))
Job 4: Calculate TF-IDF

• Mapper
  – Input: ((word, docname), (n, N, m))
  – Assume $D$ (Number of documents) is known (or, easy MR to find it)
  – Output ((word, docname), TF-IDF)
    • TF-IDF = $\frac{n}{N} \times \log\left(\frac{D}{m}\right)$

• Reducer
  – Just the identity function
Working At Scale

• Buffering \((doc, n, N)\) counts while summing 1’s into \(m\) may not fit in memory
  – How many documents does the word “the” occur in?

• Possible solutions
  – Ignore very-high-frequency words
  – Write out intermediate data to a file
  – Use another MR pass
Final Thoughts on TF-IDF

• Several small jobs add up to full algorithm
• Lots of code reuse possible
  – Stock classes exist for aggregation, identity
• Jobs 3 and 4 can really be done at once in same reducer, saving a write/read cycle
• Very easy to handle medium-large scale, but must take care to ensure flat memory usage for largest scale
How Index is Interesting?

• The indexing problem
  – Must be relatively fast, but need not be real time
  – For Web, incremental updates are important
• The retrieval problem
  – Must have sub-second response
  – For Web, only need relatively few results
• Fundamentally, a large sorting problem
  – Terms usually fit in memory
  – Postings usually don’t
• How is it done on a single machine?
• How large is the inverted index?
  – Size of vocabulary
  – Size of postings
Query Execution

• MapReduce is meant for large-data batch processing
  – Not suitable for lots of real time operations requiring low latency
MapReduce: Query Execution

• High-throughput batch query execution:
  – Instead of sequentially accumulating scores per query term:
  – Have mappers traverse postings in parallel, emitting partial score components
  – Reducers serve as the accumulators, summing contributions for each query term

• MapReduce does all the heavy lifting
  – Replace random access with sequential reads
  – Pays off over lots of queries
  – Examine multiple postings in parallel
This week in lab...

• You’ll try information retrieval with MapReduce
Next Lecture

• **Note**: Next week (5\textsuperscript{th} October 2018) no lecture
  – ICS Day
  – Visit Mobile & Cloud Lab (Ulikooli 17 - 321)
  – Relevant labs also will be pushed by a week!
    • No new lab exercises for 8\textsuperscript{th} & 9\textsuperscript{th} October
    • Consultation session for finishing all the pending tasks

• Graph Data Processing with MapReduce
  – 12\textsuperscript{th} October 2018
References

• Check “Vector space model” in Wikipedia
  http://en.wikipedia.org/wiki/Vector_space_model

• Data-Intensive Text Processing with MapReduce
  Authors: Jimmy Lin and Chris Dyer
  http://lintool.github.io/MapReduceAlgorithms/
  MapReduce-book-final.pdf