Information Retrieval and Web Search

Salvatore Orlando


Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008
Introduction

• Text mining refers to data mining using text documents as data.
• Most text mining tasks use Information Retrieval (IR) methods to pre-process text documents.
• These methods are quite different from traditional data pre-processing methods used for relational tables.

• Web search also has its root in IR.
• First, discuss the feature of the Web data
Web

• Web:
  – A huge, widely-distributed, highly heterogeneous, semistructured, interconnected, evolving, hypertext/hypermedia information repository

• Main issues
  – *Abundance of information*
    • The 99% of all the information are not interesting for the 99% of all users
  – *The static Web is a very small part of all the Web*
    • Dynamic Website
  – *To access the Web user need to exploit Search Engines (SE)*
    • SE must be improved
    • To help people to better formulate their information needs
    • More personalization is needed
Such numbers represent an estimation of minimum size of Internet.
The websites are many more, while the number of pages is almost endless.
Web: Trends and Features

- Google in July 2007 announced to have identified 1 trillion \(10^{12}\) of unique pages/URLs in the Web
  - After removing duplicates (about 30%-40%) !!!
  - Estimated growth: several billions of pages per day
  - Source: http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html

- Note that many pages are dynamically created .... and this introduces complexity for systems like Google
  - Think about a Web calendar on the Web .... and a link to next month ... we can follow this link a unbounded number of times, by creating always new pages
Web: Trends and Features

- **How many disks for all the Web pages?**
  - Consider only the text (HTML)
  - On the average, 10K Byte
  - Considering a trillion of pages
    - About $10^{16}$ Byte
  - Using Hard Disks of 1 Terabyte (about $10^{12}$ bytes)
    - About 10,000 disks

- **Things make worse if we consider multimedia data**
Web: Trends and Features

• Besides the grows of the page number, the pages are also continuously updated or removed
  – About the 23% of all the pages are modified daily
  – In the \texttt{.com} domain, this percentage rises to 40%
  – On the average, after 10 days, half of the new pages are removed
    • Their URL are no longer valid

A. Arasu et al., \textit{“Searching the Web”}, ACM Transaction on Internet Technology, 1(1), 2001.
Web: Trends and Features

The structure of the Web network/graph (Bow-tie)

- 28% of all the pages
  - Core of the network
  - Important pages ... highly interconnected with each other

- 22% of all the pages
  - reachable from the core, but not vice versa

- The rest of the pages are disconnected from the network core

• Power law
  – The degree of a node is the number of incoming/outgoing links
  – If we call $k$ the degree of a node, a scale-free network is defined by the power-law, which corresponds to this distribution:
  
  $p(k) \approx c \cdot k^{-\alpha}$

  – Most of the nodes are poorly interconnected

The *Power law* (Long Tail) is ubiquitous in the Web

- **Contents**
  - Words in the pages (frequency of words): the most common words are very popular, but there is a long tail of infrequent words!

- **Structure**
  - In-degrees / Out-degrees / Numbers of pages per website

- **Usage patterns**
  - Numbers of visitors
  - Query/Terms submitted by users of Search Engine
Long Tail in retail: product popularity (songs)

Sources: Erik Brynjolfsson and Jeffrey Hu, MIT, and Michael Smith, Carnegie Mellon; Barnes & Noble; Netflix; RealNetworks
Information Retrieval (IR)

- IR helps users find information that matches their information needs expressed as queries.
- Historically, IR is about document retrieval, emphasizing *document* as the *basic unit*.
  - Finding documents relevant to user queries.
- Technically, IR studies the acquisition, organization, storage, retrieval, and distribution of information.
IR architecture

The user

User query

Query operations

Executable query

Ranked documents

Retrieval system

Document collection

Indexer

Document index
IR queries

- Keyword queries
- Boolean queries (using AND, OR, NOT)
- Phrase queries
- Proximity queries
- Full document queries
- Natural language questions
Information retrieval models

• An IR model governs how a document and a query are represented and how the relevance of a document to a user query is defined

• Main models:
  – Boolean model
  – Vector space model
  – Statistical language model
  – etc
Boolean model

• Each document or query is treated as a “bag” of words or terms
  – Ordering of words is not considered

• Given a collection of documents $D$, let
  
  \[
  V = \{ t_1, t_2, \ldots, t_{|V|} \}
  \]

  be the set of distinctive words/terms in the collection. $V$ is called the vocabulary

• A weight $w_{ij} > 0$ is associated with each term $t_i$ of a document $d_j \in D$
  – $w_{ij} = 0/1$ (absence/presence)
  – $d_j = (w_{1j}, w_{2j}, \ldots, w_{|V|j})$
Boolean model (contd)

• Query terms are combined logically using the Boolean operators AND, OR, and NOT.
  – E.g., ((data AND mining) AND (NOT text))

• Retrieval
  – Given a Boolean query, the system retrieves every document that makes the query logically true
  – Exact match

• The retrieval results are usually quite poor because term frequency is not considered.
Boolean model: an Example

• Consider a document space defined by three terms, i.e., the vocabulary / lexicon:
  – hardware, software, users
• A set of documents is defined as:
  – A1=(1, 0, 0), A2=(0, 1, 0), A3=(0, 0, 1)
  – A4=(1, 1, 0), A5=(1, 0, 1), A6=(0, 1, 1)
  – A7=(1, 1, 1) A8=(1, 0, 1), A9=(0, 1, 1)
• If the query is: “hardware, software”
  i.e., (1, 1, 0)
what documents should be retrieved?
  – AND: documents A4, A7
  – OR: all documents, but A3
Similarity matching: an Example

• A set of documents is defined as:
  – A1=(1, 0, 0),  A2=(0, 1, 0),  A3=(0, 0, 1)
  – A4=(1, 1, 0),  A5=(1, 0, 1),  A6=(0, 1, 1)
  – A7=(1, 1, 1),  A8=(1, 0, 1),  A9=(0, 1, 1)

• In similarity matching (cosine in the Boolean vector space):
  – q=(1, 1, 0)
  – S(q, A1)=0.71,  S(q, A2)=0.71,  S(q, A3)=0
  – S(q, A4)=1,  S(q, A5)=0.5,  S(q, A6)=0.5
  – S(q, A7)=0.82,  S(q, A8)=0.5,  S(q, A9)=0.5
  – Document retrieved set (with ranking, where cosine>0):
    • {A4, A7, A1, A2, A5, A6, A8, A9}
Vector space model

- Documents are still treated as a “bag” of words or terms.
- Each document is still represented as a vector.
- However, the term weights are not forced to be 0 or 1, like in the Boolean model
  - Each term weight is computed on the basis of some variations of TF or TF-IDF scheme.

- **Term Frequency (TF) Scheme:** The weight of a term $t_i$ in document $d_j$ is the number of times that $t_i$ appears in $d_j$, denoted by $f_{ij}$. Normalization may also be applied.
The most well known weighting scheme

- TF: term frequency
- IDF: inverse document frequency.

\[ tf_{ij} = \frac{f_{ij}}{\max\{f_{1j}, f_{2j}, \ldots, f_{|V|j}\}} \]

\[ idf_i = \log \frac{N}{df_i} \]

The final TF-IDF term weight is:

\[ w_{ij} = tf_{ij} \times idf_i. \]
Retrieval in vector space model

- Query \( q \) is represented in the same way or slightly differently.

- **Relevance of \( d_i \) to \( q \):** Compare the similarity of query \( q \) and document \( d_i \), i.e. the similarity between the two associated vectors.

- **Cosine similarity** (the cosine of the angle between the two normalized vectors)

\[
\text{cosine}(d_j, q) = \frac{\langle d_j \cdot q \rangle}{\|d_j\| \times \|q\|} = \frac{\sum_{i=1}^{|V|} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{|V|} w_{ij}^2} \times \sqrt{\sum_{i=1}^{|V|} w_{iq}^2}}
\]

- Cosine is also commonly used in text clustering
Retrieval in the vector space model

- Not only documents $d_j \in D$, but also queries $q$ are represented as a vector of weights of $|V|$ elements:

$$d_j = (w_{1j}, w_{2j}, \ldots, w_{|V|j}) \quad q = (w_{1q}, w_{2q}, \ldots, w_{|V|q})$$
Retrieval in the vector space model

- **Rilevance of** $d_i$ **w.r.t.** $q$: Compare the similarity of query $q$ and document $d_i$, i.e., the similarity between the two associated vectors
- **Cosine similarity** (the cosine of the angle)

**Example with two 2-dimensional vectors**

$|V| = 2$
Relevance feedback

- Relevance feedback is one of the techniques for improving retrieval effectiveness. The steps:
  - the user first identifies some relevant \( (D_r) \) and irrelevant documents \( (D_{ir}) \) in the initial list of retrieved documents
  - goal: “expand” the query vector in order to maximize similarity with relevant documents, while minimizing similarity with irrelevant documents
    - query \( q \) expanded by extracting additional terms from the sample of relevant \( (D_r) \) and irrelevant \( (D_{ir}) \) documents to produce \( q_e \)
      \[
      q_e = \alpha q + \frac{\beta}{|D_r|} \sum_{d_r \in D_r} d_r - \frac{\gamma}{|D_{ir}|} \sum_{d_{ir} \in D_{ir}} d_{ir}
      \]
    - perform a second round of retrieval

- Rocchio method (\( \alpha, \beta \) and \( \gamma \) are parameters)
Rocchio text classifier

• Training set: relevant and irrelevant docs
  – you can train a classifier

• The Rocchio classification method, can be used to improve retrieval effectiveness too

• Rocchio classifier is constructed by producing a prototype vector $c_i$ for each class $i$ (relevant or irrelevant in this case) associated with document set $D_i$:

$$c_i = \alpha \frac{\sum_{d \in D_i} d}{|D_i|}\|d\| - \beta \frac{\sum_{d \in D-D_i} d}{|D-D_i|}\|d\|$$

• In classification, cosine is used
  – the class is determined by the closest class prototype (1NN)

Each vector is normalized (sz=1)
Text pre-processing

- Document parsing for word (term) extraction: easy
- Stopwords removal
- Stemming
- Frequency counts and computing TF-IDF term weights.
Stopwords removal

- Many of the most frequently used words in English are useless in IR and text mining – these words are called *stop words*
  - “the”, “of”, “and”, “to”, …
  - Typically about 400 to 500 such words
  - For an application, an additional domain specific stopwords list may be constructed

- Why do we need to remove stopwords?
  - Reduce indexing (or data) file size
    - stopwords accounts 20-30% of total word counts.
  - Improve efficiency and effectiveness
    - stopwords are not useful for searching or text mining
    - they may also confuse the retrieval system

- Current Web Search Engines generally do not use stopword lists for “phrase search queries”
Stemming

- Techniques used to find out the root/stem of a word. e.g.,
  - user
  - users
  - used
  - using
  - engineering
  - engineered
  - engineer

Usefulness:
- improving effectiveness of IR and text mining
  - Matching similar words
  - Mainly improve recall
- reducing indexing size
  - combing words with the same roots may reduce indexing size as much as 40-50%
  - Web Search Engine may need to index un-stemmed words too for “phrase search”
Basic stemming methods

Using a set of rules. e.g., English rules

• remove ending
  – if a word ends with a consonant other than s, followed by an s, then delete s.
  – if a word ends in es, drop the s.
  – if a word ends in ing, delete the ing unless the remaining word consists only of one letter or of th.
  – If a word ends with ed, preceded by a consonant, delete the ed unless this leaves only a single letter.
  – ……

• transform words
  – if a word ends with “ies”, but not “eies” or “aies”, then “ies → y”
Evaluation: Precision and Recall

- **Given a query:**
  - Are all retrieved documents relevant?
  - Have all the relevant documents been retrieved?

- **Measures for system performance:**
  - The first question is about the **precision** of the search
    \[
    \text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = P(\text{relevant|retrieved})
    \]
  - The second is about the completeness (**recall**) of the search
    \[
    \text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = P(\text{retrieved|relevant})
    \]

- By increasing the number of retrieved items, we usually increase the **recall**, but also reduce **precision**
  - see next slide, where we plot recall vs. precision, obtained by increasing the size of the result set of a given query
Example 2: Following Example 1, we obtain the interpolated precisions at all 11 recall levels in the table of Fig. 6.4. The precision-recall curve is shown on the right.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$p(r_i)$</th>
<th>$r_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100%</td>
<td>0%</td>
</tr>
<tr>
<td>1</td>
<td>100%</td>
<td>10%</td>
</tr>
<tr>
<td>2</td>
<td>100%</td>
<td>20%</td>
</tr>
<tr>
<td>3</td>
<td>100%</td>
<td>30%</td>
</tr>
<tr>
<td>4</td>
<td>80%</td>
<td>40%</td>
</tr>
<tr>
<td>5</td>
<td>80%</td>
<td>50%</td>
</tr>
<tr>
<td>6</td>
<td>71%</td>
<td>60%</td>
</tr>
<tr>
<td>7</td>
<td>70%</td>
<td>70%</td>
</tr>
<tr>
<td>8</td>
<td>70%</td>
<td>80%</td>
</tr>
<tr>
<td>9</td>
<td>62%</td>
<td>90%</td>
</tr>
<tr>
<td>10</td>
<td>62%</td>
<td>100%</td>
</tr>
</tbody>
</table>

**Fig. 6.4.** The precision-recall curve
Compare different retrieval algorithms

Fig. 6.5. Comparison of two retrieval algorithms based on their precision-recall curves
Compare with multiple queries

- Compute the average precision at each recall level

\[
\bar{p}(r_i) = \frac{1}{|Q|} \sum_{j=1}^{Q} p_j(r_i),
\]  

(22)

where \( Q \) is the set of all queries and \( p_j(r_i) \) is the precision of query \( j \) at the recall level \( r_i \). Using the average precision at each recall level, we can also draw a precision-recall curve.

- Draw precision recall curves
- Do not forget the F-score/F-measure evaluation
Rank precision

• Compute the precision values at some selected rank positions.
  – Mainly used in Web search evaluation

• For a Web search engine, we can compute precisions for the top 5, 10, 15, 20, 25 and 30 returned pages
  – as the user seldom looks at more than 30 pages
  – P@5, P@10, P@15, P@20, P@25, P@30

• Recall is not very meaningful in Web search.
  – Why?
Inverted index

• The inverted index of a document collection is basically a data structure that
  – attaches each distinctive term with a list of all documents that contain the term.
• Thus, in retrieval, it takes constant time to
  – find the documents that contains a query term.
• Multiple query terms are also easy handled as we will see soon.
An example

Example 3: We have three documents of \( id_1 \), \( id_2 \), and \( id_3 \):

- **\( id_1 \):** Web mining is useful.
  
  \[
  \begin{array}{cccc}
  1 & 2 & 3 & 4 \\
  \end{array}
  \]

- **\( id_2 \):** Usage mining applications.
  
  \[
  \begin{array}{ccc}
  1 & 2 & 3 \\
  \end{array}
  \]

- **\( id_3 \):** Web structure mining studies the Web hyperlink structure.
  
  \[
  \begin{array}{cccccc}
  1 & 2 & 3 & 4 & 5 & 6 \\
  \end{array}
  \]

### Lexicon and Postings List

**Lexicon (A):**
- Applications: \( id_2 \), \( id_3 \)
- Hyperlink: \( id_3 \), \( id_1, id_2, id_3 \)
- Mining: \( id_3 \)
- Structure: \( id_3 \)
- Studies: \( id_3 \)
- Usage: \( id_2 \)
- Useful: \( id_1, id_3 \)
- Web: \( id_1, id_3 \)

**Postings List (B):**
- \( \langle id_2, 1, [3]\rangle \)
- \( \langle id_3, 1, [7]\rangle \)
- \( \langle id_1, 1, [2]\rangle, \langle id_2, 1, [2]\rangle, \langle id_3, 1, [3]\rangle \)
- \( \langle id_3, 2, [2, 8]\rangle \)
- \( \langle id_3, 1, [4]\rangle \)
- \( \langle id_2, 1, [1]\rangle \)
- \( \langle id_1, 1, [4]\rangle \)
- \( \langle id_1, 1, [1]\rangle, \langle id_3, 2, [1, 6]\rangle \)

**Fig. 6.7.** Two inverted indices: a simple version and a more complex version.
Index construction

• Easy! See the example,

Fig. 6.8. The vocabulary trie and the inverted lists
Index compression

- Postings lists are ordered by doclIDs
  - Compression – instead of doclIDs we can compress smaller gaps between doclIDs, thus reducing space requirements for the index
- Use a variable number of bit/byte for gap representation
  - the gaps have a smaller magnitude than doclIDs

apple $\rightarrow$ 1,2,3,5
pear $\rightarrow$ 2,4,5
tomato $\rightarrow$ 3,5

\[
dGap_0 = \text{doclID}_0 \\
dGap_{i>0} = \text{doclID}_i - \text{doclID}_{(i-1)}
\]
Index compression

- Example of compression using Variable Byte encoding

- \( 824_{10} = 110 0111000_2 \)
- \( 5_{10} = 101_2 \)
- \( 214577_{10} = 1101 0001100 0110001_2 \)

<table>
<thead>
<tr>
<th>docIDs</th>
<th>824</th>
<th>829</th>
<th>215406</th>
</tr>
</thead>
<tbody>
<tr>
<td>gaps</td>
<td>5</td>
<td>214577</td>
<td></td>
</tr>
<tr>
<td>VB code</td>
<td>00000110 0111000 10000101 0001101 0001100 0110001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.4  Variable byte (VB) encoding. Gaps are encoded using an integral number of bytes. The first bit, the continuation bit, of each byte indicates whether the code ends with this byte (1) or not (0).
Search using inverted index

Given a query $q$, search has the following steps:

- **Step 1 (Vocabulary search):** find each term/word in $q$ in the inverted index.
- **Step 2 (Results merging):** Merge results to find documents that contain all or some of the words/terms in $q$
  - **AND/OR of postings lists**
- **Step 3 (Rank score computation):** To rank the resulting documents/pages, by using
  - content-based ranking (e.g. TF-IDF)
  - link-based ranking $\leftarrow$ Web Search Engine
  - etc. etc.
Mission impossible?

- **WSE**
  - Crawl and index **billions** of pages
  - Answer **hundreds of millions** of queries per day
  - In less than **1 sec.** per query

- **Users**
  - Want to submit **short queries** (on avg. **2.5 terms**), often with orthographic errors
  - Expect to receive the **most relevant results** of the Web
  - In a **blink of eye**

- In terms of 1990 IR, almost unimaginable
Web Search as a huge IR system

User

Web spider

Indexer

The Web

Indexes

Ad indexes
Different search engines

• The real differences between different search engines are
  – their index weighting schemes
    • Including context where terms appear, e.g., title, body, emphasized words, etc.
  – their query processing methods (e.g., query classification, expansion, etc)
  – their ranking algorithms
  – few of these are published by any of the search engine companies. They are tightly guarded secrets.
Web Search Engines
Web Search Engines: what do the users search?

- The 250 most frequent terms in the famous AOL query log!
Query analysis to evaluate user needs

- **Informational** – want to learn about something (~40% / 65%)
  - Low hemoglobin

- **Navigational** – want to go to that page (~25% / 15%)
  - United Airlines

- **Transactional** – want to do something (web-mediated) (~35% / 20%)
  - Access a service
    - Seattle weather
  - Downloads
    - Mars surface images
  - Shop
    - Canon S410

- **Gray areas**
  - Find a good hub
    - Car rental Brasil
  - Exploratory search “see what’s there”

Anatomy of a modern Web Search Engine

Crawler

- Crawler(s)
- Page Repository
- Indexer Module
- Collection Analysis Module
- Query Engine
- Ranking
- Indexes: Text, Structure, Utility
- Usage feedback
- Client
- Queries → Results

Data and Web Mining - S. Orlando 49
Crawler

- It is a program that navigates the Web following the hyperlinks and stores them in a page repository.

- Design Issues of the Crawl module:
  - What pages to download
  - When to refresh
  - Minimize load on web sites
  - How to parallelize the process

- Page selection during crawling: Importance metric
  - Given a page P, define how “good” that page is, on the basis of several metrics (combination of them):
    - Popularity driven: Incoming-link counts (or PageRank)
    - Location driven: Deepness of the page in a site
    - Usage driven: Click counts of the pages (feedback)
    - Interest driven: driven from a query, based on the similarity with page contents (focused crawling)
Storage: Page repository

- The Page Repository is a scalable storage system for web pages
  - Allows the Crawler to store pages
  - Allows the Indexer and Collection Analysis to retrieve them
  - Similar to other data storage systems – DB or file systems
  - Does not have to provide some of the other systems’ features: transactions, logging, directory.
Designing a Distributed Page Repository

- Repository designed to work over a cluster of interconnected nodes
- Page distribution across nodes
  - Uniform distribution – any page can be sent to any node
  - Hash distribution policy – hash page ID space into node ID space
- Physical organization within a node
- Update strategy
  - batch (Periodically executed)
  - steady (Run all the time)
Indexer and collection analysis modules

• The **Indexer module** creates two indexes:
  – Text (content) index: Uses “Traditional” indexing methods like **Inverted Indexing**.
  – Structure (links) index: Uses a directed **graph** of pages and links.
    Sometimes also creates an inverted graph, in order to answer queries that ask for all the pages that have hyperlinks pointing to a given page.

• The **collection analysis module** uses the 2 basic indexes created by the indexer module in order to assemble “Utility Indexes”
  – e.g.: a site index.
Indexer: Design Issues and Challenges

- Index build must be:
  - Fast
  - Economic
  (unlike traditional index builds)

- Incremental indexing must be supported
- Personalization
- Storage: compression vs. speed
Index partitioning

• Partitioning Inverted Files
  – **Local** inverted file
    • each node contains indexes of a disjoint partition of the document collection
    • query is broadcasted and answers are obtained by merging local results
  – **Global** inverted file
    • each node is only responsible for a subset of terms in the collection
    • query is selectively sent to the appropriate nodes only
Query engine

Crawler(s)

Page Repository

Indexer Module

Collection Analysis Module

Query Engine

Ranking

Queries

Results

Indexes:

Text

Structure

Utility

Crawl Control

Usage feedback
Query Engine

Snippet

Decreasing order of page importance (ranking)
Query engine

- The query engine module accepts queries from multitudes of users and returns the results
  - Exploits the partitioned index to quickly find the relevant pages
  - Use Page Repository to prepare the page of the (10) results
    - snippet construction is query-based
  - Since the possible results are a huge number, the ranking module has to order the results according to their relevance

- Ranking
  - not only based on traditional IR content-based approaches
  - terms may be of poor quality or not relevant
  - insufficient self-description of user intent
  - combat spam
    - Link analysis, e.g. PageRank that exploits incoming links from “important” pages to raise the rank of pages
    - Exploit proximity of query terms in the pages
    - Learning to rank
Summary

• We only gave a VERY brief introduction to IR. There are a large number of other topics, e.g.,
  – Statistical language model
  – Latent semantic indexing (LSI and SVD).
• Many other interesting topics are not covered, e.g.,
  – Web search
    • Index compression
    • Ranking: combining contents and hyperlinks (see the next block of slides)
  – Web page pre-processing
  – Combining multiple rankings and meta search
  – Web spamming
• Read the textbooks