Chapter 10:
Information Retrieval
Information retrieval - Wikipedia, the free encyclopedia

Information retrieval (IR) is the science of searching for information in documents, searching for documents themselves, searching for metadata which ...

Information Retrieval
An online book by CJ van Rijsbergen, University of Glasgow.

Information Retrieval
Online text of a book by Dr. CJ van Rijsbergen of the University of Glasgow covering advanced topics in information retrieval.

information retrieval
www.springerlink.com/link.asp?id=103814 - Similar pages

Modern Information Retrieval
A recent IR book, covering algorithms, implementation, query languages, user interfaces, and multimedia and web retrieval.

Information Retrieval Research - SearchTools Topics
An up-to-date overview of research in the field of information retrieval.

www.searchtools.com/info/info-retrieval.html - 22k - Cached - Similar pages
Evaluation of IR Systems

All documents

Retrieved

relevant

Retrieved and relevant

Recall = #(retrieved and relevant) / #(relevant)

Precision = #(retrieved and relevant) / #(retrieved)
Precision vs. Recall
Average precision

Goal:
- don’t focus on a specific recall level
- still get on number

\[
\text{AvgP} = \frac{\sum_{r=1}^{N} P(r \in \text{rel}(r))}{\sum_{r=1}^{N} \text{rel}(r)}
\]

\(P(r)\) : precision at rank \(r\)
\(\text{rel}(r)\) : indicator function;
1 if document at rank \(r\) is relevant
Mean average precision

- Problem: average precision still specific to query

\[ MAP = \frac{1}{Q} \sum_{q=1}^{Q} \text{AvgP}(q) \]

Q: number of queries
Interpolated precision

- Recall levels for each query distinct from 11 standard recall levels
- Interpolation procedure is necessary
- Let \( r_j \) be the \( j \)-th standard recall level with \( j=1,2,\ldots,10 \). Then,

\[
P(\hat{j}) = \max_{r_j \leq r \leq r_{j+1}} P(\hat{r})
\]
Interpolated precision
Preprocessing

- Stemming
- Stop words
- Longer units ("New York")
Vector-space-model
Vector-space-model

- Considering every document as vector
  - The vector contains the weights of the index terms as components
  - In case of t index terms the dimension of the vector-space is also t
  - Similarity of query to a document is the correlation between the their vectors
  - Correlation quantified by cosine of the angle between the vectors
Vector-space-model

- **Index term weights**

\[ tf_{i,j} = \frac{freq_{i,j}}{\max_l freq_{l,j}} \]

with \( freq_{i,j} \) the frequency that term \( i \) occurs in document \( j \)

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

with \( n_i \) number of documents that contain term \( i \) and \( N \) total number of documents
Vector-space-model

- **Index term weights**
  - The weight of a term in a document is then calculated as product of the tf factor and the idf factor:

    \[ w_{i,j} = tf_{i,j} \times idf_i \]

  - Or for the query:

    \[ w_{i,q} = \left( 0.5 + \frac{0.5 freq_{i,q}}{\max_l freq_{l,q}} \right) \times idf_i \]
Distance Metrics

• Pick an L-norm
• Angel/cosine between vectors

\[
\cos(\vec{q}, \vec{d}) = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q_i^2} \sqrt{\sum_{i=1}^{n} d_i^2}}
\]
Vector-space-model

• **Advantages**
  • Improves retrieval performance as compared to Boolean retrieval
  • Partial matching allowed
  • Sort according to similarity

• **Disadvantages**
  • Assumes that index terms are independent
Models for Term Distribution

• Goal:
  • Understand the statistical properties of key words in a documents collection

• Assumptions
  • Probability for a term is proportional to the length of the document
  • Short text: each word occurs only once
  • Two neighboring occurrences of the same term are statistically independent
Poission Distribution

Probability that the i-th term occurs k times in the document

\[ P_{\lambda_i}(k) = e^{-\lambda_i} \frac{\lambda_i^k}{k!} \]

\( \lambda_i \) parameter of the distribution
Processes described by Poisson Distribution (Wikipedia)

- The number of cars that pass through a certain point on a road (sufficiently distant from traffic lights) during a given period of time.
- The number of spelling mistakes one makes while typing a single page.
- The number of phone calls at a call center per minute.
- The number of times a web server is accessed per minute.
- The number of roadkill (animals killed) found per unit length of road.
- The number of mutations in a given stretch of DNA after a certain amount of radiation.
- The number of unstable nuclei that decayed within a given period of time in a piece of radioactive substance. The radioactivity of the substance will weaken with time, so the total time interval used in the model should be significantly less than the mean lifetime of the substance.
- The number of pine trees per unit area of mixed forest.
- The number of stars in a given volume of space.
- The number of V2 rocket attacks per area in England, according to the fictionalized account in Thomas Pynchon's Gravity's Rainbow.
- The number of light bulbs that burn out in a certain amount of time.
- The number of viruses that can infect a cell in cell culture.
- The number of hematopoietic stem cells in a sample of unfractionated bone marrow cells.
- The inventivity of an inventor over their career.
- The number of particles that "scatter" off of a target in a nuclear or high energy physics experiment.
Check normalization and expectation value of k -> white board
Interpretation

• Let $N$ be the number of documents in the corpus

\[ N \ E_i(k) = N \ \lambda_i = : \text{cf}_i \] (collection frequency)

\[ N \ (1 - P_{\lambda_i}(0)) =: df_i \] (document frequency)
Experimental Test of Poisson Model

<table>
<thead>
<tr>
<th>Word</th>
<th>cf&lt;sub&gt;i&lt;/sub&gt;</th>
<th>λ&lt;sub&gt;i&lt;/sub&gt;</th>
<th>N(1-P(0))</th>
<th>df&lt;sub&gt;i&lt;/sub&gt;</th>
<th>Overestimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>follows</td>
<td>23533</td>
<td>0.2968</td>
<td>20363</td>
<td>21744</td>
<td>0.94</td>
</tr>
<tr>
<td>transformed</td>
<td>840</td>
<td>0.0106</td>
<td>845</td>
<td>807</td>
<td>1.03</td>
</tr>
<tr>
<td>soviet</td>
<td>35337</td>
<td>0.4457</td>
<td>28515</td>
<td>8204</td>
<td>3.48</td>
</tr>
<tr>
<td>students</td>
<td>15925</td>
<td>0.2008</td>
<td>14425</td>
<td>4953</td>
<td>2.91</td>
</tr>
<tr>
<td>james</td>
<td>11175</td>
<td>0.1409</td>
<td>10421</td>
<td>9191</td>
<td>1.13</td>
</tr>
<tr>
<td>freshly</td>
<td>611</td>
<td>0.0077</td>
<td>609</td>
<td>395</td>
<td>1.54</td>
</tr>
</tbody>
</table>

- Model often works
- Some terms like “soviet” are bursty
- Independence assumption is not valid
- Probabilistic Retrieval
- \(\rightarrow\) white board
Latent Semantic Analysis

- Word usage defined by term and document co-occurrence – matrix structure
- Latent structure / semantics in word usage
- Clustering documents or words Singular Value Decomposition
- Cubic Computational Scaling
Term Document Matrix Structure

• Create artificially heterogeneous collection
• 100 documents from 3 distinct newsgroups
• Indexed using standard stop word list
• 12418 distinct terms
• Term × Document Matrix \((12418 \times 300)\)
• 8% fill of sparse matrix
• Matrix of cosine similarity between documents
• Clear structure apparent
Theory of LSA

- Whiteboard
LSA Performance

- LSA consistently improves recall on standard test collections (precision/recall generally improved)
- Variable performance on larger TREC collections
- Dimensionality of Latent Space – a magic number – 300 – 1000 seems to work fine – no satisfactory way of assessing value.
- Computational cost high
Language Modeling

• The probability that a query Q was generated by a probabilistic model based on a document.

\[ p(q \mid d) \]

\[ p(d \mid q) \approx p(q \mid d) \ast p(d) \]

• Uni-gram model:

\[ P(q \mid d) = \prod_{i=1}^{n} P(q_i \mid d) \]
Discussion: performance (original Ponte&Croft-paper)

- Table 1 – detailed results of different models on TREC data

<table>
<thead>
<tr>
<th>Table 1</th>
<th>tf·idf</th>
<th>LM</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prec.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.7439</td>
<td>0.7590</td>
<td>+2.0</td>
</tr>
<tr>
<td>0.10</td>
<td>0.4521</td>
<td>0.4910</td>
<td>+8.6</td>
</tr>
<tr>
<td>0.20</td>
<td>0.3514</td>
<td>0.2061</td>
<td>+15.1</td>
</tr>
<tr>
<td>0.50</td>
<td>0.1558</td>
<td>0.2061</td>
<td>+32.3</td>
</tr>
<tr>
<td>0.70</td>
<td>0.0451</td>
<td>0.0760</td>
<td>+68.7</td>
</tr>
<tr>
<td>1.00</td>
<td>0.0028</td>
<td>0.0050</td>
<td>+76.9</td>
</tr>
</tbody>
</table>

- Table 2 – average improvement for LM

<table>
<thead>
<tr>
<th>Table 2</th>
<th>% average improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision/Recall</td>
<td>+ 19.5</td>
</tr>
<tr>
<td>Precision for top N docs</td>
<td>+16.32</td>
</tr>
<tr>
<td>Prec.</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.4449</td>
</tr>
<tr>
<td>100</td>
<td>0.2157</td>
</tr>
<tr>
<td>1000</td>
<td>0.0653</td>
</tr>
</tbody>
</table>
Smoothing Methods

- **Jelinek-mercer method**: involves a linear interpolation of the ML model with the collection model.

\[
P_\lambda(w \mid d) = (1 - \lambda) P_{ml}(w \mid d) + \lambda P(w \mid C)
\]
Smoothing Methods

- **Absolute discounting:** decrease the probability of seen words by subtracting a constant from their counts.

\[
P_s(w | d) = \frac{\max(c(w;d) - \delta, 0)}{\sum_{w^* \in V} c(w^*;d)} + \sigma P(w | C)
\]
Smoothing Methods

- **Bayesian smoothing using Dirichlet priors:**
  A multinomial distribution, for which the conjugate prior for bayesian analysis is the dirichlet distribution:

\[
P_{\mu}(w \mid d) = \frac{c(w; d) + \mu P(w \mid C)}{\sum_{w^* \in V} c(w^*; d) + \mu}
\]
Comparing different smoothing methods
Improved Language Models

- Bigrams
- Class LMs
- Grammar
- Prior knowledge (document length)
- Other resources (e.g. WordNet)
Apache Lucene - Overview

Apache Lucene

Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.

Apache Lucene is an open source project available for free download. Please use the links on the left to access Lucene.

Lucene News

19 June 2007 - Release 2.2 available

This release has many improvements since release 2.1. New major features:

- "Point-in-time" searching over NFS
The Lemur Toolkit for Language Modeling and Information Retrieval

The Lemur Toolkit is an open-source toolkit designed to facilitate research in language modeling and information retrieval. Lemur supports a wide range of industrial and research language applications such as ad-hoc retrieval, site-search, and text mining.

The toolkit supports indexing of large-scale text databases, the construction of simple language models for documents, queries, or subcollections, and the implementation of retrieval systems based on language models as well as a variety of other retrieval models. The system is written in the C and C++ languages, and is designed as a research system to run under Unix operating systems, although it can also run under Windows.

The toolkit is in constant development as part of the Lemur Project, a collaboration between the Computer Science Department at the University of Massachusetts and the School of Computer Science at Carnegie Mellon University.

News

News and announcements about the Lemur Toolkit, such as the latest release notes, upcoming releases and known problems with current versions.

Features

An "at-a-glance" listing of features within the Lemur Toolkit.

The Lemur Toolkit

How to install and use the Lemur Toolkit, together with code-level documentation, applications guides, working with offset annotations and beginners guides.

Indri Search Engine

More about Indri, Lemur’s latest search engine that is also available on its own when all you need is a search engine. Indri has an index capable of indexing very large collections and a structured query...
Summary

- Evaluation measures
- Vector space model
- Models of term distribution
- Probabilistic retrieval
- Latent semantic analysis
- Language models for IR