Information Retrieval

- Basics
- Data Structures and Access
- Indexing and Preprocessing
- Retrieval Models

Overview

- Basic concepts
  - Full-text indexing
  - Query language
- Access structures
  - Sequential search
  - Signatures
  - Inverted index
- Inverted index
  - Construction
  - Boolean queries
  - Ranking

What to Index on?

- Database systems index primary and secondary keys
  - This is the hybrid approach
  - Index provides fast access to a subset of database records
  - Scan subset to find solution set
- Title, author, id, creation date, …
  - Good idea, but none of these support content-based retrieval
- IR Problem: Can’t predict the keys that people will use in queries
  - Every word in a document is a potential search term
- IR Solution: Index by all keys (terms)
  - full-text indexing

Example Document

How aspartame prevents the toxicity of ochratoxin A.

Corry IE, Enfaltion L, Anne-Maoto

The toxicosis caused by ochratoxin A (OTA) is a common problem in foods and feeds and is a significant public health concern. The mechanism of OTA toxicity is not fully understood, but it is known that OTA can bind to DNA and cause genetic damage. Aspartame, a widely used artificial sweetener, has been shown in laboratory studies to have a protective effect against OTA toxicity. The mechanism of this protective effect is not clear, but it is believed to involve the metabolism of aspartame by intestinal bacteria, which leads to the production of antioxidants that can protect against OTA toxicity. Further research is needed to understand the mechanisms involved and to determine the potential of aspartame as a protective agent against OTA toxicity.
Full-Text Indexing

<table>
<thead>
<tr>
<th>Term</th>
<th>TF</th>
<th>Term</th>
<th>TF</th>
<th>Term</th>
<th>TF</th>
<th>Term</th>
<th>TF</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>31</td>
<td>by</td>
<td>6</td>
<td>peptide</td>
<td>4</td>
<td>such</td>
<td>3</td>
</tr>
<tr>
<td>of</td>
<td>26</td>
<td>effect</td>
<td>6</td>
<td>several</td>
<td>4</td>
<td>toxic</td>
<td>3</td>
</tr>
<tr>
<td>and</td>
<td>22</td>
<td>are</td>
<td>5</td>
<td>toxin</td>
<td>4</td>
<td>vitro</td>
<td>3</td>
</tr>
<tr>
<td>in</td>
<td>21</td>
<td>aspartate</td>
<td>5</td>
<td>also</td>
<td>3</td>
<td>when</td>
<td>3</td>
</tr>
<tr>
<td>a</td>
<td>15</td>
<td>exposure</td>
<td>5</td>
<td>countries</td>
<td>3</td>
<td>added</td>
<td>2</td>
</tr>
<tr>
<td>in</td>
<td>11</td>
<td>human</td>
<td>5</td>
<td>given</td>
<td>3</td>
<td>africa</td>
<td>2</td>
</tr>
<tr>
<td>as</td>
<td>9</td>
<td>with</td>
<td>5</td>
<td>it</td>
<td>3</td>
<td>balkan</td>
<td>2</td>
</tr>
<tr>
<td>ota</td>
<td>9</td>
<td>animals</td>
<td>4</td>
<td>preventative</td>
<td>3</td>
<td>be</td>
<td>2</td>
</tr>
<tr>
<td>for</td>
<td>8</td>
<td>include</td>
<td>4</td>
<td>rate</td>
<td>3</td>
<td>been</td>
<td>2</td>
</tr>
<tr>
<td>is</td>
<td>8</td>
<td>ochratoxin</td>
<td>4</td>
<td>shown</td>
<td>3</td>
<td>compound</td>
<td>2</td>
</tr>
</tbody>
</table>

Index ↔ Query Language

- Index is accessed by the atoms of the query language
  - Words in text, punctuation
  - Manually assigned terms
  - Document structure and fields
  - Inter- or intradocument links
- **Index must support query operators**
  - Feature sequences
  - Feature sets
  - Pattern matches
  - Statistics

Basic IR Processes

- **Information Need**
- **Query Representation**
- **Query**
- **Comparison**
- **Indexed Objects**
- **Document Representation**
- **Document**
- **Retrieved Objects**
- **Evaluation/Feedback**

Why Create Index Datastructures?

- **Sequential scan of the entire collection**
  - Very flexible (e.g., search for complex patterns)
  - Available in hardware form (e.g., Fast data finder)
  - Computational and I/O costs are $O(\text{characters in collection})$
  - Practical for only “small” collections
- **Use index for direct access**
  - An index associates a document with one or more keys
  - Present a key, get back the document
  - Evaluation time $O(\text{query term occurrences in collection})$
  - Practical for “large” collections
  - Many opportunities for optimization

An Illustrative Query Language and Retrieval Model

- **Indexing**
  - Full-text indexing
  - Each atom corresponds to a word
- **Query Language**
  - Conjunctive queries: $w_1 \land w_2 \land \ldots \land w_n$
  - A query is a conjunction of words
- **Retrieval Model**
  - Return the set of documents that satisfy the query
  - A document satisfies a query, if all query words occur in the document

Inverted Index

- **Source file**: collection, organized by document
  - one record per document, listing the terms that occur in this document
- **Inverted file**: collection organized by term
  - one record per term, listing the documents the term occurs in
- **Inverted lists are today the most common indexing technique**
Inverted Index Example

<table>
<thead>
<tr>
<th>Source Term</th>
<th>Inverted Term DocIDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 machine learning</td>
<td>computer 6</td>
</tr>
<tr>
<td>2 human learning</td>
<td>database 4</td>
</tr>
<tr>
<td>3 learning systems</td>
<td>human 2</td>
</tr>
<tr>
<td>4 database theory</td>
<td>learning 1, 2, 3</td>
</tr>
<tr>
<td>5 operating systems</td>
<td>machine 1</td>
</tr>
<tr>
<td>6 computer systems</td>
<td>operating 5</td>
</tr>
<tr>
<td></td>
<td>systems 3, 5, 6</td>
</tr>
<tr>
<td></td>
<td>theory 4</td>
</tr>
</tbody>
</table>

Index Contents

- **Feature presence/absence**
  - Boolean
  - Statistical (tf, df, ctf, doclen, mustf)
  - Often about 10% the size of the raw data (compressed)
- **Positional**
  - Feature location in document
  - Granularities
  - Word-level granularity about 20-30% the size of the raw data (compressed)

Boolean Query Evaluation

- **Operators**
  - AND: intersection of inverted document list
  - OR: union of inverted document list
  - NOT: complement of inverted document list
- **Order**
  - Equivalent queries with different evaluation order
  - Difference in efficiency

Inverted Index with Positions

<table>
<thead>
<tr>
<th>Source DocID Terms</th>
<th>Inverted Term DocIDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 machine learning</td>
<td>computer 6</td>
</tr>
<tr>
<td>2 human learning</td>
<td>database 4</td>
</tr>
<tr>
<td>3 learning systems</td>
<td>human 2</td>
</tr>
<tr>
<td>4 database theory</td>
<td>learning 1, 2, 3</td>
</tr>
<tr>
<td>5 operating systems</td>
<td>machine 1</td>
</tr>
<tr>
<td>6 computer systems</td>
<td>operating 5</td>
</tr>
<tr>
<td></td>
<td>systems 3, 5, 6</td>
</tr>
<tr>
<td></td>
<td>theory 4</td>
</tr>
</tbody>
</table>

Sparse Vectors

- **Many vectors in IR are high dimensional, but sparse**
  - Store non-zero entries as sorted list
  - (0,0,0,0,0,0,0,0,0,0,0,0,0,0) \(\Rightarrow\) 5:4, 8:3, 11:1
- **More memory efficient**
  - \(O(\text{non-zero elements})\)
- **Efficient set operations**
  - Merge by going through both lists in parallel
    - Intersection: keep only those where both non-zero
    - Union: keep those where at least one non-zero
    - Dot-Product: multiply and sum for those where both non-zero
    - Etc.
  - \(O(\text{non-zero elements})\)
  - Result is again a sparse vector

Accessing the Inverted Lists

- **Trie**
  - Supports exact & range based lookup
  - "comput*" matches subtree
  - "computation" = "computer"
  - \(O(\log |w|)\) lookup for query term \(w\)
- **B-Tree**
  - Supports exact & range based lookup
  - \(O(\log n)\) lookup for each query term
- **Hash-Table**
  - Supports only exact match
  - \(O(1)\) lookup for each query term

Linear inverted file on disk
Building the Inverted Index

• Step 1: Build partial index in memory
  – Sequentially read words from sorted documents (by DocID)
  – Look up word in current index structure (O(length of word))
    • If already contained: add DocID to end of inverted list (O(1))
    • If not contained: add word to index with new inverted list
  – If memory exhausted, write partial index to disk sorted by term

• Step 2: Merge partial indexes
  – Union of sparse vectors
  – Append lists if in both partial indexes
  – Each merge requires O(size of indexes)
  – O(log n) mergers

Compressing the Inverted Index

• Inverted lists are usually compressed
  – Uncompressed, the inverted index with word locations is about the size of the raw data
  – Compressed without position: about 10% of original text
  – Compressed with position: about 20-30% of original text

• Distribution of numbers is skewed
  – Most numbers are small (e.g., word locations, term frequency)
  – Distribution easily can be made more skewed
    Delta encoding: 5, 8, 10, 17 --> 5, 3, 2, 7

• Simple compression techniques are often the best choice
  – Goal: Time saved by reduced I/O > Time required to uncompress