2.2 Classical Information Retrieval Models

- Boolean Model
- Vectorspace Model

2.2.1 The Boolean Model Retrieval Model

- Binary index: Terms are either present or absent. Thus, \( w_i \in \{0, 1\} \)

- Queries are specified as Boolean expressions in which terms are combined with the operators AND, OR, and NOT
  - \( q = ta \ AND \ (tb \ AND \ NOT \ tc) \)

- Simple model based on set theory with precise semantics
  - The model views each document as just a set of words

vehicle OR car AND accident Search
Boolean Retrieval Function

The retrieval function can be defined recursively:

\[ R(t_i, d_j) = \begin{cases} \text{TRUE}, & \text{if } w_{ij} = 1 \quad (i.e. \ t_i \text{ is in } d_j) \\ \text{FALSE}, & \text{if } w_{ij} = 0 \quad (i.e. \ t_i \text{ is not in } d_j) \end{cases} \]

\[ R(q_1 \text{ AND } q_2, d_j) = R(q_1, d_j) \text{ AND } R(q_2, d_j) \]

\[ R(q_1 \text{ OR } q_2, d_j) = R(q_1, d_j) \text{ OR } R(q_2, d_j) \]

\[ R(\text{NOT } q, d_j) = \text{NOT } R(q, d_j) \]

The Boolean functions computes only values 0 or 1, i.e. Boolean retrieval classifies documents into two categories:

- relevant \( (R = 1) \)
- irrelevant \( (R = 0) \)

Example für Boolesches Retrieval

<table>
<thead>
<tr>
<th></th>
<th>d1</th>
<th>d2</th>
<th>d3</th>
</tr>
</thead>
<tbody>
<tr>
<td>accident</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>car</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>cause</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>crowd</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>die</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>drive</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>four</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>heavy</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>injur</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>more</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>morning</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>people</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>quarter</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>register</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>truck</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>trucker</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>vehicle</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>vienna</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>yesterday</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Query:

\( (\text{vehicle OR car AND accident} \) \)

\[ R(\text{vehicle OR car AND accident}, d_1) = \text{TRUE} \]
\[ R(\text{vehicle OR car AND accident}, d_2) = \text{FALSE} \]
\[ R(\text{vehicle OR car AND accident}, d_3) = \text{FALSE} \]

Query:

\( (\text{vehicle AND car OR accident} \) \)

\[ R(\text{vehicle AND car OR accident}, d_1) = \text{TRUE} \]
\[ R(\text{vehicle AND car OR accident}, d_2) = \text{TRUE} \]
\[ R(\text{vehicle AND car OR accident}, d_3) = \text{TRUE} \]
Processing Boolean Queries

- Conjunctive queries are most widely used.
- Example: Processing simple conjunctive queries:

```
Example: car AND accident
```

- Query Optimization: For more than two terms in a conjunctive query, start with two shortest posting lists.

Drawbacks of the Boolean Model

- Retrieval based on binary decision criteria
  - No notion of partial matching
  - No ranking of the documents is provided (absence of a grading scale)
    - The query \( q = t_1 \text{ OR } t_2 \text{ OR } t_3 \) is satisfied by document containing one, two or three of the terms \( t_1, t_2, t_3 \)
- No weighting of terms, \( w_i \in \{0, 1\} \)
- Information need has to be translated into a Boolean expression which most users find awkward
- The Boolean queries formulated by the users are most often too simplistic
- As a consequence, the Boolean model frequently returns either too few or too many documents in response to a user query
2.2.2 Vector Space Model

Example:

<table>
<thead>
<tr>
<th>d1</th>
<th>d2</th>
</tr>
</thead>
<tbody>
<tr>
<td>accident</td>
<td>4</td>
</tr>
<tr>
<td>car</td>
<td>3</td>
</tr>
<tr>
<td>vehicle</td>
<td>1</td>
</tr>
</tbody>
</table>

Index can be regarded as an n-dimensional space
- $w_{ij} > 0$ whenever $t_i \in d_j$

Each term corresponds to a dimension
- To each term $t_i$ is associated a unitary vector $\text{vec}(i)$
- The unitary vectors $\text{vec}(i)$ and $\text{vec}(j)$ are assumed to be orthonormal (i.e., index terms are assumed to occur independently within the documents)

Document can be regarded as
- vector started from (0,0,0)
- point in space

2.2.2.1 Coordinate Matching

- Documents and query are represented as
  - document vectors $\text{vec}(d_j) = (w_{1j}, w_{2j}, ..., w_{kj})$
  - query vector $\text{vec}(q) = (w_{1q}, ..., w_{kq})$
- Vectors have binary values
  - $w_{ij} = 1$ if term $t_i$ occurs in Dokument $d_j$
  - $w_{ij} = 0$ else
- Ranking:
  - Return the documents containing at least one query term
  - rank by number of occurring query terms
- Ranking function: scalar product
  - $R(q,d) = q \cdot d = \sum_{i=1}^{n} q_i \cdot d_i$
Coordinate Matching: Example

<table>
<thead>
<tr>
<th>d1</th>
<th>d2</th>
<th>d3</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>accident</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>car</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>cause</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>crowd</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>die</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>drive</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>four</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>heavy</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>injur</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>more</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>morning</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>people</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>quarter</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>register</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>truck</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>trucker</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>vehicle</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>vienna</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>yesterday</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Query vector represents terms of the query

![Search](image)

Resultat:

\[ q \cdot d1 = \]
\[ q \cdot d2 = \]
\[ q \cdot d3 = \]

Assessment of Coordinate Matching

- Advantage compared to Boolean Model: Ranking
- Three main **drawbacks**
  - frequency of terms in documents in not considered
  - no weighting of terms
  - privilege for larger documents
2.2.2.2 Term Weighting

- Use of binary weights is too limiting
  - Non-binary weights provide consideration for partial matches
  - These term weights are used to compute a degree of similarity between a query and each document

- How to compute the weights \( w_{ij} \) and \( w_{iq} \)?

- A good weight must take into account two effects:
  - quantification of intra-document contents (similarity)
    - \( tf \) factor, the term frequency within a document
  - quantification of inter-documents separation (dissimilarity)
    - \( idf \) factor, the inverse document frequency

- \( w_{ij} = tf(i,j) \times idf(i) \)  
  (Baeza-Yates & Ribeiro-Neto 1999)

### TF - Term Frequency

<table>
<thead>
<tr>
<th></th>
<th>d1</th>
<th>d2</th>
<th>d3</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>accident</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>car</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cause</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>crowd</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>die</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>drive</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>four</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>heavy</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>injur</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>more</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>morning</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>people</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>quarter</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>register</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>truck</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>trucker</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>vehicle</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>vienna</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>yesterday</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Let \( freq(i,j) \) be the raw frequency of term \( t_i \) within document \( d_j \) (i.e. number of occurrences of term \( t_i \) in document \( d_j \))

- A simple tf factor can be computed as
  \[ f(i,j) = freq(i,j) \]

- A normalized \( tf \) factor is given by
  \[ f(i,j) = freq(i,j) / \max(freq(l,j)) \]
  where the maximum is computed over all terms which occur within the document \( d_j \)

For reasons of simplicity, in this example \( f(i,j) = freq(i,j) \)  
(Baeza-Yates & Ribeiro-Neto 1999)
**IDF – Inverse Document Frequency**

- IDF can also be interpreted as the amount of information associated with the term \( t_i \). A term occurring in few documents is more useful as an index term than a term occurring in nearly every document.

- Let \( n_i \) be the number of documents containing term \( t_i \) (document frequency).
  - \( N \) be the total number of documents.

- A simple idf factor can be computed as
  \[
  \text{idf}(i) = \frac{1}{n_i}
  \]

- A normalized idf factor is given by
  \[
  \text{idf}(i) = \log \left( \frac{N}{n_i} \right)
  \]
  the log is used to make the values of tf and idf comparable.

**Example with TF and IDF**

- In this example a simple tf factor
  \[
  f(i,j) = \text{freq}(i,j)
  \]
  and a simple idf factor
  \[
  \text{idf}(i) = \frac{1}{n_i}
  \]
  are used.
Indexing a new Document

- Changes of the indexes when adding a new document $d$
  - a new document vector with $tf$ factors for $d$ is created
  - $idf$ factors for terms occurring in $d$ are adapted
- All other document vectors remain unchanged

Ranking

- Scalar product computes co-occurrences of term in document and query
  - Drawback: Scalar product privileges large documents over small ones
- Euclidian distance between endpoint of vectors
  - Drawback: euclidian distance privileges small documents over large ones
- Angle between vectors
  - the smaller the angle between query and document vector the more similar they are
  - the angle is independent of the size of the document
  - the cosine is a good measure of the angle
**Cosine Ranking Formula**

- the more the directions of query $a$ and document $d_j$ coincide the more relevant is $d_j$
- the cosine formula takes into account the ratio of the terms not their concrete number
- Let $\theta$ be the angle between $q$ and $d_j$
- Because all values $w_{ij} \geq 0$ the angle $\theta$ is between $0^\circ$ und $90^\circ$
  - the larger $\theta$ the less is $\cos \theta$
  - the less $\theta$ the larger is $\cos \theta$
  - $\cos 0 = 1$
  - $\cos 90^\circ = 0$

\[
\cos(q, d_j) = \frac{q \cdot d_j}{|q| \cdot |d_j|} = \frac{\sum_{i=1}^{t} w_{ij} \cdot w_{ij}}{\sqrt{\sum_{i=1}^{t} w_{ij}^2} \cdot \sqrt{\sum_{j=1}^{t} w_{ij}^2}}
\]

**The Vector Model**

- The best term-weighting schemes use weights which are given by
  - $w_{ij} = f(i,j) \cdot \log(N/n_i)$
  - the strategy is called a *tf-idf* weighting scheme
- For the query term weights, a suggestion is
  - $w_{iq} = (0.5 + [0.5 \cdot \text{freq}(i,q) / \max(\text{freq}(l,q))] \cdot \log(N/n_i))$

(Baeza-Yates & Ribeirp-Neto 1999)
The Vector Model

- The vector model with \textit{tf-idf} weights is a good ranking strategy with general collections.
- The vector model is usually as good as the known ranking alternatives. It is also simple and fast to compute.

Advantages:
- term-weighting improves quality of the answer set
- partial matching allows retrieval of docs that approximate the query conditions
- cosine ranking formula sorts documents according to degree of similarity to the query

Disadvantages:
- assumes independence of index terms (??); not clear that this is bad though

(Baeza-Yates & Ribeiro-Neto 1999)