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,
  $w_{ij} \in \{0,1\}$
- 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 recursivley

$R(t_i,d_j)$ = *TRUE, if* $w_{ij} = 1$ (i.e. t_i is in d_j)
= *FALSE, if* $w_{ij} = 0$ (i.e. t_i is not in d_j)

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

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

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

- 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

	d1	d2	d3
accident	1	0	1
car	1	1	0
cause	0	0	1
crowd	0	0	1
die	1	0	0
drive	0	0	1
four	0	0	1
heavy	1	0	0
injur	0	0	1
more	0	1	0
morning	1	0	0
people	1	0	1
quarter	0	1	0
register	0	1	0
truck	0	0	1
trucker	0	0	1
vehicle	0	1	0
vienna	1	1	1
yesterday	1	0	0

- Query:

(vehicle OR car) AND accident [Search]

R(vehicle OR car AND accident, d1) =
R(vehicle OR car AND accident, d2) =
R(vehicle OR car AND accident, d3) =

- Query:

(vehicle AND car) OR accident [Search]

R(vehicle AND car OR accident, d1) =
R(vehicle AND car OR accident, d2) =
R(vehicle AND car OR accident, d3) =

Processing Boolean Queries

Algorithm for the intersection of two posting list p1 und p2:

```
INTERSECT(p1, p2)
  answer ← ⟨ ⟩
  while p1 ≠ NIL and p2 ≠ NIL
  do if docID(p1) = docID(p2)
        then ADD(answer, docID(p1))
             p1 ← next(p1)
             p2 ← next(p2)
        else if docID(p1) < docID(p2)
                then p1 ← next(p1)
                else p2 ← next(p2)
  return answer
```

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

  car AND accident | Search

  - Locate "car" in the dictionary
  - Retrieve its postings
  - Locate "accident" in the dictionary
  - Retrieve its postings
  - Intersect the two posting lists
- Query Optimization: For more than two terms in a conjunctive query, start with two shortest posting lists

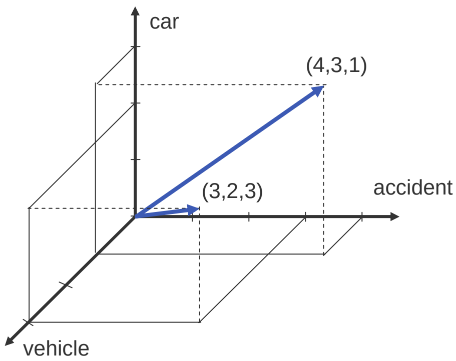
Drawbacks of the Boolean Model

- 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 OR t_2 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, $w_{ij} \in \{0,1\}$
- 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:

	d1	d2
accident	4	3
car	3	2
vehicle	1	3



- 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 *vec(i)*
  - The unitary vectors *vec(i)* and *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 $vec(d_j) = (w_{1j}, w_{2j}, \ldots, w_{kj})$
  - query vector $vec(q) = (w_{1q},\ldots,w_{kq})$
- 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 occuring query terms
- Ranking function: scalar product
  - R(q,d) = q * d

$$= \sum_{i=1}^{n} q_i * d_i$$

Multiply components and summarize

Coordinate Matching: Example

	d1	d2	d3	q
accident	1	0	1	1
car	1	1	0	0
cause	0	0	1	0
crowd	0	0	1	0
die	1	0	0	0
drive	0	0	1	0
four	0	0	1	0
heavy	1	0	0	1
injur	0	0	1	0
more	0	1	0	0
morning	1	0	0	0
people	1	0	1	0
quarter	0	1	0	0
register	0	1	0	0
truck	0	0	1	0
trucker	0	0	1	0
vehicle	0	1	0	1
vienna	1	1	1	1
yesterday	1	0	0	0

query vector represents terms of the query

accident heavy vehicles vienna [Search]

Resultat:

$q * d1 =$
$q * d2 =$
$q * d3 =$

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 (dissi-milarity)
    - *idf* factor, the *inverse document frequency*
  - $w_{ij} = tf(i,j) * idf(i)$

(Baeza-Yates & Ribeirp-Neto 1999)

TF - Term Frequency

	d1	d2	d3	q
accident	2	0	1	1
car	1	1	0	0
cause	0	0	1	0
crowd	0	0	1	0
die	1	0	0	0
drive	0	0	1	0
four	0	0	1	0
heavy	2	0	0	1
injur	0	0	1	0
more	0	2	0	0
morning	1	0	0	0
people	1	0	2	0
quarter	0	1	0	0
register	0	1	0	0
truck	0	0	1	0
trucker	0	0	1	0
vehicle	0	1	0	1
vienna	1	1	1	1
yesterday	1	0	0	0

For reasons of simplicity, in this example *f(i,j) = freq(i,j)*

- 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

(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
  - *idf(i) =* $1/n_i$
- A normalized *idf* factor is given by
  - *idf(i) = log* (N/n_i)

  the log is used to make the values of tf and idf comparable.

Example with TF and IDF

- In this examle a simple *tf* factor
  - *f(i,j) = freq(i,j)*

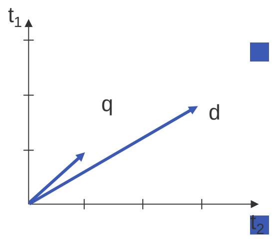
  and a simple idf factor
  - *idf(i) =* $1/n_i$

  are used

Indexing a new Document

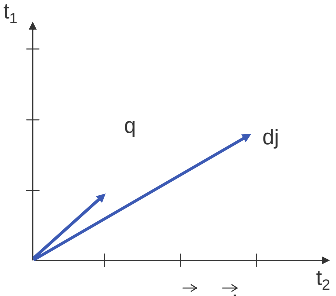
- Changes of the indexes when adding a new document d
  - a new document vector with tf factors for d is created
  - idf factors for terms occuring in d are adapted
- All other document vectors remain unchanged

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 beween 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



$$\cos(\vec{q},\vec{d_j}) = \frac{\vec{q} \circ \vec{d_j}}{|\vec{q}| \circ |\vec{d_j}|}$$

$$= \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{j=1}^{t} w_{i,q}^2}}$$

- 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 θ be the angle between q and d_j
- Because all values w_{ij} >= 0 the angle θ is between 0° und 90°
  - the larger θ the less is cos θ
  - the less θ the larger is cos θ
  - cos 0 = 1
  - cos 90° = 0

The Vector Model

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

(Baeza-Yates & Ribeirp-Neto 1999)

The Vector Model

- The vector model with *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)