Maschinelle Sprachverarbeitung
Retrieval Models and Implementation
Content of this Lecture

- Information Retrieval Models
  - Boolean Model
  - Vector Space Model
- Inverted Files
Information Retrieval Core

- The core question in IR: Which of a given set of (normalized) documents is relevant for a given query?
- Ranking: How relevant for a given query is each document?
How can Relevance be Judged?

Retrieval: Adhoc Filtering

Structured Models
  - Non-Overlapping Lists
  - Proximal Nodes

Classic Models
  - Boolean
  - Vector-Space
  - Probabilistic

Algebraic
  - Generalized Vector
  - Lat. Semantic Index
  - Neural Networks

Set Theoretic
  - Fuzzy
  - Extended Boolean

Probabilistic
  - Inference Network
  - Belief Network

User Task
  - Browsing

Flat Structure Guided Hypertext

[BYRN99]
Notation

- Most of the models we discuss use the “Bag of Words”
- Definition
  - Let $D$ be the set of all normalized documents, $d \in D$ is a document
  - Let $K$ be the set of all terms in $D$, $k_i \in K$ is a term
    - Can as well be tokens
  - Let $w$ be the function that maps a given $d$ to its multiset of distinct terms in $K$ (its bag-of-words)
  - Let $v_d$ by a vector of size $|K|$ for $d$ with
    - $v_d[i] = 0$ iff $k_i \notin w(d)$
    - $v_d[i] = 1$ iff $k_i \in w(d)$
  - Often, we use weights instead of a Boolean membership function
    - Let $w_{ij} \geq 0$ be the weight of term $k_i$ in document $d_j$ ($w_{ij} = v_j[i]$)
    - $w_{ij} = 0$ if $k_i \notin d_j$
Boolean Model

- Simple model based on set theory
- Queries are specified as **Boolean expressions** over terms
  - Terms connected by AND, OR, NOT, (XOR, ...)
  - Parenthesis are possible (but ignored here)
- Relevance of a document is either 0 or 1
  - Let q contain the atoms (terms) \(<k_1, k_2, \ldots>\)
  - An atom \(k_i\) evaluates to true for a document \(d\) iff \(v_d[k_i]=1\)
  - Compute truth values of all atoms for each \(d\)
  - Compute truth of \(q\) for \(d\) as logical expression over atom values
- Example: “(kaufen AND rad) OR NOT wir”
  - “wir kaufen ein rad” - \(<(T AND T) OR NOT T> = T\)
  - “sei kaufen ein auto” - \(<(T AND F) OR NOT F> = T\)
Properties

- Simple, clear semantics, widely used in (early) systems
- Disadvantages
  - No partial matching
    - Suppose query $k_1 \land k_2 \land \ldots \land k_9$
    - A doc $d$ with $k_1 \land k_2 \ldots k_8$ is as irrelevant as one with none of the terms
  - No ranking
  - Terms cannot be weighted
  - No synonyms, homonyms, semantically close words
  - Lay users don’t understand Boolean expressions
- Results: Often unsatisfactory
  - Too many documents (too few restrictions, many OR)
  - Too few documents (too many restrictions, many AND)
Content of this Lecture

• Information Retrieval Models
  – Boolean Model
  – Vector Space Model

• Inverted Files
Vector Space Model

  - A breakthrough in IR

- General idea
  - Fix vocabulary $K$ (the dictionary)
  - View each doc (and the query) as point in a $|K|$-dimensional space
  - Rank docs according to distance from the query in that space

- Main advantages
  - Inherent ranking (according to distance)
  - Naturally supports partial matching (increases distance)
Vector Space

- Each term is one dimension
  - Different suggestions for determining co-ordinates, i.e., term weights
- The closest docs are the most relevant ones
  - Rationale: Vectors correspond to themes which are loosely related to sets of terms
  - Distance between vectors ~ distance between themes
  - Different suggestions for defining distance
The Angle between Two Vectors

- Recall: The **scalar product** between two vectors $v$ and $w$ of equal dimension is defined as

$$v \circ w = |v| \cdot |w| \cdot \cos(v, w)$$

- This gives us the angle

$$\cos(v, w) = \frac{v \circ w}{|v| \cdot |w|}$$

- With

$$|v| = \sqrt{\sum v_i^2} \quad v \circ w = \sum_{i=1}^{n} v_i \cdot w_i$$
Distance as Angle

Distance = \textit{cosine of the angle} between doc d and query q

\[ \text{sim}(d, q) = \cos(v_d, v_q) = \frac{v_d \circ v_q}{|v_d| \times |v_q|} = \frac{\sum(v_q[i] \times v_d[i])}{\sqrt{\sum v_d[i]^2} \times \sqrt{\sum v_q[i]^2}} \]

- Length normalization
- Can be dropped for ranking
Example

- Assume stop word removal, stemming, and binary weights

<table>
<thead>
<tr>
<th></th>
<th>Text</th>
<th>verkauf</th>
<th>haus</th>
<th>italien</th>
<th>gart</th>
<th>miet</th>
<th>blüh</th>
<th>woll</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Wir verkaufen Häuser in Italien</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Häuser mit Gärten zu vermieten</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Häuser: In Italien, um Italien, um Italien herum</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Die italienschen Gärtner sind im Garten</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Der Garten in unserem italienschen Haus blüht</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Q</td>
<td>Wir wollen ein Haus mit Garten in Italien mieten</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Ranking

\[ \text{sim}(d, q) = \frac{\sum (v_q[i] \times v_d[i])}{\sqrt{\sum v_d[i]^2}} \]

- \( \text{sim}(d_1, q) = \frac{(1\times0+1\times1+1\times1+0\times1+0\times1+0\times0+0\times1)}{\sqrt{3}} \approx 1.15 \)
- \( \text{sim}(d_2, q) = \frac{(1+1+1)}{\sqrt{3}} \approx 1.73 \)
- \( \text{sim}(d_3, q) = \frac{(1+1)}{\sqrt{2}} \approx 1.41 \)
- \( \text{sim}(d_4, q) = \frac{(1+1)}{\sqrt{2}} \approx 1.41 \)
- \( \text{sim}(d_5, q) = \frac{(1+1+1)}{\sqrt{4}} \approx 1.5 \)

<table>
<thead>
<tr>
<th>Rg</th>
<th>Q: Wir wollen ein Haus mit Garten in Italien mieten</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>d_2: <strong>Häuser</strong> mit <strong>Gärten</strong> zu vermieten</td>
</tr>
<tr>
<td>2</td>
<td>d_5: Der <strong>Garten</strong> in unserem <strong>italienschen Haus</strong> blüht</td>
</tr>
<tr>
<td>3</td>
<td>d_4: Die <strong>italienschen Gärtner</strong> sind im <strong>Garten</strong></td>
</tr>
<tr>
<td></td>
<td>d_3: <strong>Häuser</strong>: In <strong>Italien</strong>, um <strong>Italien</strong>, um <strong>Italien</strong> herum</td>
</tr>
<tr>
<td>5</td>
<td>d_1: Wir verkaufen <strong>Häuser</strong> in <strong>Italien</strong></td>
</tr>
</tbody>
</table>
Introducing Term Weights

- **Definition**

  Let $D$ be a document collection, $K$ be the set of all terms in $D$, $d \in D$ and $k \in K$

  - The **relative term frequency** $tf_{dk}$ is the relative frequency of $k$ in $d$
  - The **document frequency** $df_k$ is the frequency of docs in $D$ containing $k$
    - This should rather be called “corpus frequency”
    - May also be defined as the frequency of occurrences of $k$ in $D$
    - Both definitions are valid and both are used
  - The **inverse document frequency** is defined as $idf_k = \frac{|D|}{df_k}$
    - In practice, one usually uses $idf_k = \log(\frac{|D|}{df_k})$
  - The $tf*idf$ score $w_{dk}$ of a term $k$ in a document $d$ is defined as

$$w_{dk} = \frac{tf_{dk} \cdot idf_k}{|d|}$$
Example TF*IDF

\[
sim(d, q) = \frac{\sum (v_q[i] * v_d[i])}{\sqrt{\sum v_d[i]^2}}
\]

- \( \text{sim}(d_1, q) = \frac{5/4 \times 1/3 + 5/4 \times 1/3}{\sqrt{3.13}} \approx 1.51 \)
- \( \text{sim}(d_2, q) = \frac{5/4 \times 1/3 + 5/3 \times 1/3 + 5 \times 1/3}{\sqrt{3.26}} \approx 4.80 \)
- \( \text{sim}(d_3, q) = \frac{5/4 \times 1/4 + 5/4 \times 3/4}{\sqrt{0.98}} \approx 1.57 \)
- \( \text{sim}(d_4, q) = \frac{5/4 \times 1/3 + 5/3 \times 2/3}{\sqrt{1.41}} \approx 2.08 \)
- \( \text{sim}(d_5, q) = \frac{5/4 \times 1/4 + 5/4 \times 1/4 + 5/3 \times 1/4}{\sqrt{1.93}} \approx 2.08 \)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>IDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 (tf)</td>
<td>1/3</td>
<td>1/3</td>
<td>1/3</td>
<td></td>
<td></td>
<td>5/4</td>
</tr>
<tr>
<td>2 (tf)</td>
<td>1/3</td>
<td></td>
<td>1/3</td>
<td>1/3</td>
<td>1/3</td>
<td>5/4</td>
</tr>
<tr>
<td>3 (tf)</td>
<td></td>
<td>1/4</td>
<td>3/4</td>
<td></td>
<td></td>
<td>5/4</td>
</tr>
<tr>
<td>4 (tf)</td>
<td></td>
<td></td>
<td></td>
<td>1/3</td>
<td>2/3</td>
<td>5/3</td>
</tr>
<tr>
<td>5 (tf)</td>
<td></td>
<td></td>
<td>1/4</td>
<td></td>
<td></td>
<td>5/4</td>
</tr>
</tbody>
</table>

wollen ein Haus mit Garten in Italien mieten

Häuser mit Gärten zu vermieten

Der Garten in unserem italienischen Haus blüht
Die italienischen Gärtner sind im Garten

Häuser: In Italien, um Italien, um Italien herum

Wir verkaufen Häuser in Italien
TF*IDF in Short

- Give terms in a doc d with high weights which are …
  - frequent in d and
  - infrequent in D

- IDF deals with the consequences of Zipf’s law
  - The few very frequent (and unspecific) terms get lower scores
  - The many infrequent (and specific) terms get higher scores

- Interferes with stop word removal
  - If stop words are removed, IDF might not be necessary any more
  - If IDF is used, stop word removal might not be necessary any more
BoW Shortcomings

• No treatment of **synonyms** (query expansion, …)
• No treatment of **homonyms**
  - Different senses = different dimensions
  - We would need to disambiguate terms into their senses (later)
• No consideration of **term order**
  - But order carries semantic meaning
• Assumes that all terms are **independent**
  - Clearly wrong: some terms are **semantically closer** than others
    • Their co-appearance doesn’t mean more than only one appearance
    • The appearance of “red” in a doc with “wine” doesn’t mean much
  - Extension: Topic-based Vector Space Model
    • Latent Semantic Indexing (see IR lecture)
Content of this Lecture

• Information Retrieval Models
  - Boolean Model
  - Vector Space Model

• Inverted Files
Full-Text Indexing

• Fundamental operation for all IR models: \texttt{find}( k, D)
  – Given a query term \textit{k}, find all docs from \textit{D} containing it

• Can be implemented using online search
  – Search all occurrence of \textit{k} in all docs from \textit{D}
    – Algorithms: Boyer-Moore, Knuth-Morris-Pratt, etc.

• But
  – We generally assume that \textit{D} is stable (compared to \textit{k})
  – We \textit{only search for discrete terms} (after tokenization)

• Consequence: Better to pre-compute a \textit{term index} over \textit{D}
  – Also called “full-text index”
Inverted Files (or Inverted Index)

- Simple and effective index structure for terms
- Builds on the Bag of words approach
  - We give up the order of terms in docs (see positional index later)
  - We cannot reconstruct docs based on index only
- Start from “docs containing terms” (≈ “docs”) and invert to “terms appearing in docs” (≈ “inverted docs”)

\[
\begin{align*}
  d1 & : t1, t3 \\
  d2 & : t1 \\
  d3 & : t2, t3 \\
  d4 & : t1 \\
  d5 & : t1, t2, t3 \\
  d6 & : t1, t2 \\
  d7 & : t2 \\
  d8 & : t2 \\
\end{align*}
\]

\[
\begin{align*}
  t1 & : d1, d2, d4, d5, d6 \\
  t2 & : d3, d5, d6, d7, d8 \\
  t3 & : d1, d3, d5 \\
\end{align*}
\]
### Building an Inverted File [Andreas Nürnberg, IR-2007]

#### Doc1:
Now is the time for all good men to come to the aid of their country.

#### Doc2:
It was a dark and stormy night in the country manor. The time was past midnight.

<table>
<thead>
<tr>
<th>Term</th>
<th>Doc ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>now</td>
<td>1</td>
</tr>
<tr>
<td>is</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>time</td>
<td>1</td>
</tr>
<tr>
<td>for</td>
<td>1</td>
</tr>
<tr>
<td>all</td>
<td>1</td>
</tr>
<tr>
<td>good</td>
<td>1</td>
</tr>
<tr>
<td>men</td>
<td>1</td>
</tr>
<tr>
<td>to</td>
<td>1</td>
</tr>
<tr>
<td>come</td>
<td>1</td>
</tr>
<tr>
<td>to</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>aid</td>
<td>1</td>
</tr>
<tr>
<td>of</td>
<td>1</td>
</tr>
<tr>
<td>their</td>
<td>1</td>
</tr>
<tr>
<td>country</td>
<td>1</td>
</tr>
<tr>
<td>it</td>
<td>2</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>dark</td>
<td>2</td>
</tr>
<tr>
<td>and</td>
<td>2</td>
</tr>
<tr>
<td>stormy</td>
<td>2</td>
</tr>
<tr>
<td>night</td>
<td>2</td>
</tr>
<tr>
<td>in</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
</tr>
<tr>
<td>country</td>
<td>2</td>
</tr>
<tr>
<td>manor</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
</tr>
<tr>
<td>time</td>
<td>2</td>
</tr>
<tr>
<td>to</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
</tr>
<tr>
<td>past</td>
<td>2</td>
</tr>
<tr>
<td>midnight</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Doc ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>aid</td>
<td>1</td>
</tr>
<tr>
<td>all</td>
<td>1</td>
</tr>
<tr>
<td>and</td>
<td>2</td>
</tr>
<tr>
<td>come</td>
<td>1</td>
</tr>
<tr>
<td>country</td>
<td>1</td>
</tr>
<tr>
<td>dark</td>
<td>2</td>
</tr>
<tr>
<td>for</td>
<td>1</td>
</tr>
<tr>
<td>good</td>
<td>1</td>
</tr>
<tr>
<td>in</td>
<td>2</td>
</tr>
<tr>
<td>is</td>
<td>1</td>
</tr>
<tr>
<td>it</td>
<td>2</td>
</tr>
<tr>
<td>manor</td>
<td>2</td>
</tr>
<tr>
<td>men</td>
<td>1</td>
</tr>
<tr>
<td>midnight</td>
<td>2</td>
</tr>
<tr>
<td>night</td>
<td>2</td>
</tr>
<tr>
<td>now</td>
<td>1</td>
</tr>
<tr>
<td>of</td>
<td>1</td>
</tr>
<tr>
<td>past</td>
<td>2</td>
</tr>
<tr>
<td>stormy</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>their</td>
<td>2</td>
</tr>
<tr>
<td>time</td>
<td>1</td>
</tr>
<tr>
<td>time</td>
<td>2</td>
</tr>
<tr>
<td>to</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Sort

#### Merge

<table>
<thead>
<tr>
<th>Term</th>
<th>Doc ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>aid</td>
<td>1</td>
</tr>
<tr>
<td>all</td>
<td>1</td>
</tr>
<tr>
<td>and</td>
<td>2</td>
</tr>
<tr>
<td>come</td>
<td>1</td>
</tr>
<tr>
<td>country</td>
<td>1, 2</td>
</tr>
<tr>
<td>dark</td>
<td>2</td>
</tr>
<tr>
<td>for</td>
<td>1</td>
</tr>
<tr>
<td>good</td>
<td>1</td>
</tr>
<tr>
<td>in</td>
<td>2</td>
</tr>
<tr>
<td>is</td>
<td>1</td>
</tr>
<tr>
<td>it</td>
<td>2</td>
</tr>
<tr>
<td>manor</td>
<td>2</td>
</tr>
<tr>
<td>men</td>
<td>1</td>
</tr>
<tr>
<td>midnight</td>
<td>2</td>
</tr>
<tr>
<td>night</td>
<td>2</td>
</tr>
<tr>
<td>now</td>
<td>1</td>
</tr>
<tr>
<td>of</td>
<td>1</td>
</tr>
<tr>
<td>past</td>
<td>2</td>
</tr>
<tr>
<td>stormy</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1, 2</td>
</tr>
<tr>
<td>their</td>
<td>1, 2</td>
</tr>
<tr>
<td>time</td>
<td>1, 2</td>
</tr>
<tr>
<td>to</td>
<td>2</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
</tr>
</tbody>
</table>
Boolean Retrieval

- For each query term \( k_i \), look-up doc-list \( D_i \) containing \( k_i \)
- Evaluate query in the usual order
  - \( k_i \land k_j : D_i \cap D_j \)
  - \( k_i \lor k_j : D_i \cup D_j \)
  - NOT \( k_i : D \setminus D_i \)
- Example

\[
\text{(time AND past AND the) OR (men)}
\]
\[
= (D_{\text{time}} \cap D_{\text{past}} \cap D_{\text{the}}) \cup D_{\text{men}}
\]
\[
= (\{1,2\} \cap \{2\} \cap \{1,2\}) \cup \{1\}
\]
\[
= \{1,2\}
\]
Necessary and Obvious Tricks

• How do we support union and intersection efficiently?
  - Naïve algorithm requires $O(|D_i| \times |D_j|)$
  - Better: Keep doc-lists sorted
    - Intersection $D_i \cap D_j$: Sort-Merge in $O(|D_i| + |D_j|)$
    - Union $D_i \cup D_j$: Sort-Merge in $O(|D_i| + |D_j|)$
    - If $|D_i| << |D_j|$, use binsearch in $D_j$ for all terms in $D_i$
      • Whenever $|D_i| + |D_j| > |D_i| \times \log(|D_j|)$

• How do we efficiently look-up doc-list $D_i$?
  - Inverted file is very large – store on disc
  - Bin-search on inverted file: $O(\log(|K|))$ access to disk
  - Inefficient: Random access
Dictionary and Posting List

- Split up inverted file into dictionary and posting list
  - Dictionary is not very large – keep in memory
  - Each entry maintains a pointer to its posting list
  - Posting lists are on disk
  - 1 IO for finding posting list for a given term
Adding Term Weighting

- VSM with TF*IDF requires (relative) term frequencies
  - Dictionary stored IDF (or DF) per term
  - For each docID entry in posting list, add (relative) term frequency
Searching in VSM

• Assume we want to retrieve the top-r docs

• Algorithm
  - Initialize an empty doc-list $S$ (as hash table or priority queue)
  - Iterate through query terms $k_i$
    • Walk through posting list (elements $(docID, TF)$)
      - If $docID \in S$: $S[docID] \leftarrow IDF[k_i] \times TF$
      - else: $S = S.append( (docID, IDF[k_i] \times TF))$
    • Length-normalize value and compute cosine
  - Return top-r docs in $S$

• $S$ contains all and only those docs containing at least one $k_i$
Space Usage

- **Size of dictionary:** $O(|K|)$
  - Zipf’s law: From a certain corpus size on, new terms appear only very infrequently
    - But there are always new terms, no matter how large $D$
    - Example: 1GB text (TREC-2) generates only 5MB dictionary
  - Typically: $<1$ Million
    - Many more in multi-lingual corpora, web corpora, etc.

- **Size of posting list**
  - Theoretic worst case: $O(|K|*|D|)$
  - Practical: A few hundred entries for each doc in $D$
Storing the Dictionary

- Dictionary as array (keyword, DF, ptr)
- Since keywords have different lengths: Implementation will be (ptr1, DF, ptr2)
  - ptr1: To string (the keyword)
  - ptr2: To posting list
- Search: Compute \( \log(|K|) \) memory addresses, follow ptr1, compare strings: \( O(\log(|K|) \times |k|) \)
- Construction: Essentially for free
- Alternatives: Hashing, Keyword Trees
Storing the Posting File

- Posting file is usually kept on disk
- Thus, we need an **IO-optimized data structure**
- **Static**
  - Store posting lists **one after the** other in large file
  - Posting-ptr is (large) offset in this file
- **Prepare for inserts**
  - Reserve additional space per posting
    - Good idea: Large initial posting lists get large extra space
    - Many inserts can be handled internally
  - Upon **overflow**, append entire posting list at the end of the file
    - Place **pointer at old position** - at most two access per posting list
  - Can lead to many holes - requires regular **reorganization**
Positional Information

- What if we search for phrases: “Bill Clinton”, “Ulf Leser”
  - ~10% of web searches are phrase queries
- What if we search by proximity “car AND rent/5”
  - “We rent cars”, “cars for rent”, “special care rent”, “if you want to rent a car, click here”, “Cars and motorcycles for rent”, ...
- We need positional information

**Doc1:**
Now is the time for all good men to come to the aid of their country. It was a dark and stormy night in the country manor. The time was past midnight.
Answering Phrase Queries

- Search posting lists of all query terms
- During intersection, also positions must fit
Effects

- Dictionary is not affected
- Posting lists get much larger
  - Store many \(<\text{docID}, \text{pos}>, \text{TF}\>\) instead of few \(<\text{docID}, \text{TF}\>\)
  - Index with positional information typically 30-50% larger than the corpus itself
  - Especially frequent words require excessive storage
- Use compression
Self Assessment

• Explain the vector space model
• How is the size of K (vocabulary) influenced by pre-processing?
• Describe some variations of deducing term weights
• How could we extend the VSM to also consider the order of terms (to a certain degree)?
• Explain idea and structure of inverted files?
• What are possible data structures for the dictionary? Advantages / disadvantages?
• What decisions influence the size of posting lists?