Basic ranking Models

Boolean and Vector Space Models
Document Processing

Indexing

Ranking
Boolean Model
Two “classic” models

- Boolean
- Vector Space
The Boolean Model

- Simple model based on set theory
- Queries and documents specified as boolean expressions
  - Precise semantics
  - E.g., \( q = ka \land (kb \lor \neg kc) \)
  - \((apple \land (computer \lor \neg red))\)
- Terms are either present or absent. Thus, \( wij \in \{0,1\} \)
Example

$q = ka \land (kb \lor \neg kc)$

$v(q_{dnf}) = (1,1,1) (1,1,0) (1,0,0)$  
» Disjunctive Normal Form

(\text{apple, computer, red}) \lor (\text{apple, computer}) \lor (\text{apple})

$v(q_{cc}) = (1,1,0)$
» Conjunctive Component

• Similar/Matching documents
  • $md1 = [\text{apple apple blue day}] \Rightarrow (1,0,0)$
  • $md2 = [\text{apple computer red}] \Rightarrow (1,1,1)$

• Unmatched documents
  • $ud1 = [\text{apple red}] \Rightarrow (1,0,1)$
  • $ud2 = [\text{day}] \Rightarrow (0,0,0)$
Similarity/Matching function

\[ \text{sim}(q, dj) = 1 \text{ if } \text{vec}(dj) = v(qcc)_i, v(qcc)_i \in v(qdnf) \]
\[ 0 \text{ otherwise} \]

In other terms, only documents whose vector are one of the conjunctive components of the query disjunctive normal form
\[ q = \text{ka} \land ((\text{kb} \lor \neg \text{kc})) \]
Drawbacks of the Boolean Model

- Expressive power of boolean expressions to capture information need and document semantics *inadequate*
- Retrieval based on binary decision criteria (with no partial match) does not reflect our intuitions behind relevance adequately

• As a result
  - Answer set contains either too few or too many documents in response to a user query
  - No ranking of documents
Boolean Search

• Boolean query almost disappeared from web search engines (not used by most users)
• “Advanced search” allows for other types of search

Still used in dedicated search engines (e.g. Westlaw legal web search engine)
Vector Model
Ranked retrieval

• Thus far, our queries have all been Boolean.
  – Documents either match or don’t.
  – Good for expert users with precise understanding of their needs and the collection (e.g., legal search).
  – Not good for the majority of users.
  – Most users incapable of writing Boolean queries (or they are, but they think it’s too much work).
  – Most users don’t want to wade through 1000s of results (e.g., web search).
Problem with Boolean search

• Boolean queries often result in either too few (=0) or too many (1000s) results.
  – Query 1: “standard user dlink 650” → 200,000 hits
  – Query 2: “standard user dlink 650 no card found”: 0 hits

• It takes skill to come up with a query that produces a manageable number of hits.

• With a ranked list of documents, it does not matter how large the retrieved set is.
Scoring as the basis of ranked retrieval

• We wish to return *in order of relevance* the documents most likely to be useful to the searcher

• How can we rank-order the documents in the collection with respect to a query?

• Assign a score – say in [0, 1] – to each document

• This score measures how well document and query “match”.

L08VSM-tfidf
Query-document matching scores

• We need a way of assigning a score to a query/document pair
• Let’s start with a one-term query
• If the query term does not occur in the document: score should be 0
• The more frequent the query term in the document, the higher the score (should be)
• We will look at a number of alternatives for this.
### Binary term-document incidence matrix

<table>
<thead>
<tr>
<th>words</th>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brutus</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>worser</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$
Term-document **count matrices**

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector in $\mathbb{N}^v$

<table>
<thead>
<tr>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>157</td>
<td>73</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Brutus</td>
<td>4</td>
<td>157</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>232</td>
<td>227</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>57</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>worser</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

L08VSM-tfidf
**Bag of words model**

- Vector representation doesn’t consider the ordering of words in a document
  - *D1: John is quicker than Mary* and *D2: Mary is quicker than John* have the same vectors

- This is called (as we said in previous lessons) the **bag of words** model.
  - In a sense, this is a step back: the **positional index** (see lectures on indexing) was able to distinguish these two documents.
Term frequency $tf$

- The term frequency $tf_{t,d}$ of term $t$ in document $d$ is defined as the number of times that $t$ occurs in $d$.
- We want to use $tf$ when computing query-document match scores. But how?
- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term may be more relevant than a document with one occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.
Log-frequency weighting

• The log frequency weight of term $t$ in $d$ is

$$w_{t,d} = \begin{cases} 1 + \log_{10} \text{tf}_{t,d}, & \text{if } \text{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

• $0 \rightarrow 0$, $1 \rightarrow 1$, $2 \rightarrow 1.3$, $10 \rightarrow 2$, $1000 \rightarrow 4$, etc.

• Score for a document-query pair: sum over terms $t$ in both $q$ and $d$:

$$\text{score} = \sum_{t \in q \cap d} \left( 1 + \log \text{tf}_{t,d} \right)$$

• The score is 0 if none of the query terms is present in the document.
Document frequency

- Rare terms are more informative than frequent terms
  - Recall stop words!
  - Consider a term in the query that is rare in the collection (e.g., *arachnocentric*)
  - A document containing this term is very likely to be relevant to the query *arachnocentric*
  - → We want a higher weight for rare terms like *arachnocentric*.
Document frequency, continued

- Consider a query term that is frequent in the collection (e.g., high, increase, line)
  - A document containing such a term is more likely to be relevant than a document that doesn’t, but it’s not a sure indicator of relevance.
  - → For frequent terms, we want positive weights for words like high, increase, and line, but lower weights than for rare terms.

- We will use document frequency (df) to capture this in the score.

- df (≤ N) is the number of documents that contain the term
idf weight

• $df_t$ is the document frequency of $t$: the number of documents that contain $t$
  – $df$ is a measure of the informativeness of $t$

• We define the idf (inverse document frequency) of $t$ by

$$idf_t = \log_{10} \frac{N}{df_t}$$

  – We use $\log N/df_t$ instead of $N/df_t$ to “dampen” the effect of idf.

Will turn out that the base of the log is immaterial.
idf example, suppose $N = 1$ million

<table>
<thead>
<tr>
<th>term</th>
<th>$df_t$</th>
<th>$idf_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>calpurnia</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>animal</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>sunday</td>
<td>1,000</td>
<td>3</td>
</tr>
<tr>
<td>fly</td>
<td>10,000</td>
<td>2</td>
</tr>
<tr>
<td>under</td>
<td>100,000</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1,000,000</td>
<td>0</td>
</tr>
</tbody>
</table>

There is one idf value for each term $t$ in a collection.
Collection vs. Document frequency

- The **collection frequency** of $t$ is the number of occurrences of $t$ in the collection, **counting multiple occurrences**.

<table>
<thead>
<tr>
<th>Word</th>
<th>Collection frequency</th>
<th>Document frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>insurance</td>
<td>10440</td>
<td>3997</td>
</tr>
<tr>
<td>try</td>
<td>10422</td>
<td>8760</td>
</tr>
</tbody>
</table>

- Which word is a better search term (and should get a higher weight)?
tf-idf weighting

• The tf-idf weight of a term is the product of its tf weight and its idf weight.

\[
w_{t,d} = (1 + \log \text{tf}_{t,d}) \times \log \frac{N}{\text{df}_t}
\]

• Best known weighting scheme in information retrieval
  • Note: the “-” in tf-idf is a hyphen, not a minus sign!
  • Alternative names: tf.idf, tf x idf

• Increases with the number of occurrences within a document
• Increases with the rarity of the term in the collection
Binary $\rightarrow$ count $\rightarrow$ weight matrix

<table>
<thead>
<tr>
<th></th>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>5.25</td>
<td>3.18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.35</td>
</tr>
<tr>
<td>Brutus</td>
<td>1.21</td>
<td>6.1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>8.59</td>
<td>2.54</td>
<td>0</td>
<td>1.51</td>
<td>0.25</td>
<td>0</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1.54</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>2.85</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1.51</td>
<td>0</td>
<td>1.9</td>
<td>0.12</td>
<td>5.25</td>
<td>0.88</td>
</tr>
<tr>
<td>worser</td>
<td>1.37</td>
<td>0</td>
<td>0.11</td>
<td>4.15</td>
<td>0.25</td>
<td>1.95</td>
</tr>
</tbody>
</table>

Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$
Documents as vectors

• So we have a $|V|$-dimensional vector space
• Terms are axes of the space
• Documents are points or vectors in this space

• Very high-dimensional: hundreds of millions of dimensions when you apply this to a web search engine
• This is a very sparse vector - most entries are zero (will see later in this course how to reduce dimensionality).
Queries as vectors

- **Key idea 1**: Do the same for queries: represent them as vectors in the space
- **Key idea 2**: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity \(\approx\) inverse of distance
- Recall: We do this because we want to get away from the you’re-either-in-or-out Boolean model.
- Instead: rank more relevant documents higher than less relevant documents
Formalizing vector space proximity

• First cut: distance between two points
  – ( = distance between the end points of the two vectors)
• Euclidean distance?

$$d(d_j, q) = \sqrt{\sum_i (w_{ij} - w_{iq})^2}$$

• Euclidean distance is a bad idea . . .
• . . . because Euclidean distance is large for vectors of different lengths.
Why Euclidean distance is a bad idea

The Euclidean distance between $\mathbf{q}$ and $\mathbf{d}_2$ is large even though the distribution of terms in the query $\mathbf{q}$ and the distribution of terms in the document $\mathbf{d}_2$ are very similar (about 50% gossip, 50% Jealous). Absolute frequencies cause the difference.
Use angle instead of distance

• Experiment: take a document d and append it to itself. Call this document d’.
• “Semantically” d and d’ have the same content
• The Euclidean distance between the two documents can be quite large (word frequency doubles in d’)
• The angle between the two documents is 0, corresponding to maximal similarity.
• Key idea: Rank documents according to angle with query.
From angles to cosines

• The following two notions are equivalent.
  – Rank documents in decreasing order of the angle between query and document
  – Rank documents in increasing order of \(\text{cosine}(\text{query}, \text{document})\)

• Cosine is a monotonically decreasing function for the interval \([0^\circ, 180^\circ]\)
Length normalization

• A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L₂ norm:

\[ \| \vec{x} \|_2 = \sqrt{\sum_i x_i^2} \]

– Dividing a vector by its L₂ norm makes it a unit (length) vector
– Effect on the two documents d and d’ (d appended to itself) from earlier slide: they have identical vectors after length-normalization.
cosine(query, document)

\[
\cos(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|q||d|} = \frac{\bar{q} \cdot \bar{d}}{\|q\| \|d\|} = \frac{\sum_{i=1}^{V} q_i d_i}{\sqrt{\sum_{i=1}^{V} q_i^2} \sqrt{\sum_{i=1}^{V} d_i^2}}
\]

- \(q_i\) is the tf-idf weight of term \(i\) in the query
- \(d_i\) is the tf-idf weight of term \(i\) in the document
- \(\cos(q,d)\) is the cosine similarity of \(\vec{q}\) and \(\vec{d}\) ... or, equivalently, the cosine of the angle between \(\vec{q}\) and \(\vec{d}\).
Computing Similarity Scores

\[ D_1 = (0.8, 0.3) \]
\[ D_2 = (0.2, 0.7) \]
\[ Q = (0.4, 0.8) \]
\[ \cos \alpha_1 = 0.74 \]
\[ \cos \alpha_2 = 0.98 \]
A complete example

A small collection of 3 documents

\begin{align*}
d1: & \text{“new york times”} \\
d2: & \text{“new york post”} \\
d3: & \text{“los angeles times”}
\end{align*}

Compute idf

\begin{align*}
\text{angles} & \quad \log_2(3/1)=1.584 \\
\text{los} & \quad \log_2(3/1)=1.584 \\
\text{new} & \quad \log_2(3/2)=0.584 \\
\text{post} & \quad \log_2(3/1)=1.584 \\
\text{times} & \quad \log_2(3/2)=0.584 \\
\text{york} & \quad \log_2(3/2)=0.584
\end{align*}
A complete example

Document-term matrix (no need to normalize, every word occurs just once)

<table>
<thead>
<tr>
<th></th>
<th>angeles</th>
<th>los</th>
<th>new</th>
<th>post</th>
<th>times</th>
<th>york</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>d2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>d3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

tf-idf

<table>
<thead>
<tr>
<th></th>
<th>angeles</th>
<th>los</th>
<th>new</th>
<th>post</th>
<th>times</th>
<th>york</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1</td>
<td>0</td>
<td>0</td>
<td>0.584</td>
<td>0</td>
<td>0.584</td>
<td>0.584</td>
</tr>
<tr>
<td>d2</td>
<td>0</td>
<td>0</td>
<td>0.584</td>
<td>1.584</td>
<td>0</td>
<td>0.584</td>
</tr>
<tr>
<td>d3</td>
<td>1.584</td>
<td>1.584</td>
<td>0</td>
<td>0</td>
<td>0.584</td>
<td>0</td>
</tr>
</tbody>
</table>
A complete example

Query: “new new times”

When computing the *tf-idf* values for the query terms we divide the frequency by the maximum frequency (2) and multiply with the *idf* values.

| q | 0  | 0  | \((2/2)\times0.584=0.584\) | 0  | \((1/2)\times0.584=0.292\) | 0  |

We calculate the length (the NORM) of each document vector and of the query:

Length of \(d_1\) = \(\sqrt{0.584^2+0.584^2+0.584^2}\) = 1.011
Length of \(d_2\) = \(\sqrt{0.584^2+1.584^2+0.584^2}\) = 1.786
Length of \(d_3\) = \(\sqrt{1.584^2+1.584^2+0.584^2}\) = 2.316
Length of \(q\) = \(\sqrt{0.584^2+0.292^2}\) = 0.652
A complete example

Then the similarity values are:

$$\cos(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{\|\vec{q}\| \|\vec{d}\|} = \frac{\vec{q} \cdot \vec{d}}{\sqrt{\sum_{i=1}^{|\vec{q}|} q_i^2} \sqrt{\sum_{i=1}^{|\vec{d}|} d_i^2}} = \frac{\sum_{i=1}^{|\vec{q}|} q_i d_i}{\sqrt{\sum_{i=1}^{|\vec{q}|} q_i^2} \sqrt{\sum_{i=1}^{|\vec{d}|} d_i^2}}$$

$\cos\text{Sim}(d_1, q) = (0*0+0*0+0.584*0.584+0*0+0.584*0.292+0.584*0) / (1.011*0.652) = 0.776$

$\cos\text{Sim}(d_2, q) = (0*0+0*0+0.584*0.584+1.584*0+0*0.292+0.584*0) / (1.786*0.652) = 0.292$

$\cos\text{Sim}(d_3, q) = (1.584*0+1.584*0+0*0.584+0*0+0.584*0.292+0*0) / (2.316*0.652) = 0.112$

According to the similarity values, the final order in which the documents are presented as result to the query will be: d1, d2, d3.
Documents in Vector Space
Can be used also to measure similarity between documents

How similar are the novels:

**SaS**: Sense and Sensibility

**PaP**: Pride and Prejudice, and

**WH**: Wuthering Heights?

<table>
<thead>
<tr>
<th>term</th>
<th>SaS</th>
<th>PaP</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>affection</td>
<td>115</td>
<td>58</td>
<td>20</td>
</tr>
<tr>
<td>jealous</td>
<td>10</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>gossip</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>wuthering</td>
<td>0</td>
<td>0</td>
<td>38</td>
</tr>
</tbody>
</table>

Cosine similarity amongst 3 documents

Term frequencies (counts)
3 documents example contd.

Log frequency weighting

<table>
<thead>
<tr>
<th>term</th>
<th>SaS</th>
<th>PaP</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>affection</td>
<td>3.06</td>
<td>2.76</td>
<td>2.30</td>
</tr>
<tr>
<td>jealous</td>
<td>2.00</td>
<td>1.85</td>
<td>2.04</td>
</tr>
<tr>
<td>gossip</td>
<td>1.30</td>
<td>0</td>
<td>1.78</td>
</tr>
<tr>
<td>wuthering</td>
<td>0</td>
<td>0</td>
<td>2.58</td>
</tr>
</tbody>
</table>

After normalization

<table>
<thead>
<tr>
<th>term</th>
<th>SaS</th>
<th>PaP</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>affection</td>
<td>0.789</td>
<td>0.832</td>
<td>0.524</td>
</tr>
<tr>
<td>jealous</td>
<td>0.515</td>
<td>0.555</td>
<td>0.465</td>
</tr>
<tr>
<td>gossip</td>
<td>0.335</td>
<td>0</td>
<td>0.405</td>
</tr>
<tr>
<td>wuthering</td>
<td>0</td>
<td>0</td>
<td>0.588</td>
</tr>
</tbody>
</table>

\[
\cos(SaS,PaP) \approx 0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0 + 0 \times 0 \\
\approx 0.94
\]

\[
\cos(SaS,WH) \approx 0.79
\]

\[
\cos(PaP,WH) \approx 0.69
\]
Computing cosine scores

**CosineScore**\( (q) \)

1. float Scores[\(N\)] = 0
2. float Length[\(N\)]
3. for each query term \(t\)
4. do calculate \(w_{t,q}\) and fetch postings list for \(t\)
5. for each pair \((d, tf_{t,d})\) in postings list
6. do Scores[\(d\)] += \(w_{t,d} \times w_{t,q}\)
7. Read the array Length
8. for each \(d\)
9. do Scores[\(d\)] = Scores[\(d\)] / Length[\(d\)]
10. return Top \(K\) components of Scores[]
tf-idf weighting has many variants

<table>
<thead>
<tr>
<th>Term frequency</th>
<th>Document frequency</th>
<th>Normalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>n (natural)</td>
<td>n (no)</td>
<td>n (none)</td>
</tr>
<tr>
<td>$t_{f,t,d}$</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>l (logarithm)</th>
<th>t (idf)</th>
<th>c (cosine)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 + \log(t_{f,t,d})$</td>
<td>$\log \frac{N}{df_t}$</td>
<td>$\frac{1}{\sqrt{w_1^2 + w_2^2 + \ldots + w_M^2}}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a (augmented)</th>
<th>p (prob idf)</th>
<th>u (pivoted unique)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0.5 + \frac{0.5\times t_{f,t,d}}{\max_t(t_{f,t,d})}$</td>
<td>$\max{0, \log \frac{N - df_t}{df_t}}$</td>
<td>$1/u$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>b (boolean)</th>
<th>b (byte size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\begin{cases} 1 &amp; \text{if } t_{f,t,d} &gt; 0 \ 0 &amp; \text{otherwise} \end{cases}$</td>
<td>$1/\text{CharLength}^\alpha$, $\alpha &lt; 1$</td>
</tr>
</tbody>
</table>

Augmented used to assign same relevance to very rare words
Weighting may differ in queries vs documents

• Many search engines allow for different weightings for queries vs documents (queries are very short, each word occurs typically once)

• To denote the combination in use in an engine, we use the notation qqq.ddd with the acronyms from the previous table

• Example: ltn.lnc means:
  – Query: logarithmic tf (l in leftmost column), idf (t in second column), no normalization ...
  – Document: logarithmic tf, no idf and cosine normalization
Summary – vector space ranking

• Represent the query as a weighted tf-idf vector
• Represent each document as a weighted tf-idf vector

• Compute the cosine similarity score for the query vector and each document vector

• Rank documents with respect to the query by score
• Return the top $K$ (e.g., $K = 10$) to the user
Summary: What’s the point of using vector spaces?

• A well-formed algebraic space for retrieval
• Query becomes a vector in the same space as the docs.
  – Can measure each doc’s proximity to it.
• Natural measure of scores/ranking – no longer Boolean.
  – Documents and queries are expressed as bags of words
The Vector Model

- Non-binary (numeric) term weights used to compute *degree of similarity* between a query and each of the documents.

- Enables
  - *partial matches*
    - to deal with incompleteness
  - *answer set ranking*
    - to deal with information overload
The Vector Model: Pros and Cons

• Advantages:
  - term-weighting improves answer set quality
  - 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
Google ranking method

• Ranking is based on the content and on the specific page (later in this course, PageRank)
• Basically, keywords are interpreted as a boolean AND search (advanced options for complex boolean queries)
• However, answers are returned even if a word is not included (basically, it is a mixed boolean-vector space model)
• Additionally, query words are spell-corrected, and additional words can be added (see Query Expansion)