Chapter 6

Scoring, Term Weighting, and the Vector Space Model
Problems with Boolean Queries

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.

Also good for applications: Applications can easily consume 1000s of results.

Not good for the majority of users, who are unable or unwilling to write Boolean queries

Most users don’t want to wade through 1000s of results.

This is particularly true of web search.
Ranked Retrieval

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 a lot of 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.
We wish to return in order 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?

We need a way of assigning a score to a query/document pair.

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

Scoring Documents
Query-Document Matching Scores

Let’s start with a simple approach

Count how many of the query terms appear in a document:

Score = |Q ∩ D|

Can be computed easily

However, is very biased towards large documents

Large documents have a greater probability of getting a higher score (they just contain more terms)

Bigger is not always better . . .
Jaccard Coefficient

- We need some way of normalizing the score
- Why not use Jaccard coefficient?

\[
\text{JACCARD}(A, B) = \frac{|A \cap B|}{|A \cup B|}
\]

- \text{JACCARD}(A, A) = 1
- \text{JACCARD}(A, B) = 0 \text{ if } A \cap B = 0
- \text{A and B don’t have to be the same size.}
- Always assigns a number between 0 and 1.
What’s Wrong with Jaccard?

- Having a higher term frequency makes a document more relevant.
  - How many occurrences does a term have in a document?
  - Rare terms are more informative than frequent terms.
  - How often does a term occur in a document collection?
- Jaccard doesn’t consider this information.
- We need a more sophisticated way of normalizing for length.
Binary Incidence Matrix

Up to now, we used a binary incidence matrix

<table>
<thead>
<tr>
<th></th>
<th>Anthony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anthony</td>
<td>1</td>
<td>1</td>
<td>0</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>
<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>
<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>
<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>
<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>
<td>1</td>
</tr>
<tr>
<td>worse</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Each document represented by binary vector $\in \{0, 1\}^{|V|}$. 
Term Frequency Matrix

We will now use a matrix containing the term frequencies:

<table>
<thead>
<tr>
<th></th>
<th>Anthony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anthony</td>
<td>157</td>
<td>73</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Brutus</td>
<td>4</td>
<td>157</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Caesar</td>
<td>232</td>
<td>227</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Cleopatra</td>
<td>57</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>mercy</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>worser</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Each document represented by count vector $\in \mathbb{N}^{|V|}$
Bag of Words Model

- For now, we do not consider the order of words in a document.
- “John is quicker than Mary” and “Mary is quicker than John” are represented the same way.
- This is called a *bag of words model*.
- In a sense, this is a step back: The positional index was able to distinguish these two documents.
- We will look at recovering positional information later in this course.
The term frequency $t_f$ of term $t$ in document $d$ is defined as the *number of times that $t$ occurs in $d$*. We want to use $t_f$ when computing query-document match scores. However, raw term frequency is not what we want. A document with 10 occurrences of the term is 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.
Term Frequency Weighting

- The effect of non-proportional increases can be seen in other areas as well
  - Economics: Law of Diminishing Returns (e.g. sowing)
  - Biology: human senses operate logarithmically (10 times increase in sound volume is perceived as being twice as loud)
- Term frequencies can be weighted in a similar way
Log Frequency Weighting

- The log frequency weight of term $t$ in $d$ is defined as follows:

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

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

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

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

- The score is 0 if none of the query terms is present in the document.
A document containing a query term is more likely to be relevant than a document that doesn’t, but that’s not the whole story.

Rare terms are more informative than frequent terms.

For instance, a collection of documents on the auto industry is likely to have the term `auto` in almost every document.

A document containing the term `auto` is not very relevant for a query containing the term `auto`.

Now, 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.
We want a high weight for rare terms

We want low weights (but still larger than 0) for common terms

We will use document frequency to factor this into computing the matching score.

The document frequency is *the number of documents in the collection that the term occurs in*.

The higher the document frequency, the lower the weight (and vice versa)
Inverse Document Frequency

- $df_t$ is the document frequency, the number of documents that $t$ occurs in (with $N$ documents in the collection).
- $df$ is an inverse measure of the *informativeness* of the term.
- We define the *idf weight* of term $t$ as follows (note the logarithmic weighting):

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

- idf is a measure of the *informativeness* of the term.
- We use the log transformation for both term frequency and document frequency.
**Effect on Ranking**

- idf affects the ranking of documents only if the query has at least two terms.
- For example, in the query “arachnocentric line”, idf weighting increases the relative weight of *arachnocentric* and decreases the relative weight of *line*.
- idf has no effect on ranking for one-term queries.
The tf-idf weight of a term is the **product of its tf weight and its idf weight**:

$$w_{t,d} = (1 + \log \text{tf}_{t,d}) \cdot \log \frac{N}{df_t}$$

One of the 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
### Weight Matrix

|               | Anthony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth | ... |
|---------------|------------------------|--------------|-------------|--------|---------|---------|
| Anthony       | 5.25                   | 3.18         | 0.0         | 0.0    | 0.0     | 0.35    |
| Brutus        | 1.21                   | 6.10         | 0.0         | 1.0    | 0.0     | 0.0     |
| Caesar        | 8.59                   | 2.54         | 0.0         | 1.51   | 0.25    | 0.0     |
| Calpurnia     | 0.0                    | 1.54         | 0.0         | 0.0    | 0.0     | 0.0     |
| Cleopatra     | 2.85                   | 0.0          | 0.0         | 0.0    | 0.0     | 0.0     |
| mercy         | 1.51                   | 0.0          | 1.90        | 0.12   | 5.25    | 0.88    |
| worser        | 1.37                   | 0.0          | 0.11        | 4.15   | 0.25    | 1.95    |

Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$. 
So we have a $|V|$-dimensional real-valued vector space.

Terms are axes of the space.

Documents are points or vectors in this space.

Very high-dimensional: tens of millions of dimensions when you apply this to a web search engine.

This is a very sparse vector - most entries are zero.
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
- proximity = similarity
- Recall: We’re doing 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 Similarity

- First cut: distance between two points
  - ( = distance between the end points of the two vectors)
  - Euclidean distance?
  - Euclidean distance is a bad idea . . .
  - . . . because Euclidean distance is large for vectors of different lengths.
Why Distance is a Bad Idea

The Euclidean distance of $\vec{q}$ and $\vec{d}_2$ is large although the distribution of terms in the query $q$ and the distribution of terms in the document $d_2$ are very similar.
Use Angle Instead of Distance

- Rank documents according to angle with query
- Thought experiment: take a document $d$ and append it to itself, call this document $d'$
- “Semantically” $d$ and $d'$ have the same content.
- The angle between the two documents is 0, corresponding to maximal similarity
- The Euclidean distance between the two documents can be quite large
- Thus measuring the angle $\theta$ between the query vector and a document vector is much better
Illustration

$\vec{v}(d_1)$

$\vec{v}(d_2)$

$\vec{v}(d_3)$

$\vec{v}(q)$

$\theta$

gossip

jealous
From Angles to Cosines

- Because all vector components are greater equal to 0, all vectors are in the same quadrant
  - We only have angles between $0^\circ$ and $90^\circ$
- The larger the angle $\theta$, the smaller the cosine of $\theta$
- The smaller the angle $\theta$, the larger the cosine of $\theta$
From Angles to Cosines (2)

The following two notions are equivalent.

- Rank documents according to the *angle* between query and document in increasing order
- Rank documents according to \( \text{cosine}(\text{query}, \text{document}) \) in decreasing order

Cosine is a monotonically decreasing function of the angle for the interval \([0^\circ, 90^\circ]\)

On top of that, the cosine of an angle can be computed more easily than the angle itself
The cosine between a vector $\vec{x}$ and a vector $\vec{y}$ is computed as follows:

$$\cos \theta = \frac{\vec{x} \circ \vec{y}}{|\vec{x}| \cdot |\vec{y}|}$$

where $\circ$ is the dot product (or inner product) of the vectors:

$$\vec{x} \circ \vec{y} = \sum_{i=1}^{k} x_i \cdot y_i$$

and

$$|\vec{x}| = \sqrt{\sum_{i=1}^{k} x_i^2}$$
Computing the Cosine (2)

- So the matching-score of a document \( d_j \) with regard to a query \( q \) is

\[
\frac{\vec{q} \cdot \vec{d}_j}{|\vec{q}| \cdot |\vec{d}_j|}
\]

- \( |\vec{x}| \) is the length of a vector

- The length is used for normalization purposes (every matching-score is between 0 and 1)

- Vectors \( \vec{q} \) and \( \vec{d}_j \) are made up of tf-idf weights
Algorithm

\textbf{CosineScore}(q)
1. \texttt{float Scores}[N] = 0
2. \texttt{Initialize Length}[N]
3. \texttt{for each} query term $t$
4. \hspace{1em} \texttt{do calculate} $w_{t,q}$ \texttt{and fetch postings list for} $t$
5. \hspace{2em} \texttt{for each} pair $(d, tf_{t,d})$ \texttt{in postings list}
6. \hspace{3em} \texttt{do} $\text{Scores}[d] += w_{t,d} \times w_{t,q}$
7. \texttt{Read the array Length}[d]
8. \texttt{for each} $d$
9. \hspace{1em} \texttt{do} $\text{Scores}[d] = \text{Scores}[d] / \text{Length}[d]$
10. \texttt{return} Top K components of $\text{Scores}$

- The array $\text{Length}$ contains the lengths of each document (used for normalization)
- We don’t divide by the query length (as this is just a constant factor)
Variants

There are variants for tf-idf factors: a ranking is called a tf-idf ranking, when importance of a document

- increases with the number of occurrences within a document
- decreases with the number of occurrences of the term in the collection
Examples

<table>
<thead>
<tr>
<th>Term frequency</th>
<th>Document frequency</th>
<th>Normalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>n (natural)</td>
<td>tf(_{t,d})</td>
<td>n (no) 1</td>
</tr>
<tr>
<td>l (logarithm)</td>
<td>1 + log(tf(_{t,d}))</td>
<td>t (idf) ( \log \frac{N}{df_t} )</td>
</tr>
<tr>
<td>a (augmented)</td>
<td>0.5 + ( \frac{0.5 \times tf_{t,d}}{\max_i(tf_{i,d})} )</td>
<td>p (prob idf) ( \max{0, \log \frac{N-df_t}{df_i}} )</td>
</tr>
<tr>
<td>b (boolean)</td>
<td>\begin{cases} 1 &amp; \text{if } tf_{t,d} &gt; 0 \ 0 &amp; \text{otherwise} \end{cases}</td>
<td>u (pivoted unique) 1/(u) (Section 6.4.4)</td>
</tr>
<tr>
<td>L (log ave)</td>
<td>( \frac{1+\log(tf_{t,d})}{1+\log(\text{ave}<em>{i\in D}(tf</em>{i,d}))} )</td>
<td>b (byte size) 1/(\text{CharLength}^\alpha), (\alpha &lt; 1)</td>
</tr>
</tbody>
</table>

- Most popular one is logarithmic one
- According to Zobel and Moffat, there is no big difference in terms of quality for most tf-idf heuristics
Variants (2)

- We often use *different weightings* for queries and documents.
- Notation: qqq.ddd
- Example: ltn.lnc
  - query: logarithmic tf, idf, no normalization
  - document: logarithmic tf, no df weighting, cosine normalization
- bnn.ltc can be computed quite efficiently
  - Only multiplication with 0 or 1 in line 6 of the algorithm
Summary

- Represent the query as a weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Compute the cosine similarity between the query vector and each document vector
- Rank documents with respect to the query
- Return the top $K$ (e.g., $K = 20$) to the user