Text Technologies for Data Science
INFR11145

Ranked IR

Instructor:
Walid Magdy

10-Oct-2018

Lecture Objectives

• **Learn** about Ranked IR
  • TFIDF
  • VSM
  • SMART notation

• **Implement**:
  • TFIDF
**Boolean Retrieval**

- Thus far, our queries have all been Boolean.
  - Documents either: “match” or “no match”.

- Good for **expert users** with precise understanding of their needs and the collection.
  - Patent search uses sophisticated sets of Boolean queries and check hundreds of search results
    (car OR vehicle) AND (motor OR engine) AND NOT (cooler)

- Not good for the majority of users.
  - Most incapable of writing Boolean queries.
  - Most don’t want to go through 1000s of results.
    - This is particularly true of web search
    - Question: What is the most unused web-search feature?

---

**Ranked Retrieval**

- Typical queries: free text queries

- Results are “ranked” with respect to a query

- Large result sets are not an issue
  - We just show the top k (≈ 10) results
  - We don't overwhelm the user

- Criteria:
  - Top ranked documents are the most likely to satisfy user’s query
  - Score is based on how well documents match a query

\[
\text{Score}(d, q)
\]
Old Example

- Find documents matching query \{ink wink\}
  1. Load inverted lists for each query word
  2. Merge two postings lists → Linear merge
- Apply function for matches
  - Boolean: exist / not exist = 0 or 1
  - Ranked: \( f(tf, df, length, \ldots) = 0 \rightarrow 1 \)

\[
\text{Matches}
\begin{align*}
1: f(0,1) \\
3: f(1,0) \\
4: f(1,0) \\
5: f(1,1)
\end{align*}
\]

\[
\begin{array}{c}
\text{ink} \\
\text{wink}
\end{array}
\]

Function example: Jaccard coefficient

- a commonly used measure of overlap of two sets \( A \) and \( B \)
  \[
  \text{jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}
  \]
- \( \text{jaccard}(A, A) = 1 \)
- \( \text{jaccard}(A, B) = 0, \text{ if } A \cap B = 0 \)
- Example:
  - \( D1 \cup D2 = \{\text{he, likes, to, wink, and, drink}\} \)
  - \( D1 \cap D2 = \{\text{he, likes, to, drink}\} \)
  - \( \text{jaccard}(D1, D2) = \frac{4}{6} = 0.6667 \)
**Jaccard coefficient: Issues**

- Does not consider **term frequency** (how many times a term occurs in a document)
- It treats all terms equally!
  - How about **rare terms** in a collection? more informative than frequent terms.
  - *He likes to drink*, should “to” == “drink”
- Needs more sophisticated way of **length normalization**
  - |D1| = 3, |D2| = 1000!
  - D1 → Q, D2 → D

**Should terms be treaded the same?**

- Collection of 5 documents (balls = terms)
- Query

![balls](image)
- Which is the least relevant document?
- Which is the most relevant document?
**TFIDF**

- **TFIDF**: Term Frequency, Inverse Document Frequency
  - **tf(t,d)**: number of times term \( t \) appeared in document \( d \)
    - As \( tf(t,d) \) \( \uparrow \) \( \rightarrow \) importance of \( t \) in \( d \) \( \uparrow \)
    - Document about IR, contains “retrieval” more than others
  - **df(t)**: number of documents term \( t \) appeared in
    - As \( df(t) \) \( \uparrow \) \( \rightarrow \) importance if \( t \) in a collection \( \downarrow \)
      - “the” appears in many document \( \rightarrow \) not important
      - “FT” is not important word in financial times articles

**DF, CF, & IDF**

- **DF ≠ CF** (collection frequency)
  - \( cf(t) = \) total number of occurrences of term \( t \) in a collection
  - \( df(t) \leq N \) (\( N \): number of documents in a collection)
  - \( cf(t) \) can be \( \geq N \)
- **DF** is more commonly used in IR than **CF**
  - **CF** is still used
- **idf(t)**: inverse of \( df(t) \)
  - As \( idf(t) \) \( \uparrow \) \( \rightarrow \) rare term \( \rightarrow \) importance \( \uparrow \)
  - \( idf(t) \) \( \rightarrow \) measure of the informativeness of \( t \)
DF vs CF

<table>
<thead>
<tr>
<th></th>
<th>he</th>
<th>drink</th>
<th>ink</th>
<th>likes</th>
<th>pink</th>
<th>think</th>
<th>wink</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D2</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

D1: He likes to wink, he likes to drink
D2: He likes to drink, and drink, and drink
D3: The thing he likes to drink is ink
D4: The ink he likes to drink is pink
D5: He likes to wink, and drink pink ink

IDF: formula

\[ idf(t) = \log_{10}(\frac{N}{df(t)}) \]

- Log scale used to dampen the effect of IDF
- 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>sky</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>
**TFIDF term weighting**

- One of the best known term weights schemes in IR
  - Increases with the number of occurrences within a document
  - Increases with the rarity of the term in the collection
- Combines TF and IDF to find the weight of terms:
  \[
  w_{t,d} = \left(1 + \log_{10} tf(t,d)\right) \times \log_{10} \left(\frac{N}{df(t)}\right)
  \]
- For a query \( q \) and document \( d \), retrieval score \( f(q,d) \):
  \[
  \text{Score}(q,d) = \sum_{t \in q \cap d} w_{t,d}
  \]

---

**Document/Term vectors with tfidf**

<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>5.25</td>
<td>3.18</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Brutus</td>
<td>1.21</td>
<td>6.1</td>
<td>0</td>
<td>1</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>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1.54</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>
</tr>
<tr>
<td>mercy</td>
<td>1.51</td>
<td>0</td>
<td>1.9</td>
<td>0.12</td>
<td>5.25</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>
</tr>
</tbody>
</table>

→ Vector Space Model
**Vector Space Model**

- Documents and Queries are presented as vectors.
- Match (Q,D) = Distance between vectors.
- Example: Q= Gossip Jealous
- Euclidean Distance? 
  
  *Distance between the endpoints of the two vectors*

- Large for vectors of different lengths.
- Take a document d and append it to itself. Call this document d′.
  - “Semantically” d and d′ have the same content
  - Euclidean distance can be quite large.

**Angle Instead of Distance**

- The angle between the two documents is 0, corresponding to maximal similarity.
- Key idea: Rank documents according to angle with query.
  - Rank documents in increasing order of the angle with query
  - Rank documents in decreasing order of cosine (query, document)
- Cosine of angle = projection of one vector on the other.
**Length Normalization**

- A vector can be normalized by dividing each of its components by its length – for this we use the L$_2$ norm:
  \[ \|\tilde{x}\|_2 = \sqrt{\sum_i x_i^2} \]
- Dividing a vector by its L$_2$ norm makes it a unit (length) vector (on surface of unit hypersphere)
- Effect on the two documents $d$ and $d'$ ($d$ appended to itself) from earlier slide: they have identical vectors after length-normalization.
  - Long and short documents now have comparable weights

**Example**

- $D_1 = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} \Rightarrow \|D_1\|_2 = \sqrt{1 + 9 + 4} = 3.74$
- $D_{1\text{ normalized}} = \begin{bmatrix} 0.267 \\ 0.802 \\ 0.535 \end{bmatrix}$
- $D_2 = \begin{bmatrix} 3 \\ 9 \\ 6 \end{bmatrix} \Rightarrow \|D_2\|_2 = \sqrt{9 + 81 + 36} = 11.25$
- $D_{2\text{ normalized}} = \begin{bmatrix} 0.267 \\ 0.802 \\ 0.535 \end{bmatrix}$
Cosine “Similarity” (Query, Document)

- $\tilde{q}_i$ is the tf-idf weight of term $i$ in the query
- $\tilde{d}_i$ is the tf-idf weight of term $i$ in the document
- For normalized vectors:
  \[
  \cos(\tilde{q}, \tilde{d}) = \frac{\tilde{q} \cdot \tilde{d}}{\|\tilde{q}\| \|\tilde{d}\|} = \frac{\sum_{i=1}^{V} q_i d_i}{\sqrt{\sum_{i=1}^{V} q_i^2 \sqrt{\sum_{i=1}^{V} d_i^2}}}
  \]
- For non-normalized vectors:
  \[
  \cos(\tilde{q}, \tilde{d}) = \frac{\tilde{q} \cdot \tilde{d}}{\|\tilde{q}\| \|\tilde{d}\|} = \frac{\sum_{i=1}^{V} q_i d_i}{\sqrt{\sum_{i=1}^{V} q_i^2} \sqrt{\sum_{i=1}^{V} d_i^2}}
  \]

Algorithm

\[
\text{COSINESCORE}(q)
\]

1. \(\text{float Scores}[N] = 0\)
2. \(\text{float Length}[N]\)
3. \(\text{for each query term } t\)
4. \(\text{do calculate } w_{t,q} \text{ and fetch postings list for } t\)
5. \(\text{for each pair}(d, tf_{t,d}) \text{ in postings list}\)
6. \(\text{do } Scores[d] + = w_{t,d} \times w_{t,q}\)
7. \(\text{Read the array Length}\)
8. \(\text{for each } d\)
9. \(\text{do } Scores[d] = Scores[d] / Length[d]\)
10. \(\text{return Top } K \text{ components of } Scores[]\)
TFIDF 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>l (logarithm)</td>
<td>1 + log(tf, d)</td>
<td>1</td>
</tr>
<tr>
<td>a (augmented)</td>
<td>0.5 + 0.5*tf, d / max(tf, d)</td>
<td>p (prob idf)</td>
</tr>
<tr>
<td>b (boolean)</td>
<td>1 if tf, d &gt; 0, 0 otherwise</td>
<td>max(0, log N / df, d)</td>
</tr>
<tr>
<td>l (log ave)</td>
<td>1 + log(tf, d) / log(avg(tf, d))</td>
<td>1 / b (pivoted unique)</td>
</tr>
</tbody>
</table>

- Many search engines allow for different weightings for queries vs. documents
- **SMART** Notation: use notation `ddd.qqq`, using the acronyms from the table
- A very standard weighting scheme is: `lncltc`

Summary of Steps:

- 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
Retrieval Output

• For a query $q_1$, the output would be a list of documents ranked according to the $score(q_1,d)$

• Common output format:

  1 0 710 0 0.9234 0
  1 0 213 0 0.7678 0
  1 0 103 0 0.6761 0
  1 0 13 0 0.6556 0
  1 0 501 0 0.4301 0

Query id  document id  score

Resources

• Text book 1: Intro to IR, Chapter 6.2 → 6.4
• Text book 2: IR in Practice, Chapter 7

• Lab 3