Querying
Introduction to Information Retrieval
INF 141/ CS 121
Donald J. Patterson

Content adapted from Hinrich Schütze
http://www.informationretrieval.org
Ranked Search

- Rather than saying
  - (query, document) matches or not (0,1)
    - ("Capulet","Romeo and Juliet") = 1
- Now we are going to assign rankings
  - (query, document) in {0,1}
    - ("capulet","Romeo and Juliet") = 0.7
• **Metadata** = structured additional information about a document.

• Examples:
  • The author of a document
  • The creation date of a document
  • The title of a document
  • The location where a document was created
  • author, creation date, title, location are fields

• searching for “William Shakespeare” in a doc differs from

• searching for “William Shakespeare” in the author of a doc
Querying

• **Parametric Search**
  
  • supports searching on meta-data explicitly
  
  • a parametric search interface allows a mix of full-text query and meta-data queries

• Example:
  
  • [www.carfinder.com](http://www.carfinder.com)
Querying

- **Parametric Search**
  - Example:
    - Result is a large table
    - Columns are fields
    - Searching for “2013” only applied to year field
  - [www.carfinder.com](http://www.carfinder.com)
Querying

- **Parametric Search**

- **Example:**
Querying

- Parametric Search
- Example:
  - http://www.ocregister.com/realestate/
  - 92614: 218 results
Querying

- **Parametric Search**

- **Example:**
  - http://www.ocregister.com/realestate/
  - 92614: 218 results
In these examples we select field values

- Values could be hierarchical
  - USA -> California -> Orange County -> Newport Beach

- It is a paradigm for navigating through a corpus
  - e.g., “Aerospace companies in Brazil” can be found by combining “Geography” and “Industry”

Approach:
- Filter for relevant documents
- Run text searches on subset
Index support for parametric search

Must be able to support queries of the form:
- Find pdf documents that contain “UCI”
- Field selection and text query

Field selection approach

- Use inverted index of field values
  - (field value, docID)
  - organized by field name
- Using same compression and sorting techniques
Building up our query technology

- “Matching” search
- Linear on-demand retrieval (aka grep)
- 0/1 Vector-Based Boolean Queries
- Posting-Based Boolean Queries
- Ranked search
- Parametric Search
- Zones
A zone is an extension of a field.
A zone is an identified region of a document:
- e.g., title, abstract, bibliography
- Generally identified by mark-up in a document:
  - `<title>Romeo and Juliet</title>`

Contents of zone are free text:
- Not a finite vocabulary

Indices required for each zone to enable queries like:
- (instant in TITLE) AND (oatmeal in BODY)

Doesn’t cover “all papers whose authors cite themselves”
- Why?
Now, we crawl the corpus
We parse the document keeping track of terms, fields and docIDs
Instead of building just a (term, docID) pair
We build (term, field, docID) triples
These can then be combined into postings like this:

<table>
<thead>
<tr>
<th>Term</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>William.author</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>William.title</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>13</td>
</tr>
<tr>
<td>William.abstract</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>11</td>
</tr>
</tbody>
</table>
So are we just creating a database?

Not really.

Databases have more functionality

- Transactions
- Recovery
  - Our index can be recreated. Not so with database.
  - Text is never stored outside of indices

- We are focusing on optimized indices for text-oriented queries not a full SQL engine
Building up our query technology

- “Matching” search
  - Linear on-demand retrieval (aka grep)
  - 0/1 Vector-Based Boolean Queries
  - Posting-Based Boolean Queries
- Ranked search
  - Parametric Search
  - Zones
- Scoring
Boolean queries “match” or “don’t match”

Good for experts with needs for precision and coverage
- knowledge of corpus
- need 1000’s of results

Not good with non-expert users
- who don’t understand boolean operators
- or how they apply to search
- or who don’t want 1000’s of results
• Boolean queries require careful crafting to get the right number of results (Ferrari example)
• Ranked lists eliminate this concern
  • Doesn’t matter how big the list is
• **Scoring** is the basis for ranking or sorting document that are returned from a query.
  • Ideally the score is high when the document is **relevant**
  • **WLOG** we will assume scores are between 0 and 1 for each doc.
Scoring

- First generation of scoring used a linear combination of Booleans

\[ \text{Score} = 0.6(oatmeal \in TITLE) + 0.3(oatmeal \in BODY) + 0.1(oatmeal \in ABSTRACT) \]

- Explicit decision about importance of zone
- Each subquery is 0 or 1
- This example has a finite number of possible values
- What are they?
Scoring

\[ \text{Score} = 0.6(oatmeal \in \text{TITLE}) + 0.3(oatmeal \in \text{BODY}) + 0.1(oatmeal \in \text{ABSTRACT}) \]

- Subqueries could be *any* Boolean query
- Where do we get the weights? (e.g., 0.6, 0.3, 0.1)
  - Rarely from the user
  - Usually built into the query engine
  - Where does the query engine get them from?
    - Machine learning
Scoring Exercise

- Calculate the score for each document based on the weightings (0.1 author), (0.3 body), (0.6 title)
- For the query “bill” or “rights”
Building up our query technology

- “Matching” search
  - Linear on-demand retrieval (aka grep)
  - 0/1 Vector-Based Boolean Queries
  - Posting-Based Boolean Queries
- Ranked search
  - Parametric Search
  - Zones
  - Scoring
Zones combination index

- Encode the zone in the posting
- At query time accumulate the contributions to the total score from the various postings
Zone scoring with zones combination index

“bill OR rights” (0.1 author), (0.3 body), (0.6 title)
Zone scoring with zones combination index

“bill OR rights” (0.1 author), (0.3 body), (0.6 title)

1: 0.4
Zone scoring with zones combination index

“bill OR rights” (0.1 author), (0.3 body), (0.6 title)

1: 0.4
2: 0.4
Zone scoring with zones combination index

“bill OR rights” (0.1 author), (0.3 body), (0.6 title)

1: 0.4
2: 0.4
3: 0.9
Zone scoring with zones combination index

“bill OR rights” (0.1 author), (0.3 body), (0.6 title)

1: 0.4  
2: 0.4  
3: 0.9  
5: 0.9
Zone scoring with zones combination index

“bill OR rights” (0.1 author), (0.3 body), (0.6 title)

bill      1.author  1.body  2.author  2.body  3.title  5.body  5.title  8.title  9.body

1: 0.4  5: 0.9
2: 0.4  8: 0.9
3: 0.9
Zone scoring with zones combination index

“bill OR rights” (0.1 author), (0.3 body), (0.6 title)

1: 0.4  5: 0.9
2: 0.4  8: 0.9
3: 0.9  9: 0.9
Zone scoring with zones combination index

“bill OR rights” (0.1 author), (0.3 body), (0.6 title)

Results: 9, 8, 5, 3, 2, 1
Zone scoring with zones combination index

• As we walk, we accumulate scores linearly
• Note: getting “bill” and “rights” in the title field didn’t cause us to score any higher
  • Should it?
• Where do the weights come from?
  • Machine learning
    • Given a corpus, test queries and “gold standard” relevance scores, compute weights which come as close as possible to “gold standard”
Full text queries

- Previous example was for “bill OR rights”
- Average user is likely to type “bill rights” or “bill of rights”
  - How do we interpret such a query?
  - No Boolean operators
  - Some query terms might not be in the document
  - Some query terms might not be in a zone
Full text queries

- To use zone combinations for free text queries, we need:
  - A way of scoring = Score(full-text-query, zone)
  - Zero query terms in zone -> zero score
  - More query terms in a zone -> higher score
  - Scores don’t have to be boolean (0 or 1) anymore

- Let’s look at the alternatives...
Building up our query technology

- “Matching” search
  - Linear on-demand retrieval (aka grep)
  - 0/1 Vector-Based Boolean Queries
  - Posting-Based Boolean Queries
- Ranked search
  - Parametric Search
  - Zones
  - Scoring
- Term Frequency Matrices
Incidence Matrices

- Recall how a document, d, (or a zone) is a (0,1) column vector.
- A query, q, is also a column vector. How so?

<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>
</tr>
</thead>
<tbody>
<tr>
<td>Anthony</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>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Incidence Matrices

- Using this formalism, score can be overlap measure:

$$|q \cap D|$$
Incidence Matrices

- Example:
  - Query “ides of march”
  - Shakespeare’s “Julius Caesar” has a score of 3
  - Plays that contain “march” and “of” score 2
  - Plays that contain “of” score 1

- Algorithm:
  - Bitwise-And between q and matrix, D
  - Column summation
  - Sort
Incidence Matrices

- What is wrong with the overlap measure?
- It doesn’t consider:
  - Term frequency in a document
  - Term scarcity in corpus
    - “ides” is much rarer than “of”
  - Length of a document
  - Length of queries
Toward better scoring

- Overlap Measure
- Normalizing queries
  - **Jaccard Coefficient**
    - Score is number of words that overlap divided by total number of words
    - What documents would score best?
- **Cosine Measure**
  - Will the same documents score well?
Toward Better Scoring

- Scores so far capture position (zone) and overlap
- Next step: a document which talks about a topic should be a better match
  - Even when there is a single term in the query
  - Document is relevant if the term occurs a lot
- This brings us to term weighting
Querying

Bag of Words Model

- “Don fears the mole man” equals “The mole man fears Don”
- The incidence matrix for both looks the same
Querying

Term Frequency Matrix

- Bag of words
- Document is vector with integer elements

<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>157</td>
<td>73</td>
<td>0</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>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>232</td>
<td>227</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>10</td>
<td>0</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>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>worse</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Querying

Term Frequency - tf

- Long documents are favored because they are more likely to contain query terms
- Reduce the impact by normalizing by document length
- Is raw term frequency the right number?
Querying

Weighting Term Frequency - WTF

- What is the relative importance of
  - 0 vs. 1 occurrence of a word in a document?
  - 1 vs. 2 occurrences of a word in a document?
  - 2 vs. 100 occurrences of a word in a document?

- Answer is unclear:
  - More is better, but not proportionally

- An alternative to raw tf:

$$\text{WTF}(t, d) = \begin{cases} 
1 & \text{if } tf_{t,d} = 0 \\
2 & \text{then } \text{return}(0) \\
3 & \text{else } \text{return}(1 + \log(tf_{t,d})) 
\end{cases}$$