Information Retrieval

Prof: Ehab Ezzat Hassanein

Introducing Ranked Retrieval & Scoring with Jaccard Coefficient Introduction to term Frequency

Ranked Retrieval

- Until now, our queries have all been Boolean
  - Documents either match or don’t
- Good expert user's with precise understanding of their needs and the collection.
  - Also good for applications: Applications easily consume 1000’s of results.
- Not good for majority of users.
  - Most users incapable of writing Boolean queries (or they are, but they think it is too much work).
  - Most users don’t want to wade through 1000’s of results.
    - This is particularly true of web search

Problem with Boolean search: feast or famine

- Boolean queries often result in either too few (≈0) 0r to many (1000’s) results0.
  - Query 1: “stanford user dlink 650” → 200,000 hits
  - Query 2: “stanford user dlink 650 no card found” → 0 hits
- It takes a lot of skill to come up with a query that produce a manageable number of hits.
  - AND gives too few; OR gives too many.

Ranked retrieval model

- Rather than a set of documents satisfying a query expression, in ranked retrieval models, the system returns an ordering over the(top) document in the collection with respect to a query.
- Free text queries: Rather than a query language of operator and expressions, the user’s query is just one or more words in a human language.
- In principle, there are two separate choices here, but in practice, ranked retrieval mosels have normally been associated with free text queries and vice versa.

Feast or famine: not a problem in ranked retrieval

- When a user produces aranked result set, large result sets ar not an issue
  - Indeed , the size of the result set is not an issue
  - We just show the to ***k*** (≈10) results
  - We don’t overwhelm the user

  - Premise: the ranking algorithm works

Scoring as the basis of ranked retrieval

- We wish to return in order the document 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”.

Query-document matching scores

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

The Jaccard coefficient

- A commonly used measure of overlap of two sets **A** and **B** is the Jaccard coefficient
- $jaccard(A,B) = |A \cap B| / |A \cup B|$
- $jaccard(A,A) = 1$
- $jaccard(A,B) = 0$ if $A \cap B = 0$
- **A** and **B** don’t have to be the same size.
- Always assigns a number between 0 and 1.

Jaccard Coefficient: Scoring Example

- What is the query-document match score that the Jaccard coefficient computes for each of the two documents below?
  - Query: idea of march
  - Document 1: ceaser died in march
  - Document 2: the long march

J(q, d1) = 1/6 = 0.1667

J(q, d2) = 1/5 = 0.2

d2 wins!

Issues with Jaccard for scoring

- It does not consider term frequency (how many times a term occurs in a document)
  - Rare terms in a collection are more information than frequent terms
  - Jaccard doesn’t consider this information
- We need or sophisticate way of normalizing length
  - Later we will use $|A \cap B| / \sqrt{|A \cup B|}$

    instead of $|A \cap B| / |A \cup B|$

Recall: Binary term-document incidence matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

- Each document is rpresented by a binay victor member of {0,1}

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Bag of words model

- Vector representation does not consider the ordring of words.
- John is quicker than Mary and Mary is quicker than John have the same vectors
- This is called the bag of words model
- In a sense it is a step back:
  - The positional indx was able to distinguish these two documents
    - W will look at recovering positional information later
    - But fornow: bag of words model

Term frequency tf

- The term frequency $tf_{t,d}$ of term t in th 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 trm frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 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

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

- 0→0, 1→1, 2→1.3, 10→2 1000→ 4   etc.

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

- Note the score is 0 if none of the query terms is present in the document