THE UNIVERSITY of EDINBURGH

Text Technologies for Data Science

INFR11145

Ranked IR

Instructor:
Walid Magdy

10-Oct-2017

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 <u>expert users</u> 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?

Walid Magdy, TTDS 2017/2018

THE UNIVERSITY of EDINBURGH

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
    Score(*d*,*q*)

Walid Magdy, TTDS 2017/2018

THE UNIVERSITY of EDINBURGH

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*, ….) = 0 → 1

ink ⇒ | 3:1 | 4:1 | 5:1 |

wink ⇒ | 1:1 | 5:1 |

Matches

1: f(0,1)

3: f(1,0)

4: f(1,0)

5: f(1,1)

Function example: Jaccard coeffecient

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

D1: He likes to wink, he likes to drink

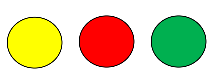
D2: He likes to drink, and drink, and drink

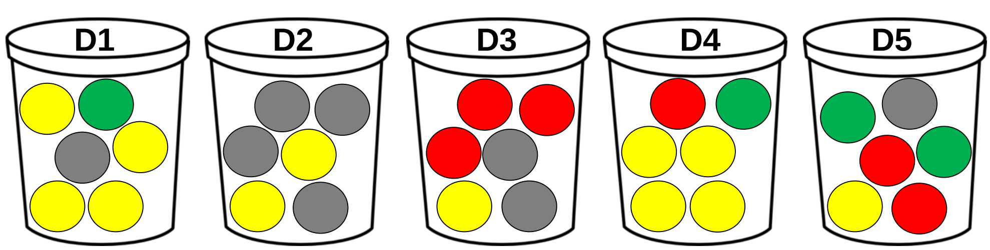
Jaccard coefficient: Issues

- 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

THE UNIVERSITY *of* EDINBURGH

Should are terms be treaded the same?

- Collection of 5 documents (balls = terms)
- Query 
- Which is the least relevant document?
- Which is the most relevant document?



THE UNIVERSITY *of* EDINBURGH

TFIDF

- **TFIDF**:
  Term **F**requency, **I**nverse **D**ocument **F**requency
- ***tf(t,d)***:
  number of times term ***t*** appeared in document ***d***
  - As $tf(t,d)$ ↑↑ → importance of *t* in *d* ↑↑
  - Document about IR, contains "retrieval" more than others
- ***df(t)***:
  number of documents term ***t*** appeared in
  - As $\mathrm{d}f(d)$ ↑↑ → importance if *t* in a collection ↓↓
    - "the" appears in many document → 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)*** ≤ ***N*** (***N***: number of documents in a collection)
  - ***cf(t)*** can be ≥ ***N***
- **DF** is more commonly used in IR than **CF**
- ***idf(t)***: inverse of ***df(t)***
  - As $\mathrm{i}df(t)$ ↑↑ → rare term → importance ↑↑
  - ***idf(t)*** → measure of the informativeness of *t*

IDF: formula

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

- Log scale used to dampen the effect of IDF
- Suppose N = 1 million →

term	df(t)	idf(t)
calpurnia	1	6
animal	100	4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

TFIDF term weighting

- One 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

$$w_{t.d} = \left(1 + log_{10}tf(t,d)\right) \times log_{10}(\frac{N}{df(t)})$$

- For a query *q* and document *d*, retrieval score *f(q,d)*:

$$Score(q,d) = \sum_{t \in q \cap d} w_{t.d}$$

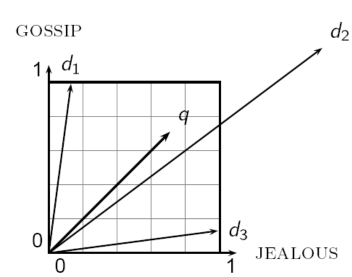
Document/Term vectors with tfidf

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

→ **Vector Space Model**

Vector Space Model

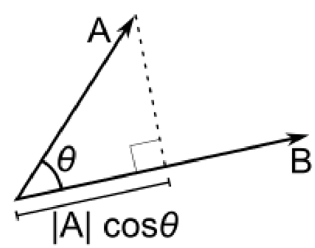
- Documents and Queries are presented as vectors
- Match (Q,D) = Distance between vectors
- Example: Q= Gossip Jealous
- Euclidean Distance?
  Distance between the endpoints of the two vectors
- Large for vectors of diff. lengths



- 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:

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

- 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

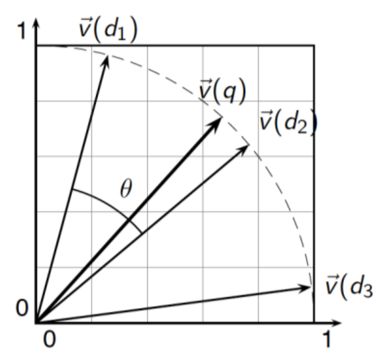
Example

- D1 = $\begin{bmatrix}1\\3\\2\end{bmatrix}$ ➔ $\left\|\overrightarrow{D1}\right\|_2 = \sqrt{1+9+4} = 3.74$
- D1$_{normalized}$ = $\begin{bmatrix}0.267\\0.802\\0.535\end{bmatrix}$
- D2 = $\begin{bmatrix}3\\9\\6\end{bmatrix}$ ➔ $\left\|\overrightarrow{D1}\right\|_2 = \sqrt{9+81+36} = 11.25$
- D2$_{normalized}$ = $\begin{bmatrix}0.267\\0.802\\0.535\end{bmatrix}$

Cosine "Similarity" (Query, Document)

- $\vec{q}_i$ is the tf-idf weight of term i in the query
- $\vec{d}_i$ is the tf-idf weight of term i in the document
- For normalized vectors:

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



- For non-normalized vectors:

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

Algorithm

COSINESCORE(q)

```
float Scores[N] = 0
float Length[N]
for each query term t
do calculate w_{t,q} and fetch postings list for t
    for each pair(d, tf_{t,d}) in postings list
    do Scores[d] += w_{t,d} × w_{t,q}
Read the array Length
for each d
do Scores[d] = Scores[d]/Length[d]
return Top K components of Scores[]
```


THE UNIVERSITY of EDINBURGH

TFIDF Variants

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5\times\mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/\mathit{CharLength}^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

- 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: *lnc.ltc*

THE UNIVERSITY of EDINBURGH

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

Resources

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

- Ranked IR (2) → BM25:
  *Robertson, Stephen E., et al. "**Okapi at TREC-3**." Nist Special Publication Sp 109 (1995): 109.*