Information Retrieval (CSD510)

Vector Space Model

February 1, 2024

Lecture outline

1. *tf* − *idf* scoring

2. Vector space model

$tf - idf$ scoring

Ranked retrieval

- For *Boolean queries*, documents either match or don't match.
  - Not possible to judge the degree of relevance of a document with respect to a query.
  - Good for expert users with precise understanding of their needs and the collection
  - Not good for the majority of users as they are incapable of writing Boolean queries
- Thus, *Boolean retrieval* not suitable for web search
- Boolean queries often result in either too few (=0) or too many (1000s) results.
  - "standard user dlink 650" → 200,000 hits
  - "standard user dlink 650 no card found" → 0 hits
  - AND gives too few; OR gives too many results

Ranked retrieval models

- Boolean retrieval: Returns set of documents satisfying a query expression
- **Ranked retrieval:** The system returns an ordering over the (top) documents in the collection for a query
- When a system produces a ranked result set
  - the size of the result set is not an issue
  - The top k most relevant (highest ranking) documents can be returned
  - Don't overwhelm the user

Ranked retrieval models

- Return the documents in *an order* 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** to each document which estimates how well the document "**matches**" the query
    - The score may be in the range [0, 1]

Query-document matching scores

- We need a way of assigning a score to a query/document pair
- For a given one-term query
  - IF (query term not in document): score $= 0$
  - The more frequent the query term in the document, the higher the score
- The rest of the discussion is based on this idea and we explore several alternatives and extensions

Query-document matching scores

- **Term frequency (*tf*)**: The number of times a term occurs in the document.
  - Rare terms in a collection are more informative than frequent terms.
  - The documents themselves may vary in length
  - We need a more sophisticated way of normalizing the length of document

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 represented by a binary vector $\in 0, 1^{|V|}$

Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector

	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

- Considers a document or a query as a multi-set set of terms
- Does not take the order of the words in the document into account
- In the **vector space representation** also the ordering is not maintained
- *John runs faster than Mary* and *Mary runs faster than John* have the same vector representation
- Step back from positional indexing

	John	runs	faster	than	Mary
John runs faster than Mary	1	1	1	1	1
Mary runs faster than John	1	1	1	1	1

Term frequency - tf

- The term frequency $tf_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.
- We want to use t when computing query-document match scores.
- Using raw term frequency has some disadvantages
  - Relevance is not directly proportional to the term frequency
  - A document with 10 occurrences of a term may be *more relevant* than a document containing the term once but not 10 times more relevant

Term frequency - tf

- A *normalization scheme* is the log frequency weight of term **t in d** is
$$w_{t,d} = \begin{cases} 1 + log_{10} tf_{t,d} & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$
- $tf_{td} \rightarrow w_{td}$:
$0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4$
- Score for a document-query pair: sum over terms t in both q and d:
$tf\_matching\_score(q, d) = \sum_{t \in q \cap d} (1 + logtf_{t,d})$
- The score is 0 if none of the query terms is present in the document.

Document Frequency

- **Document frequency (df_t):** Number of documents in the collection in which term t occurs.
- Use the frequency of the term in the collection for weighting and ranking.
- Rare terms are more informative than frequent terms
- 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 high weights for rare terms like **arachnocentric**

Document Frequency

- Frequent terms are less informative than rare terms.
- Consider a term in the query that is frequent in the collection (e.g., good, increase, line).
- A document containing this term is more likely to be relevant than a document that doesn't.
  - These frequent terms are not sure indicators of relevance.
  - For these frequent terms we want positive weights, but lower weights than the rare terms
- Need high weights for rare terms
- We can use the document frequency to factor this phenomenon into computing the matching score.

Inverse document frequency (*idf*) score

- df_t is an inverse measure of the informativeness of term t
- **Inverse document frequency:** Is a measure of the informativeness of term t.
- We define the **idf** weight of term t as follows

  $idf_t = log(N/df_t)$

  (N is the number of documents in the collection.)
- idf_t is a measure of the informativeness of the term.

Effect of idf on ranking

- Does idf have an effect on ranking for one-term queries e.g. iPhone
- *idf* has no effect on ranking one term queries
  - *idf* used to measure the relative importance of terms
  - *idf* affects the ranking of documents for queries with at least two terms
  - For the query capricious person, *idf* weighting makes occurrences of capricious count for much more in the final document ranking than occurrences of person

Collection vs. Document frequency

- The collection frequency of t is the number of occurrences of t in the collection, counted multiple occurrences

Word	**Collection**	**Frequency**
insurance	10440	3997
try	10422	8760

tf − idf weighting

- The *tf − idf* weight of a term is the product of its *tf* weight and its *idf* weight.

$$w_{t,d} = (1 + log\ tf_{t,d}).log\frac{N}{df_t}$$

- Best known weighting scheme in information retrieval
- Increases with the
  - number of occurrences within a document (term frequency)
  - rarity of the term in the collection (inverse document frequency)

Vector space model

$tf - idf$ score matrix

	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

Each document is now represented by a real-valued vector of $tf - idf$ weights $\in R^{|V|}$

Documents as Vectors

- Each document is now represented by a real-valued vector of tf-idf weights $\in \mathcal{R}^{|V|}$
- It is 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 web search engines
- Each vector is very sparse - 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 in this space
- proximity $=$ similarity of vectors
- proximity $\approx$ inverse of distance

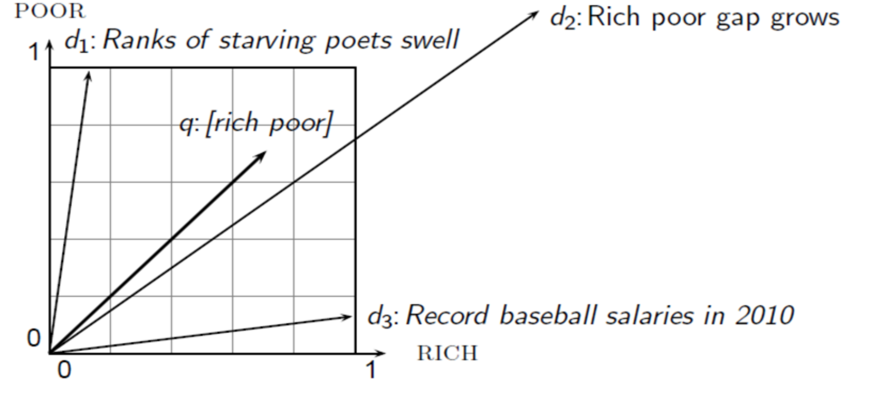
Queries as vectors

- 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 in this space
- proximity = similarity of vectors
- proximity $\approx$ inverse of distance
- How to quantify the similarity between the vectors?

Similarity using distance (difference)

The Euclidean distance between q and d_2 is large even though the distribution of terms in the query q and the distribution of terms in the document d_2 are very similar

Vector representation



Use angle instead of distance

- Take a document d and append it to itself. Call this document $\hat{d}$.
- Semantically d and $\hat{d}$ have the same distance
- The Euclidean distance between the two documents can be quite large
- The angle between the two documents is 0, corresponding to maximal similarity

Use angle instead of distance

- Take a document d and append it to itself. Call this document $\hat{d}$.
- Semantically d and $\hat{d}$ have the same distance
- The Euclidean distance between the two documents can be quite large
- The angle between the two documents is 0, corresponding to maximal similarity
- **Key idea: Rank documents according to angle with query**

From angles to cosines

- The following two notions are equivalent
  - Rank documents in **decreasing order** of the angle between query and document
  - Rank documents in **increasing order** of cosine of the angle between query and document
- Cosine is a monotonically decreasing function for the interval [0°, 180°]
- Advantages of cosine similarity
  - Cosine score is proportional to similarity
  - Scales down the similarity score in the range [0,1]

Cosine(query, document)

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

Dot product: $\vec{q} \bullet \vec{d}$

q_i is the tf-idf weight of term i in the query

d_i is the tf-idf weight of term i in the document

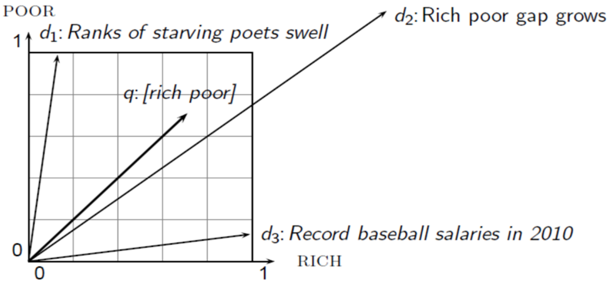
- $cos(\overrightarrow{q}, \overrightarrow{d})$ is the cosine of the angle between $\overrightarrow{q}$ and $\overrightarrow{d}$.
- $cos(\overrightarrow{q}, \overrightarrow{d})$ is the **cosine similarity** of $\overrightarrow{q}$ and $\overrightarrow{d}$
- If $\overrightarrow{q}$ and $\overrightarrow{d}$ are length normalized, then cosine similarity is the scalar (dot) product.

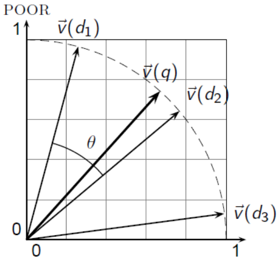
$$cos(\overrightarrow{q}, \overrightarrow{d}) = \overrightarrow{q} \cdot \overrightarrow{d} = \sum_{i=1}^{|V|} q_i d_i \quad (1)$$

Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length.
- For this we use the L_2 norm
$$\|x\|_2 = \sqrt{\sum_i x_i^2}$$
- Dividing a vector by its L_2 norm makes it a unit (length) vector
- Effect on the two documents d and $\hat{d}$ (d appended to itself) from earlier slide: they have identical vectors after length-normalization
  - Long and short documents now have comparable weights

Cosine similarity





Cosine similarity among 3 documents

How similar are the novels?

SaS: Sense and Sensibility , PaP: Pride and Prejudice, and WH: Wuthering Heights

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Table: Term frequencies (counts)

- Note: To simplify this example, we don't do idf weighting

Cosine similarity among 3 documents

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

Table: Log frequency weighting

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

Table: After length normalization

cos(SaS,PaP) $\approx$

$0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0$

≈ 0.94

cos(SaS,WH) ≈ 0.79

cos(PaP,WH) ≈ 0.69

Computing cosine scores

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$(d, \mathrm{tf}_{t,d})$ in postings list
    do $Scores[d] += w_{t,d} \times w_{t,q}$
Read the array *Length*
for each d
do $Scores[d] = Scores[d]/Length[d]$
return Top K components of $Scores[]$