Information Retrieval and Web Search

Cornelia Caragea

Computer Science
University of Illinois at Chicago

Credits for slides: Mooney

Retrieval Models

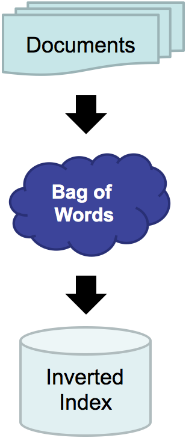
Vector Space Model: Implementation Steps

- Step 1: Preprocessing
- Step 2: Indexing
- Step 3: Retrieval
- Step 4: Ranking

Step 1: Preprocessing

- Implement the preprocessing functions:
  - For tokenization
  - For stop word removal
  - For stemming
- **Input:** Documents that are read one by one from the collection
- **Output:** Tokens to be added to the index
  - No punctuation, no stop-words, stemmed

Step 1: Preprocessing (II)



case folding, tokenization, stopword removal, stemming

~~semantics~~, ~~syntax~~, etc.

Text as Sparse Vectors

- Vocabulary and therefore dimensionality of vectors can be very large, $\approx 10^5$.
- However, most documents and queries do not contain most words, so vectors are sparse (i.e. most entries are 0).
- Need efficient methods for storing and computing with sparse vectors.

Step 2: Indexing

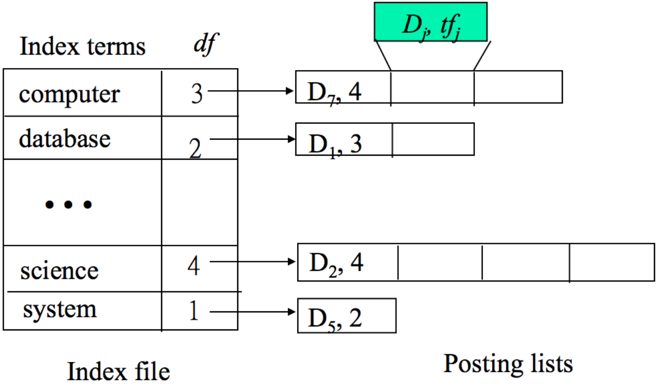
- Build an inverted index, with an entry for each word in the vocabulary
- **Input:** Tokens obtained from the preprocessing module
- **Output:** An inverted index for fast access

- Many data structures are appropriate for fast access
  - We will use hashtables
    - Store tokens in hashtable, with token string as key and weight as value.
    - Table must fit in main memory.

Step 2: Indexing (II)

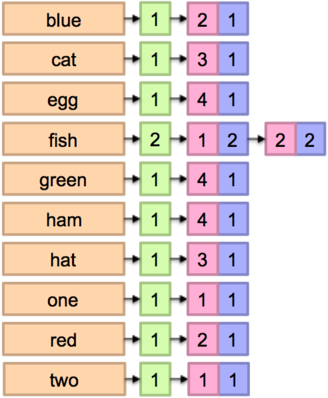
- We need:
  - One entry for each word in the vocabulary
  - For each such entry:
    - Keep a list of all the documents where it appears together with the corresponding frequency $\rightarrow$ TF
    - Keep the total number of documents in which the corresponding word appears $\rightarrow$ IDF
- Constant time to find or update weight of a specific token.

Inverted Index



Inverted Index: TF-IDF

Doc1
one fish, two fish
Doc2
red fish, blue fish
Doc3
cat in the hat
Doc4
green eggs and ham



Indexing - How many passes through the data?

- TF and IDF for each token can be computed in one pass
- Cosine similarity also requires document lengths
- Need a second pass to compute document vector lengths
  - Remember that the length of a document vector is the square-root of sum of the squares of the weights of its tokens.
  - Remember the weight of a token is: TF * IDF
  - Therefore, must wait until IDF's are known (and therefore until all documents are indexed) before document lengths can be determined.
- Do a second pass over all documents: keep a list or hashtable with all document id's, and for each document determine its length.

Time Complexity of Indexing

- Complexity of creating vector and indexing a document of n tokens is O(n).
- So indexing m such documents is O(m n).
- Computing token IDFs can be done during the same first pass
- Computing vector lengths is also O(m n).
- Complete process is O(m n), which is also the complexity of just reading in the corpus.

Step 3: Retrieval

- **Input:** Query and Inverted Index (from Step 2)
- **Output:** Similarity values between query and documents

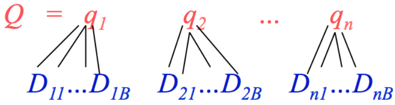
- Tokens that are not in both the query and the document have no effect on the cosine similarity.
    - Product of token weights is zero and does not contribute to the dot product.
- Usually the query is fairly short, and therefore its vector is *extremely* sparse.
- Use the inverted index (from Step 2) to find the limited set of documents that contain at least one of the query words.

Processing the Query

- Incrementally compute cosine similarity of each indexed document as query words are processed one by one.
- To accumulate a total score for each retrieved document, store retrieved documents in a hashtable, where the document id is the key and the partial accumulated score is the value.

Inverted Query Retrieval Efficiency

- Assume that, on average, a query word appears in B documents:



- Then retrieval time is $O(|Q|B)$, which is typically much better than naïve retrieval that examines all $|D|$ documents, $O(|V||D|)$, because $|Q| << |V|$ and $B << |D|$.

Step 4: Ranking

- Sort the hashtable including the retrieved documents based on the value of cosine similarity
- Return the documents in descending order of their relevance
- **Input:** Similarity values between query and documents
- **Output:** Ranked list of documents in reversed order of their relevance

Term Weights

- Weights applied to both document terms and query terms
- Direct impact on the final ranking
  - Direct impact on the results
  - Direct impact on the quality of IR system

Next

- Evaluation of IR systems