CIS 833 – Information Retrieval and Text Mining

Lecture 8

Vector Space Model & Evaluation in IR

September 17, 2015

Credits for slides: Hofmann, Mihalcea, Mobasher, Mooney, Schutze.

Assignments

- HW2 posted online
- The *warmup* WordCount MapReduce programming assignment has been posted online (due September 23rd)

Required Reading

- "Information Retrieval" textbook
  - Chapter 2: Term Vocabulary and Posting Lists
  - Chapter 4: Index Construction
  - Chapter 8: Evaluation in IR
- MapReduce textbook
  - Chapter 4: Inverted Indexing for text retrieval

Vector Space Model: Implementation Steps

Step 1: Preprocessing

Step 2: Indexing

Step 3: Retrieval

Step 4: Ranking

Review

- Given a document collection, what can be computed off-line?
- How do we store the inverted index?
- How do we calculate document lengths?
- How many passes through the data are necessary?
- Time complexity of indexing?

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.

Vector Space Model: Implementation Steps

Step 1: Preprocessing

Step 2: Indexing

Step 3: Retrieval

Step 4: Ranking

Step 3: Retrieval with an Inverted Index

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

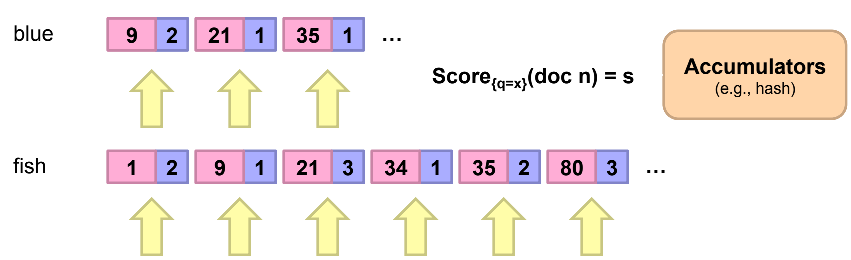
- Tokens that are not in both the query and the document do not affect cosine similarity.
  - Product of token weights is zero and does not contribute to the dot product.
- Usually the query is fairly short, and therefore its vector is *extremely* sparse.
- Use inverted index to find the limited set 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 DocumentReference is the key and the partial accumulated score is the value.

Retrieval: Query-At-A-Time

Evaluate documents one query term at a time

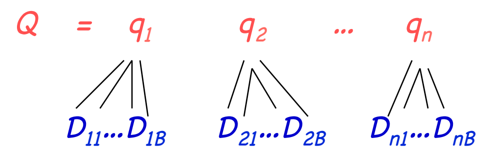


- We assume the query is "blue fish"

$$sim(d_j,d_k)=\frac{\vec{d}_j\cdot\vec{d}_k}{\left|\vec{d}_j\right|\left|\vec{d}_k\right|}=\frac{\sum_{i=1}^{n}w_{i,j}w_{i,k}}{\sqrt{\sum_{i=1}^{n}w_{i,j}^2}\sqrt{\sum_{i=1}^{n}w_{i,k}^2}}$$

Inverted Query Retrieval Efficiency

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



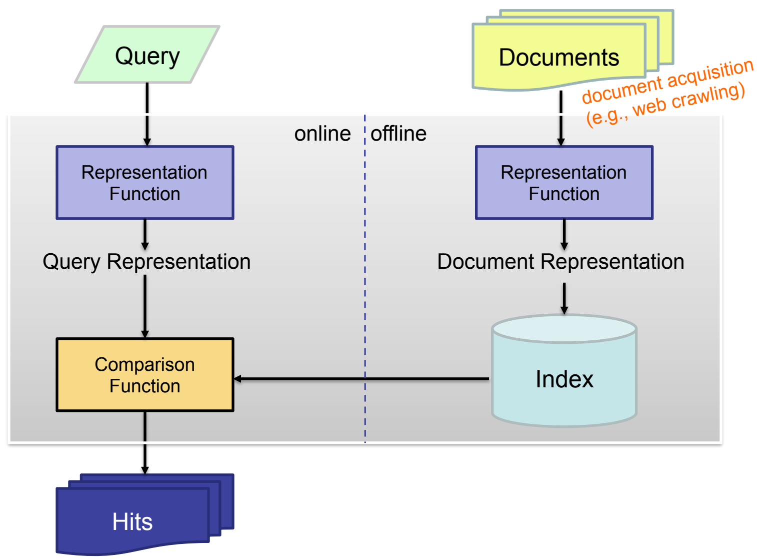
- Then retrieval time is O(|*Q*| *B*), which is typically, **much** better than naïve retrieval that examines all |*D*| documents, O(|*V*| |*D*|), because |*Q*| << |*V*| and *B* << |*D*|.

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

- <u>Input</u>: Similarity values between query and documents
- <u>Output</u>: Ranked list of documented in reversed order of their relevance

Abstract IR Architecture



MapReduce it?

- The indexing problem

  Perfect for MapReduce!
  - Scalability is critical
  - Must be relatively fast, but need not be real time
  - Fundamentally a batch operation
  - Incremental updates may or may not be important
- The retrieval problem
  - Must have sub-second response time
  - For the web, only need relatively few results

  Uh... not so good...

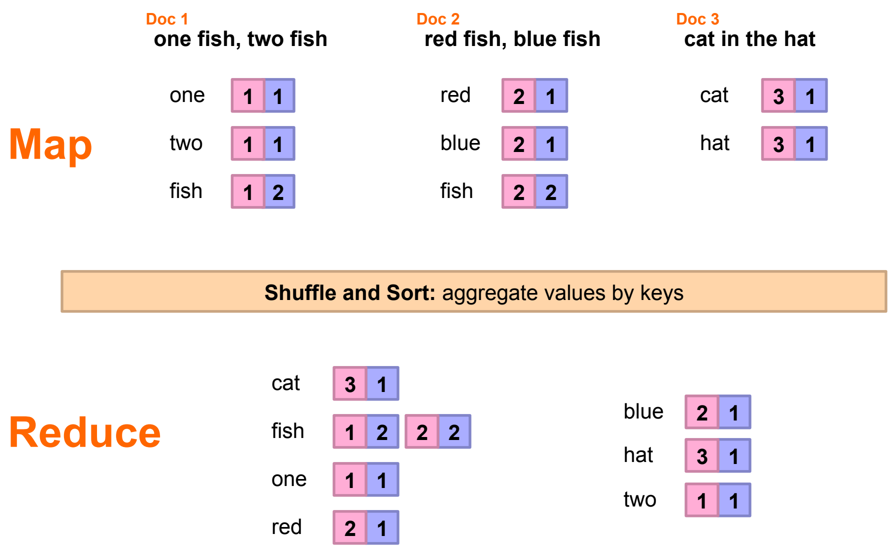
Indexing: Performance Analysis

- Fundamentally, a large sorting problem
  - Terms usually fit in memory
  - Postings usually don't – in our programming assignment, we assume that they fit
- How can it be done with MapReduce?

MapReduce: Index Construction

- Map over all documents
  - Emit *term* as key, (*docno*, *tf)* as value
- Sort/shuffle: group postings by term
- Reduce
  - Gather the postings (could also be useful to sort postings by *docno* or *tf*)
  - Write postings to disk
- MapReduce does all the heavy lifting!

Inverted Indexing with MapReduce



Inverted Indexing: Pseudo-Code

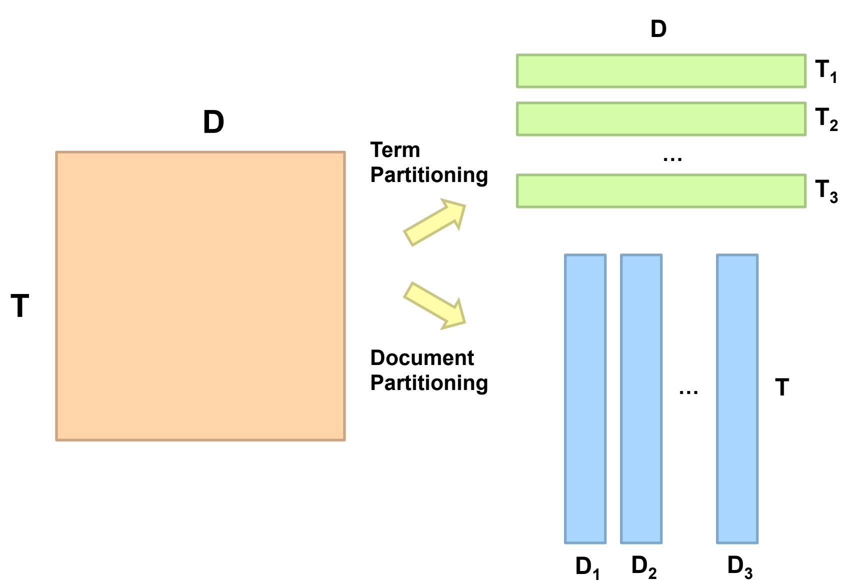
class MAPPER
    procedure MAP(docid n, doc d)
        $H \leftarrow$ new ASSOCIATIVEARRAY
        for all term $t \in$ doc d **do**
            $H\{t\} \leftarrow H\{t\} + 1$
        for all term $t \in H$ **do**
            EMIT(term t, posting $\langle n, H\{t\}\rangle$)

class REDUCER
    procedure REDUCE(term t, postings $[\langle a_1, f_1\rangle, \langle a_2, f_2\rangle \ldots]$)
        $P \leftarrow$ new LIST
        for all posting $\langle a, f\rangle \in$ postings $[\langle a_1, f_1\rangle, \langle a_2, f_2\rangle \ldots]$ **do**
            APPEND(P, $\langle a, f\rangle$)
        SORT(P)
        EMIT(term t, postings P)

Retrieval with MapReduce?

- MapReduce is fundamentally batch-oriented
  - Optimized for throughput, not latency
  - Startup of mappers and reducers is expensive
- MapReduce is not suitable for real-time queries!
  - Use separate approach for retrieval…

Term vs. Document Partitioning



Evaluation in IR

Why System Evaluation?

- There are many retrieval models/ algorithms/ systems, which one is the best?
- What is the best component for:
  - Ranking function (dot-product, cosine, …)
  - Term selection (stopword removal, stemming…)
  - Term weighting (TF, TF-IDF,…)
- How far down the ranked list will a user need to look to find some/all relevant documents?

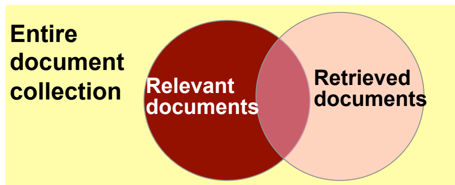
Difficulties in Evaluating IR Systems

- Effectiveness is related to the ***relevancy*** of retrieved items.
- From a human standpoint, relevancy is:
  - Subjective: Depends upon a specific user's judgment.
  - Situational: Relates to user's current needs.
  - Cognitive: Depends on human perception and behavior.
  - Dynamic: Changes over time.

Human Labeled Corpora (Gold Standard)

- Start with a corpus of documents.
- Collect a set of queries for this corpus.
- Have one or more human experts exhaustively label the relevant documents for each query.
- Typically assumes binary relevance judgments.
- Requires considerable human effort for large document/query corpora.

Precision and Recall



$$recall = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ relevant\ documents}$$

$$precision = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ documents\ retrieved}$$

Contingency Table

	retrieved	not retrieved	
relevant	w	x	$n_1 = w + x$
not relevant	y	z	
	$n_2 = w + y$		N

Precision and Recall

From all the documents that are relevant out there,
how many did the IR system retrieve?

Recall: $$\frac{w}{w+x}$$

From all the documents that are retrieved by the IR system,
how many are relevant?

Precision: $$\frac{w}{w+y}$$

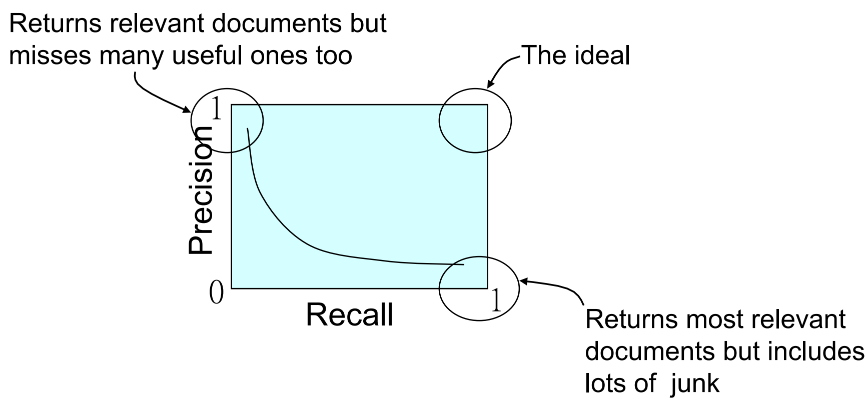
Precision and Recall

- Precision
  - The ability to retrieve top-ranked documents that are mostly relevant.
- Recall
  - The ability of the search to find ***all*** of the relevant items in the corpus.

Determining Recall is Difficult

- Total number of relevant items is not always available:
  - Sample across the database and perform relevance judgment on these items.
  - Apply different retrieval algorithms to the same database for the same query. The aggregate of relevant items is taken as the total relevant set.

Trade-off between Recall and Precision



Computing Recall/Precision Points

- For a given query, produce the ranked list of retrievals.
- Adjusting a threshold on this ranked list produces different sets of retrieved documents, and therefore different recall/precision measures.
- Mark each document in the ranked list that is relevant according to the gold standard.
- Compute a recall/precision pair for each position in the ranked list that contains a relevant document.

Computing Recall/Precision Points: Example 1

n	doc #	relevant
1	588	x
2	589	x
3	576	
4	590	x
5	986	
6	592	x
7	984	
8	988	
9	578	
10	985	
11	103	
12	591	
13	772	x
14	990	

Let total # of relevant docs = 6
Check each new recall point:

R=1/6=0.167; P=1/1=1

R=2/6=0.333; P=2/2=1

R=3/6=0.5; P=3/4=0.75

R=4/6=0.667; P=4/6=0.667

R=5/6=0.833; p=5/13=0.38

Missing one relevant document.
Never reach 100% recall