Reading Summary IR: Ch 1.1, 1.2, 6.3

Xin Qian
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
xinq@cs.cmu.edu

1. CH 1.1

This section uses the simple query example of *Brutus AND Caesar AND NOT Calpurnia* to bring out more retrieval requirements beyond a simple linear scan method. Large document collections, slop distance matching support and ranked retrieval support are three major requirements. The binary term-document *incidence matrix* has each row vector for a term, to show which document it appears in. It has each coloum vector for each document, to show which term appears in it. To answer the example query, we can take the corresponding vector and do a bitwise boolean operation. This brings us to several related concepts of the *Boolean retrieval model*, in which each query is written as a Boolean expression of terms. *Collection* and *corpus* are used interchangeably to refer the group of documents. *Ad hoc retrieval* searches relevant documents out of a collection according to user information need. *Information need* is the topic that user is interested to know about and expresses in the form of a *query*. *Relevant* means that the user receives valuable information w.r.t. their information need. Statistics like *precision* and *recall* are used to evaluate the *effectiveness* of an IR system. Instead of using a sparse term-document matrix, we might want to represent only 1 positions with *inverted index*. For each term in a *dictionary* of terms, the sorted *postings list* marks those documents that the term appeared in.

2. CH 1.2

Steps to build an index in advance includes collecting the documents, tokenizing the documents, normalizing tokens and creating the inverted index. In the 4th step, we input a list of pairs of term and docID. *Sorting* the list makes terms alphabetical. Occurrences of the same term are merged. Instances of the same term are grouped. *Document frequency* is the length of each postings list. Postings are then sorted by docID. We use either singly linked lists or variable length arrays, or even a hybrid scheme for the postings list for each term. The postings lists usually stored on disk.

11741'F16 Pittsburgh, Pennsylvania USA

DOI: 10.1145/1235

3. CH 6.3

The *vector space model* represents documents as vectors in a vector space. Each component in the vector represents each dictionary term. This *bag of words* representation is order insensitive. We compute the similarity between two documents with *cosine similarity - dot product* divided by the *length-normalize* product of two *Euclidean lengths*. We use this similarity measure to find the most similar document to a document d within a collection. *Term-document matrix* represents a collection of N documents. We could also view a *query* as a short document and use a vector to represent it. Likewise, we could also compute query-document similarity. The basic *term-at-a-time* scoring algorithm for computing vector space scores is illustrated in pseudo code snippet.