CMPS340 File Processing
Notes on Information Retrieval: Vector Model and Ranking
Based upon Chapter 4 (especially Section 4.4) of
***Managing Gigabytes* by Witten, Moffat, and Bell**

Still under construction...

Overview
The discipline of Information Retrieval (IR) is concerned, in part, with the problem of retrieving **relevant** information in response to user queries.

The typical scenario is that there is a *document database* or *corpus* (i.e., a set of (mostly textual) documents) and that a response to a query is in the form of a list of document ID's (or hyperlinks to the documents or to descriptions thereof) ranked according to their relevance to the user's need (as calculated by the IR system).

What Constitutes a Document?

That depends...

With respect to search engines (e.g., Google) on the WWW, it's typically a file containing a web page, an image, a Powerpoint presentation, etc.

in the ACM Digital Library, it's an article that appeared in some ACM publication.

in the U of S library's IR system, it's a cataloged item (e.g., book, DVD)

in a business IR system, it's an internal e-mail, memo, or contract

in an IR system for searching the Bible, it could be a Bible verse

in an IR system for searching The Works of Shakespeare, it could be a verse in a poem, or a scene in a play.

Issues of **granularity**. Coarse (e.g., entire book) vs. fine (one sentence, one verse, or even one word!).

Semantic Content of Documents is Represented by Terms
Now, how can a computer form a "judgement" about which documents are relevant to a user? Well, currently the whole process is fairly crude.

Basically, the semantic content of each document is represented by a list of **representative terms**. Traditionally, humans would devise such a list for cataloging purposes. (Persons of sufficient age will remember the days when libraries had *card catalogs* to aid searches for books based upon title, author, and subject. Indeed, this is probably the origin of "index card".) With the advent in recent times of digital computers and, in particular, documents stored in digital form, it has become common to construct indexes using *all* the words that occur in a document (and, likely, their frequencies of occurrence), probably **stemmed** and **stopped**.

The user query is simply a list of terms, or possibly such a list with boolean operators (e.g., `golf AND (club OR course)`).

The IR system takes a query and searches within the document database (corpus) (whose documents are indexed according to the terms that represent them) for documents associated to any terms that appear in the query (or that do not, if the `NOT` operator is used). The IR system applies some kind of **similarity** measure between the query and each document in order to calculate each document's **relevance**. Then it outputs the list of relevant documents, ranked from most to least relevant.

How is this all done?

Over the years, researchers have devised several different models upon which IR is based, including the "standard" **boolean**, **vector**, and **probabilistic** models. Within each model, there are lots of variations as to exactly how relevance is estimated.

Evaluating/Measuring Retrieval Effectiveness

How can we determine the quality of an IR system (or, for that matter, the quality of its response to a particular query)? There are two basic measures: **precision** and **recall**.

Suppose that, for a particular query Q (and a particular document database) the set of reLevant documents is L and that the IR system, in response to Q, reTrieves the set of documents T. Then

$$\text{Precision} = \frac{|T \cap L|}{|T|}$$

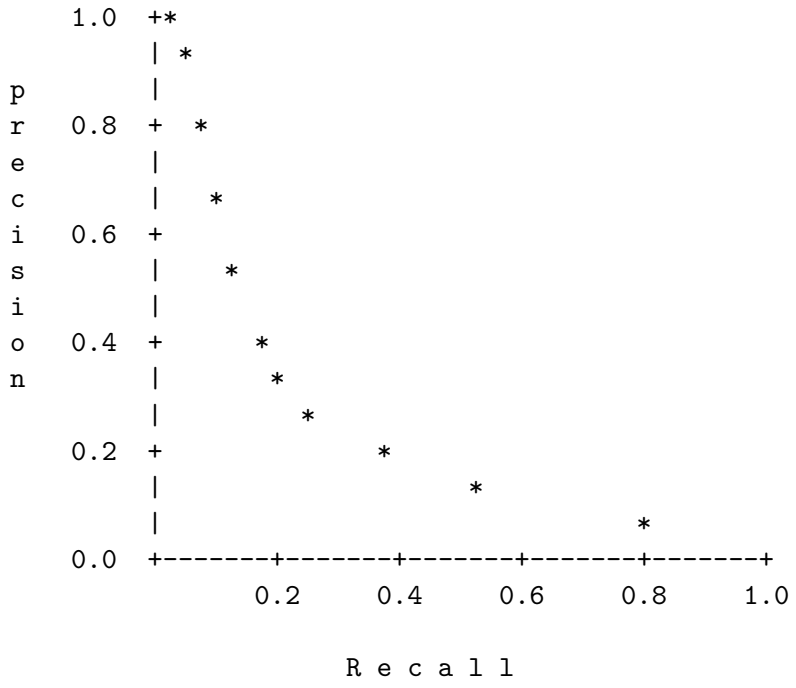
$$\text{Recall} = \frac{|T \cap L|}{|L|}$$

That is, precision is a measure of the likelihood that a retrieved document is relevant and recall is a measure of the likelihood that a relevant document is retrieved.

Now, each measure, by itself, is not very useful, because it is easy to design an IR system that tends to score highly in one or the other:
To achieve high recall, an IR system could simply retrieve every document in response to every query! To achieve high precision, retrieve no documents (in which case all those retrieved are relevant)!

Hence, these two measures are usually combined into a **recall/precision curve** that plots precision as a function of recall, in a sense. That is, we scan down the list of retrieved documents and, upon finding a relevant one, we calculate the precision at that level of recall. A typical graph has a shape similar to that shown in the nearby figure.

Of course, being able to make these measurements at all assumes that there is some final arbiter of which documents are relevant and which are not! Hence, these measurements cannot be applied to all queries. Rather, IR researchers have set up document databases and queries and experts have judged which documents are relevant to each query. Then researchers can do experiments with their new ranking algorithms to see how they stack up to others.



One of most well known is the TREC (Text Retrieval Conference) document database that was formed in about 1990.

How Document Ranking is Done

Let's start with the boolean model. Here, a query specifies an exact set of relevant documents because it is taken to be a boolean expression indicating exactly which combinations of terms are to be present/absent in order for a document to be relevant.

Example: golf AND (club OR course) AND (NOT Tiger)

The relevant documents are those that
(1) are associated to the term *golf*,
(2) are associated to at least one of *club* or *course*, but
(3) are not associated to *Tiger*

Under this model, the response to each query is to partition the set of documents into two categories: relevant and not relevant! In some settings (such as when the user knows exactly what information he needs), this is acceptable, or even preferred. (Consider the kinds of queries applied to a relational database via SQL.) But in other settings (such as typical web searches, where the information need is known less precisely), users would prefer to provide a query that is simply a list of words (implicitly connected by OR (AND?) operators) and to receive as a response a more nuanced ranking of relevant documents.

Also, the boolean model tends to be very sensitive to minor variations in the query. For example, the query *text AND compression AND retrieval* is likely to produce a much different

set of results from the query *data AND compression AND retrieval*.

This leads us to the **vector model**.

Here, each document (and query) is characterized by an n-dimensional vector, where n = number of distinct terms in the lexicon/dictionary. Then, to rank the documents by relevance (to a given query), we calculate a **similarity measure** between the query and each document. Documents are then ranked in descending order occording to their degree of similarity with the query.

One similarity measure is **inner product** (or **dot product**):

$$M(Q, D) = Q \bullet D = \sum_{i=1}^{n} (x_i \cdot y_i)$$

A very primitive version of this (in which the vectors are all boolean) is illustrated (see Table 4.7, page 182) with respect to the set of documents in Table 4.6 (page 181) and the two queries "eat" and "hot porridge".

Applying the similarity measure to D_1, we get

$$M(\text{“hot porridge}'', D_1) = (0, 0, 0, 1, 0, 0, 0, 0, 1, 0) \bullet (1, 0, 0, 1, 0, 0, 0, 1, 1, 0) = 2$$

Meanwhile, $M(\text{“hot porridge}'', D_2) = 1$, so D_1 gets a higher ranking, as we would expect (as it contains both terms in the query whereas D_2 contains only one).

This similarity measure has some drawbacks, though:

(1) It takes no account of term frequency (within a document)
(2) It takes no account of term scarcity (within the set of documents)
(3) It favors long documents (because a larger number of terms tends to appear in them).

The first problem is addressed by using **document term-weights** rather than boolean (0 or 1) values in the vectors describing documents.

The most obvious value to use for document term weight ($w_{d,t}$) (the weight of term t in document d) is $f_{d,t}$, the number of times term t occurs in D_d. Doing this, our D_1 vector becomes $(1, 0, 0, 1, 0, 0, 0, 2, 2, 0)$, so that now $M(\text{“hot porridge}'', D_1) = 3$, the increase due to the two occurrences of "porridge" in D_1.

More generally, we can also assign a query term weight $w_{q,t}$. (Discussion below.) The similarity measure is then

$$M(Q, D_d) = \sum_{1 \le t \le n} (w_{q,t} \cdot w_{d,t})$$

Now, how to address the "scarcity of terms problem"? That is, we want to attach more weight to terms that appear infrequently in the document database. (Example: "gorn" vs. "dog".)

Let f_t be the number of documents in which term t occurs. Then we want terms with small f_t values to have a large weight w_t associated to them.

Various definitions of *term weight* exist, but one popular one is

$$w_t = \log_e(1 + \frac{N}{f_t})$$

where N is the number of documents in the collection. We can use a term weight in a few different ways. One way is to multiply it by *relative term frequency* ($r_{d,t}$), which is a measure of a term's frequency within a document, in order to obtain $w_{d,t}$.

There are various ways to compute the relative term frequency component $r_{d,t}$, too. A popular one (see page 185) is

$$r_{d,t} = 1 + \log_e f_{d,t}$$

The document term weights are typically calculated either as

$$w_{d,t} = r_{d,t} \qquad \text{or} \qquad w_{d,t} = r_{d,t} \cdot w_t$$

The second of these is a particular case of the TF $\times$ IDF rule: term frequency times inverse document frequency.

The query-term weights $w_{q,t}$ are calculated similarly, where the within-query frequency $f_{q,t}$ may or may not be taken into account, and similarly for term weight w_t.

Based upon the research, there is no one choice for the quantities discussed here that consistently outperforms the others over a range of different queries.

But there is a widely held belief that, whatever particular formulas we choose, two monotonicity constraints should be followed:

If term t_1 appears in more documents than t_2 (i.e., $f_{t_1} > f_{t_2}$) , t_1's weight should be at most t_2's ($w_{t_1} \leq w_{t_2}$), and if t_1 appears in D_d more often than t_2 ($f_{d,t_1} > f_{d,t_2}$), then the document-term weight of t_1 with respect to D_d should be no less than t_2's ($w_{d,t_1} \geq w_{d,t_2}$).

To try to erase the long-documents-are-favored problem, we introduce a normalization factor (see page 185), which "normalizes" $M(Q, D_d)$ by dividing by $|D_d|$, a measure of document d's length. One possibility is to take

$$|D_d| = \sum_{1 \leq t \leq n} f_{d,t}$$

which is just the the number of occurrences of indexed terms in D_d. Another possibility is to take the square root of this number. Yet another is to use the Euclidean length of the document's vector of docment-term weights:

$$|D_d| = W_d = \sqrt{\sum_{1 \leq t \leq n} w_{d,t}^2}$$

Our similarity measure becomes

$$M(Q, D_d) = \frac{\sum_{1 \leq t \leq n}(w_{q,t} \cdot w_{d,t})}{|D_d|}$$