Information Retrieval Methods

By Kit Marlow

CIS650

02/26/03

Sources

- Christos Faloutsos and Douglas W. Oard. A Survey of Information Retrieval and Filtering Methods. Technical Report, University of Maryland, 1995.

- Gerald Salton and Christopher Buckley. Term Weighting Approaches in Automatic Text Retrieval. Information Processing and Management, vol. 24, no. 5, pp. 513--523, 1988.

- If you want more information, a fun book is:

  Modern Information Retrieval by Ricardo Baeza-Yates and Berthier Ribeiro-Neto. Addison Wesley, 1999.

Databases vs. Information Retrieval

DATABASES

We know the schema in advance, so semantic correlation between queries and data is clear.

We can get exact answers

Strong theoretical foundation (at least with relational…)

IR

No schema, but rather unstructured natural language text. The result is that there is not a clear semantic correlation between queries and data.

We get inexact, estimated answers

Theory not well understood (especially Natural Language Processing)

IR – lots of junk

- Because of the semantic disconnect between query and documents, IR is liable to return a lot of junk.
- So, the IR System has to interpret and rank its documents, according to how relevant to they are to the user's query.
- "The notion of relevance is at the center of information retrieval." - Baeza-Yates, p.2

Condensing the Data

- IR systems condense and simplify searchable documents by getting a *logical view* of each doc
- To do this, we get a set of keywords (“index terms”) that are representative of the document
- Store the signatures for a set of documents together in one small and quickly searchable file.
- To keep the size of this file small, we can eliminate stopwords (“and”, “the”), and we can *stem* words to their roots (‘clone’ from cloning or cloned), we can limit our list to nouns, and we can compress the list.
- Now we have a neat, easily searchable index for these documents.
- All of the traditional IR models are built on this kind of indexing system.

Signature Files

- Word oriented index-structures based on hashing. Maps words to a bit mask (which gives word occurrence info) and to a pointer to the original document.
- Compresses a document into a ‘signature’
- Advanced knowledge of occurrence frequencies can allow for an organization of the signature file in a way that reduces false drops and the negative effects of skew.

Inverted Files

- Important – most indices use some variant of the inverted file.
- A list of sorted words, each associated with a set pointers to the page in which it occurs.
- Inverted files do better than signature files for most applications. Used in nearly all commercial systems.

Some definitions

d_j = a document

k_i = an index term

$w_{i,j}$ = a weight associated with a doc-and-index term pair

(Zero if the term is not in the document)

$K = \{k_1, \ldots k_t\}$, is the set of all index terms over all docs in the system

q = the query

$\vec{d}_j = (w_{1,j}, w_{2,j} \ldots w_{n,j})$ - index term vector describing the relevance (weight) of each index term in the system to this document

The Boolean Model

-- The simplest retrieval model

--Queries are index terms linked by AND, OR or NOT. They are converted into disjunctive normal form, where each part is a binary weighted vector corresponding the tuple

(k_a, k_b, k_c)

--So the weights for each keyword end up either 1 or 0 – there, or not there

--Disadvantage: since weights are binary, docs are either relevant or irrelevant, there's no further ranking.

--we're going to get a lot of irrelevant junk.

$$q = k_a \wedge (k_b \vee \neg k_c)$$

$$\vec{q} = (1,1,1) \vee (1,1,0) \vee (1,0,0)$$

Measuring Document-Query Relevance

The Boolean model is crude -- indexed document keywords actually vary in relevance to the query.

- “Making a pie is easy. It is not rocket science.”
- ‘pie’ is of low relevance in a query for a document on rockets.
- How do we measure degree of relevance?

The Vector Model, I

- Non-binary weighting
- the relevance of index terms to a query and to documents are quantified as a graded scale of weights.

$W_{i,q}$ = weight associated with a query and a document index term in the system.

$W_{i,j}$ = weight associated with a document and a document index term in the system

How are index terms weighted?

Weighting Index Terms

- The weight of an index term is proportional to its frequency in a document (term frequency or tf factor), and inversely proportional to its frequency among all documents in the system (inverse doc frequency or idf factor).
- A word like “report” will show up in a relatively high number of documents, so it can’t be very useful in distinguishing this document from all others. So the word’s idf factor would be high, compared to a word like “Crotaphytus” (assuming it’s not a lizard database).

Calculating term frequency

$freq_{i,j}$ = times index term k_i shows up in doc d_j

max = all terms in document d

$$freq_{i,j} = \frac{freq_{i,j}}{\max_l freq_{l,j}}$$

Calculating Inverse Doc Frequency

N = total number of docs

n_i= number of documents in which term k_i appears

$$\text{idf}_i = \log \frac{N}{n_i}$$

Calculating Index Term Weight

$$w_{i,j} = f_{i,j} * \log \frac{N}{n_i}$$

The Vector Model, II

Index term weights with relation to each doc, and to the query, are stored in vectors.

$\vec{d}_j = (w_{1,j}, w_{2,j} \ldots w_{t,j})$ - the document vector describing the relevance of each index term in the system to document d_j. (t is the number of index terms in the system).

$\vec{q} = (w_{1,q}, w_{2,q} \ldots w_{t,q})$ - the query vect or describing the relevance of each index term in the system to the query (t is the total number of index term s in the system)

The Vector Model, III

- Document-Query Relevance is measured by the correlation between a document vector and the query vector.
- All document vectors are measured against the query vector
- The correlation is quantified as the cosine of the angle between the two vectors
- Similarity sim(d_j, q) will be a value that ranges from 0 to 1.
- The IR system can set a threshold somewhere in between and return only the docs above that threshold of similarity.

The Vector Model, IV

- similarity(d_j, q) =

$$\frac{\sum_{i=1}^{t} w_{i,j} * w_{i,q}}{\sqrt{\sum_{i=j}^{t} (w_{i,j} * w_{i,j})} * \sqrt{\sum_{j=1}^{t} (w_{i,q} * w_{i,q})}}$$

Vector Model, V

- The Vector Model works because the documents returned are more relevant to our query, and they are *ranked* by relevance to the query.

Probabilistic Model

- Also known as the binary independence retrieval model (called binary because the index term weights for the docs and the query are 1 or 0). Performs about as well as the vector model (vector model probably a bit better).
- Start by guessing the probability that an index term in a query will show up in a set of retrieved docs. Then use a recursive process on the retrieved docs to improve upon this guess.

- R = set of relevant docs (or guessed to be relevant)

  $\overline{R}$ = set of irrelevant docs (or guessed to be irrelevant)

Probabilistic Method, III

Since we do not know R in the beginning, we initially assume $P(k_i \mid R)$ to be a constant (say, 0.5) for all index terms k, and

approximate $P(k_i \mid \bar{R})$ as $\frac{n_i}{N}$ where n_i is the number of docs containing index term k_i, and N is the total number of documents in the collection.

With this initial guess, we can retrieve some documents containing query terms and give them an initial probabilistic ranking.

Then, we can can gradually improve the ranking.

Improving the Ranking

- To improve our probabilistic ranking, we need to improve our guesses of the probability that
- k will appear in R and not-R:

V = subset of docs initially retrieved and ranked

V_i = subset of V which contain the index term k_i.

New Estimates:

$$P(k_i \mid R) = \frac{V_i}{V} \text{ (docs that contain } k_i \text{ divided by docs retrieved)}$$

$$P(k_i \mid \bar{R}) = \frac{n_i - V_i}{N - V} \text{ (considers that all the non - retrieved documents are irrelevant)}$$

This process can be repeated recursively!

Probabilistic Model, IV

- Its main advantage: documents are ranked in decreasing order of their probability of being relevant to the query.
- Disadvantage: Since weights are binary, the model can't take advantage of an index term's frequency within a document.

The End