It has been well-known that the Boolean model is too inflexible, requiring skilful use of Boolean operators to obtain good results. On the other hand, the vector space model is flexible but not precise enough. Is there a middle ground?

1 Extended Boolean Model

1.1 What is wrong with the Boolean and vector space models?

We can see that the Boolean model has extremely rigid conditions that the returned documents must satisfy. On the other hand, the pure vector space model essentially applies the OR operation to the query keywords, matches the keywords against a document, and sums up the score of each query term that occurs in a document. Thus, the vector space model can theoretically rank at the top a document that matches only one query term. As discussed earlier in the vector space model, this does not meet the expectation of the searchers.

To add ranking capability to the Boolean model, we saw previously that the Boolean model can be used as a pre-filter to restrict the documents to a subset that satisfies the Boolean condition and the vector space model can then be applied to the subset to produce a ranked list of results. This approach is suitable for (and indeed largely adopted by) web search engines because all web search engines nowadays assume by default the AND operator between the query keywords when users don't specify any Boolean operators in the query (99% of users don't!). When the web has an *abundance of information*, the default AND operator works very well.[1]

Information abundance on the web changed the way search engines should work.

The Boolean pre-filter inherits the restriction of the Boolean model in that when the user specifies an AND operator in the query, the condition may already be too restricted that the result had excluded many useful documents which would otherwise ranked very high in the result. When the user uses an OR operator then it has no difference from the simple vector space model. Again, such design assumes that users are able to specify a good Boolean expression for his query, which is in general far from the truth.

1.2 Integrating Boolean and ranking

The Extended Boolean Model is an interesting extension of the Boolean model that combines the control of the Boolean model and the ranking capability of the vector space model into a *uniform framework*. The idea is very simple. Users can use AND and OR in the queries (as in the Boolean model) but keywords are weighted (as in the vector space model). Furthermore, documents are ranked by a similarity function that is designed to exhibit the following behaviors:

[1] In fact, Google was one of the first, if not the first, search engines that assumes an AND operator between query keywords, when at least Alta Vista insisted an OR operator for a long time until Google had proven to be a threat to all first-generation search engines.

- For conjunctive queries, documents containing all keywords are awarded higher scores, while documents that don't contain all of the query keywords still receive non-zero scores and as such have a chance to be ranked high should the matching query keywords have heavy enough weights.
- For disjunctive queries, documents containing all keywords are awarded with higher scores and documents that don't contain all of the query keywords shall receive non-zero weights so that they too have a good chance to be ranked high in the result.

The two cases are very similar. The only difference is that in the conjunctive case, the missing of one query keyword in a document results in a heavy penalty on the document's score but in the disjunctive case, the penalty is relatively small. Now, let's look at how the similarity formula achieves these effects. In the following, we assume that the weight of a keyword is normalized to [0,1]. For example, the tfxidf weighting formula can be modified as:

$$w_{x,j} = \frac{tf_{x,j}}{max_l tf_{l,j}} \times \frac{idf_x}{max_i idf_i} \tag{1}$$

where $w_{x,j}$ is the weight of term x in document j, the term frequency $tf_{x,j}$ is normalized by the highest tf in document j, and the idf of term x is normalized by the highest idf among all keywords in the collection.

1.3 Disjunctive queries

Given a disjunctive query $x \vee y$, and the weights of x and y in document j are w_x and w_y, respectively, the similarity between the query and document j is defined as follows.

$$sim(q_{or}, d_j) = \left(\frac{w_{x,j}^2 + w_{y,j}^2}{2} \right)^{0.5} \tag{2}$$

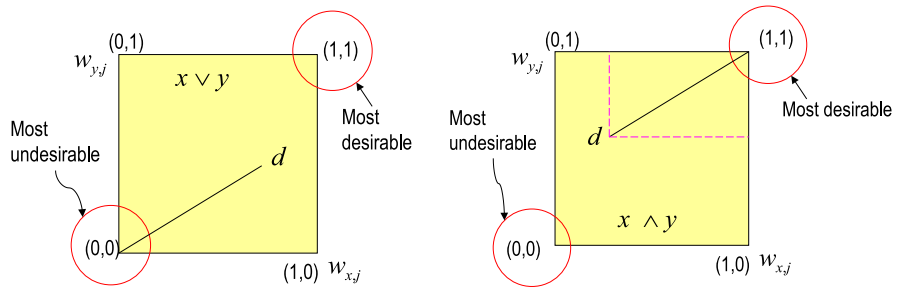
The geometric interpretation of the function is shown in Figure 1(a). In essence, a document has the highest possible weight at (1,1), meaning that it contains both x and y and their weights in the document are highest possible (i.e., equal to 1). The similarity function measures the vector length of document d from the origin.

1.4 Conjunctive queries

Given a conjunctive query $x \wedge y$, the similarity between q and document j is defined as follows.

$$sim(q_{and}, d_j) = 1 - \left(\frac{(1 - w_{x,j})^2 + (1 - w_{y,j})^2)}{2} \right)^{0.5} \tag{3}$$

The similarity function measures the *complement* of the vector length d from the coordinates (1,1). Intuitively, a document at (1,1) would be the best match



(a) $x \vee y$. (b) $x \wedge y$.

Figure 1: Geometric interpretation.

d_j contains	Term weights	Document Score	
		$sim(x \vee y, d_j)$	$sim(x \wedge y, d_j)$
$xx..xx$	$wt_{x,j} = 0.5$	0.353	0.209
$xx..xx$	$wt_{x,j} = 1.0$	0.707	0.293
$xx..yy$	$wt_{x,j} = wt_{y,j} = 0.5$	0.5	0.5
$xx..yy$	$wt_{x,j} = wt_{y,j} = 1.0$	1.0	1.0

Table 1: The variation of document score against term weights.

to the query, and the farther away a document is from this ideal position, the lower is its score with respect to q. Figure 1(b) is the geometric interpretation of the similarity function.

1.5 Observations

Let's look at how the score of a document varies according to (i) the terms it contains and the term weights, and (ii) the type of the query. Table 1 shows the variation of the score of document d_j for both disjunctive and conjunctive queries when: (i) d_j contains only one query keyword (x is assumed), and (ii) when d_j contains both x and y.

We can see that if a document contains both terms, the document score increases rapidly to the maximum (i.e., 1). The behavior is the same for both conjunctive and disjunctive queries. However, when a document contains only one term, the existence of the term gives a gain 0.707 to an OR query but only 0.293 to an AND query. In other words, missing a term gives a much smaller penalty to an OR query than an AND query. This is consistent with the intuition that when the user specifies an AND query, he expects to see both terms in a document. Thus, when a document contains only one term it should be heavily penalized. This is not the case for an OR query when the user expects

The Extended Boolean model is consistent with user's expectation.

to see only one of the two terms. Therefore, a document containing only one of the two query terms should get a reasonably high score whereas a document containing both terms exceeds the user's expectation and thus should get the highest possible score (i.e., 1).

1.6 Generalization of the Extended Boolean model

The description of the Extended Boolean Model is based on two query keywords. Needless to day, the model has to be able to handle more than two query keywords. The extended similarity functions for m query keywords are as follows.

$$sim(q_{or}, d) = \left(\frac{x_1^2 + x_2^2 + ... + x_m^2}{m}\right)^{0.5} \tag{4}$$

$$sim(q_{and}, d) = 1 - \left(\frac{(1-x_1)^2 + (1-x_2)^2 + ... + (1-x_m)^2}{m}\right)^{0.5} \tag{5}$$

We can see that so far the similarity function is expressed based on the Euclidean lengths of the document vectors, i.e., the L_2 norm. In fact, the similarity function can be further generalized into the *p-norm* model by replacing the L_2 norm with *p-norm*.

$$sim(q_{or}, d) = \left(\frac{x_1^p + x_2^p + ... + x_m^p}{m}\right)^{1/p} \tag{6}$$

$$sim(q_{and}, d) = 1 - \left(\frac{(1-x_1)^p + (1-x_2)^p + ... + (1-x_m)^p}{m}\right)^{1/p} \tag{7}$$

When the value of p varies, the model degenerates into different special cases. When $p = 1$, $sim(q_{or}, d) = sim(q_{and}, d) = (x_1 + x_2 + ... + x_m)/m$. The model degenerates into the inner product similarity function in the vector-space model.

When $p = \infty$, $sim(q_{or}, d) = max(x_i); sim(q_{and}, d) = min(x_i)$. This is equivalent to the fuzzy logic model.

In the *p-norm* model, the AND and OR operators in the same query can be associated with different p values. For example, the query $(x \vee^2 y) \wedge^\infty z$ means the following:

- d must contain z because when z does not exist, $sim() = 0$
- When x and y both do not exist, $sim() = 0$
- When z exists, $sim() = min(w_z, sim(x \vee^2 y, d))$

By assigning different p values to the Boolean operators, a keyword can be interpreted differently in different operator. For example, y has a L_2 norm in the AND operator but an infinite norm in the OR operator and thus creating different effects. It is clear that the Boolean expression is not commutative any more and that despite its flexibility and elegance, the generalized *p-norm* model is very difficult to use. It is not clear what impacts the p value has on

the overall retrieval quality. In order to make the model useful for the general users, perhaps the Boolean expression and the value of p must be obtained and adapted using machine learning methods.

1.7 Questions

1. At the time of writing, none of the major search engines make use of the Extended Boolean model, why?

