7

Retrieval Models

"There is no certainty, only opportunity."

V, *V for Vendetta*

7.1 Overview of Retrieval Models

During the last 45 years of information retrieval research, one of the primary goals has been to understand and formalize the processes that underlie a person making the decision that a piece of text is relevant to his information need. To develop a complete understanding would probably require understanding how language is represented and processed in the human brain, and we are a long way short of that. We can, however, propose theories about relevance in the form of mathematical retrieval models and test those theories by comparing them to human actions. Good models should produce outputs that correlate well with human decisions on relevance. To put it another way, ranking algorithms based on good retrieval models will retrieve relevant documents near the top of the ranking (and consequently will have high effectiveness).

How successful has modeling been? As an example, ranking algorithms for general search improved in effectiveness by over 100% in the 1990s, as measured using the TREC test collections. These changes in effectiveness corresponded to improvements in the associated retrieval models. Web search effectiveness has also improved substantially over the past 10 years. In experiments with TREC web collections, the most effective ranking algorithms come from well-defined retrieval models. In the case of commercial web search engines, it is less clear what the retrieval models are, but there is no doubt that the ranking algorithms rely on solid mathematical foundations.

It is possible to develop ranking algorithms without an explicit retrieval model through trial and error. Using a retrieval model, however, has generally proved to be the best approach. Retrieval models, like all mathematical models, provide a framework for defining new tasks and explaining assumptions. When problems are observed with a ranking algorithm, the retrieval model provides a structure for testing alternatives that will be much more efficient than a *brute force* (try everything) approach.

In this discussion, we must not overlook the fact that relevance is a complex concept. It is quite difficult for a person to explain why one document is more relevant than another, and when people are asked to judge the relevance of documents for a given query, they can often disagree. Information scientists have written volumes about the nature of relevance, but we will not dive into that material here. Instead, we discuss two key aspects of relevance that are important for both retrieval models and evaluation measures.

The first aspect is the difference between *topical* and *user* relevance, which was mentioned in section 1.1. A document is topically relevant to a query if it is judged to be on the same topic. In other words, the query and the document are about the same thing. A web page containing a biography of Abraham Lincoln would certainly be topically relevant to the query "Abraham Lincoln", and would also be topically relevant to the queries "U.S. presidents" and "Civil War". User relevance takes into account all the other factors that go into a user's judgment of relevance. This may include the age of the document, the language of the document, the intended target audience, the novelty of the document, and so on. A document containing just a list of all the U.S. presidents, for example, would be topically relevant to the query "Abraham Lincoln" but may not be considered relevant to the person who submitted the query because they were looking for more detail on Lincoln's life. Retrieval models cannot incorporate all the additional factors involved in user relevance, but some do take these factors into consideration.

The second aspect of relevance that we consider is whether it is *binary* or *multivalued*. Binary relevance simply means that a document is either relevant or not relevant. It seems obvious that some documents are less relevant than others, but still more relevant than documents that are completely off-topic. For example, we may consider the document containing a list of U.S. presidents to be less topically relevant than the Lincoln biography, but certainly more relevant than an advertisement for a Lincoln automobile. Based on this observation, some retrieval models and evaluation measures explicitly introduce relevance as a multivalued variable. Multiple levels of relevance are certainly important in evaluation, when people are asked to judge relevance. Having just three levels (relevant, non-relevant, unsure) has been shown to make the judges' task much easier. In the case of retrieval models, however, the advantages of multiple levels are less clear. This is because most ranking algorithms calculate a *probability* of relevance and can represent the uncertainty involved.

Many retrieval models have been proposed over the years. Two of the oldest are the *Boolean* and *vector space* models. Although these models have been largely superseded by probabilistic approaches, they are often mentioned in discussions about information retrieval, and so we describe them briefly before going into the details of other models.

7.1.1 Boolean Retrieval

The Boolean retrieval model was used by the earliest search engines and is still in use today. It is also known as *exact-match retrieval* since documents are retrieved if they exactly match the query specification, and otherwise are not retrieved. Although this defines a very simple form of ranking, Boolean retrieval is not generally described as a ranking algorithm. This is because the Boolean retrieval model assumes that all documents in the retrieved set are equivalent in terms of relevance, in addition to the assumption that relevance is binary. The name Boolean comes from the fact that there only two possible outcomes for query evaluation (TRUE and FALSE) and because the query is usually specified using operators from Boolean logic (AND, OR, NOT). As mentioned in Chapter 6, proximity operators and wildcard characters are also commonly used in Boolean queries. Searching with a regular expression utility such as `grep` is another example of exact-match retrieval.

There are some advantages to Boolean retrieval. The results of the model are very predictable and easy to explain to users. The operands of a Boolean query can be any document feature, not just words, so it is straightforward to incorporate metadata such as a document date or document type in the query specification. From an implementation point of view, Boolean retrieval is usually more efficient than ranked retrieval because documents can be rapidly eliminated from consideration in the scoring process.

Despite these positive aspects, the major drawback of this approach to search is that the effectiveness depends entirely on the user. Because of the lack of a sophisticated ranking algorithm, simple queries will not work well. All documents containing the specified query words will be retrieved, and this retrieved set will be presented to the user in some order, such as by publication date, that has little to do with relevance. It is possible to construct complex Boolean queries that narrow the retrieved set to mostly relevant documents, but this is a difficult task that requires considerable experience. In response to the difficulty of formulating queries, a class of users known as search intermediaries (mentioned in the last chapter) became associated with Boolean search systems. The task of an intermediary is to translate a user's information need into a complex Boolean query for a particular search engine. Intermediaries are still used in some specialized areas, such as in legal offices. The simplicity and effectiveness of modern search engines, however, has enabled most people to do their own searches.

As an example of Boolean query formulation, consider the following queries for a search engine that has indexed a collection of news stories. The simple query:

lincoln

would retrieve a large number of documents that mention Lincoln cars and places named Lincoln in addition to stories about President Lincoln. All of these documents would be equivalent in terms of ranking in the Boolean retrieval model, regardless of how many times the word "lincoln" occurs or in what context it occurs. Given this, the user may attempt to narrow the scope of the search with the following query:

president AND lincoln

This query will retrieve a set of documents that contain both words, occurring anywhere in the document. If there are a number of stories involving the management of the Ford Motor Company and Lincoln cars, these will be retrieved in the same set as stories about President Lincoln, for example:

Ford Motor Company today announced that Darryl Hazel will succeed Brian Kelley as **president** of **Lincoln** Mercury.

If enough of these types of documents were retrieved, the user may try to eliminate documents about cars by using the NOT operator, as follows:

president AND lincoln AND NOT (automobile OR car)

This would remove any document that contains even a single mention of the words "automobile" or "car" anywhere in the document. The use of the NOT operator, in general, removes too many relevant documents along with non-relevant documents and is not recommended. For example, one of the top-ranked documents in a web search for "President Lincoln" was a biography containing the sentence:

Lincoln's body departs Washington in a nine-**car** funeral train.

Using NOT (automobile OR car) in the query would have removed this document. If the retrieved set is still too large, the user may try to further narrow the query by adding in additional words that should occur in biographies:

> president AND lincoln AND biography AND life AND birthplace AND gettysburg AND NOT (automobile OR car)

Unfortunately, in a Boolean search engine, putting too many search terms into the query with the AND operator often results in nothing being retrieved. To avoid this, the user may try using an OR instead:

> president AND lincoln AND (biography OR life OR birthplace OR gettysburg) AND NOT (automobile OR car)

This will retrieve any document containing the words "president" and "lincoln", along with any one of the words "biography", "life", "birthplace", or "gettysburg" (and does not mention "automobile" or "car").

After all this, we have a query that may do a reasonable job at retrieving a set containing some relevant documents, but we still can't specify which words are more important or that having more of the associated words is better than any one of them. For example, a document containing the following text was retrieved at rank 500 by a web search (which does use measures of word importance):

> **President**'s Day - Holiday activities - crafts, mazes, word searches, ... "The **Life** of Washington" Read the entire book online! Abraham **Lincoln** Research Site ...

A Boolean retrieval system would make no distinction between this document and the other 499 that are ranked higher by the web search engine. It could, for example, be the first document in the result list.

The process of developing queries with a focus on the size of the retrieved set has been called *searching by numbers*, and is a consequence of the limitations of the Boolean retrieval model. To address these limitations, researchers developed models, such as the vector space model, that incorporate ranking.

7.1.2 The Vector Space Model

The vector space model was the basis for most of the research in information retrieval in the 1960s and 1970s, and papers using this model continue to appear at conferences. It has the advantage of being a simple and intuitively appealing framework for implementing term weighting, ranking, and relevance feedback.

Historically, it was very important in introducing these concepts, and effective techniques have been developed through years of experimentation. As a retrieval model, however, it has major flaws. Although it provides a convenient computational framework, it provides little guidance on the details of how weighting and ranking algorithms are related to relevance.

In this model, documents and queries are assumed to be part of a t-dimensional vector space, where t is the number of index terms (words, stems, phrases, etc.). A document D_i is represented by a vector of index terms:

$$D_i = (d_{i1}, d_{i2}, \ldots, d_{it}),$$

where d_{ij} represents the weight of the jth term. A document collection containing n documents can be represented as a matrix of term weights, where each row represents a document and each column describes weights that were assigned to a term for a particular document:

$$\begin{array}{cccccc} & Term_1 & Term_2 & \ldots & Term_t \\ Doc_1 & d_{11} & d_{12} & \ldots & d_{1t} \\ Doc_2 & d_{21} & d_{22} & \ldots & d_{2t} \\ \vdots & & & & \\ Doc_n & d_{n1} & d_{n2} & \ldots & d_{nt} \end{array}$$

Figure 7.1 gives a simple example of the vector representation for four documents. The term-document matrix has been rotated so that now the terms are the rows and the documents are the columns. The term weights are simply the count of the terms in the document. Stopwords are not indexed in this example, and the words have been stemmed. Document D_3, for example, is represented by the vector $(1, 1, 0, 2, 0, 1, 0, 1, 0, 0, 1)$.

Queries are represented the same way as documents. That is, a query Q is represented by a vector of t weights:

$$Q = (q_1, q_2, \ldots, q_t),$$

where q_j is the weight of the jth term in the query. If, for example the query was "tropical fish", then using the vector representation in Figure 7.1, the query would be $(0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1)$. One of the appealing aspects of the vector space model is the use of simple diagrams to visualize the documents and queries. Typically, they are shown as points or vectors in a three-dimensional picture, as in

D_1	Tropical Freshwater Aquarium Fish.
D_2	Tropical Fish, Aquarium Care, Tank Setup.
D_3	Keeping Tropical Fish and Goldfish in Aquariums, and Fish Bowls.
D_4	The Tropical Tank Homepage - Tropical Fish and Aquariums.

Terms	Documents			
	D_1	D_2	D_3	D_4
aquarium	1	1	1	1
bowl	0	0	1	0
care	0	1	0	0
fish	1	1	2	1
freshwater	1	0	0	0
goldfish	0	0	1	0
homepage	0	0	0	1
keep	0	0	1	0
setup	0	1	0	0
tank	0	1	0	1
tropical	1	1	1	2

Fig. 7.1. Term-document matrix for a collection of four documents

Figure 7.2. Although this can be helpful for teaching, it is misleading to think that an intuition developed using three dimensions can be applied to the actual high-dimensional document space. Remember that the t terms represent all the document features that are indexed. In enterprise and web applications, this corresponds to hundreds of thousands or even *millions* of dimensions.

Given this representation, documents could be ranked by computing the distance between the points representing the documents and the query. More commonly, a *similarity measure* is used (rather than a distance or *dissimilarity* measure), so that the documents with the highest scores are the most similar to the query. A number of similarity measures have been proposed and tested for this purpose. The most successful of these is the *cosine correlation* similarity measure. The cosine correlation measures the cosine of the angle between the query and the document vectors. When the vectors are *normalized* so that all documents and queries are represented by vectors of equal length, the cosine of the angle between two identical vectors will be 1 (the angle is zero), and for two vectors that do not share any non-zero terms, the cosine will be 0. The cosine measure is defined as:

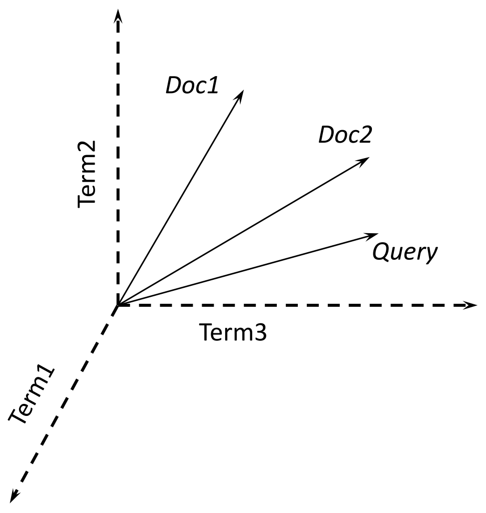

Fig. 7.2. Vector representation of documents and queries

$$Cosine(D_i, Q) = \frac{\sum_{j=1}^{t} d_{ij} \cdot q_j}{\sqrt{\sum_{j=1}^{t} {d_{ij}}^2 \cdot \sum_{j=1}^{t} {q_j}^2}}$$

The numerator of this measure is the sum of the products of the term weights for the matching query and document terms (known as the *dot product* or inner product). The denominator normalizes this score by dividing by the product of the lengths of the two vectors. There is no theoretical reason why the cosine correlation should be preferred to other similarity measures, but it does perform somewhat better in evaluations of search quality.

As an example, consider two documents $D_1 = (0.5, 0.8, 0.3)$ and $D_2 = (0.9, 0.4, 0.2)$ indexed by three terms, where the numbers represent term weights. Given the query $Q = (1.5, 1.0, 0)$ indexed by the same terms, the cosine measures for the two documents are:

$$
\begin{aligned}
Cosine(D_1, Q) &= \frac{(0.5 \times 1.5) + (0.8 \times 1.0)}{\sqrt{(0.5^2 + 0.8^2 + 0.3^2)(1.5^2 + 1.0^2)}} \\
&= \frac{1.55}{\sqrt{(0.98 \times 3.25)}} = 0.87
\end{aligned}
$$

$$
\begin{aligned}
Cosine(D_2, Q) &= \frac{(0.9 \times 1.5) + (0.4 \times 1.0)}{\sqrt{(0.9^2 + 0.4^2 + 0.2^2)(1.5^2 + 1.0^2)}} \\
&= \frac{1.75}{\sqrt{(1.01 \times 3.25)}} = 0.97
\end{aligned}
$$

The second document has a higher score because it has a high weight for the first term, which also has a high weight in the query. Even this simple example shows that ranking based on the vector space model is able to reflect term importance and the number of matching terms, which is not possible in Boolean retrieval.

In this discussion, we have yet to say anything about the form of the term weighting used in the vector space model. In fact, many different weighting schemes have been tried over the years. Most of these are variations on *tf.idf* weighting, which was described briefly in Chapter 2. The term frequency component, *tf*, reflects the importance of a term in a document D_i (or query). This is usually computed as a normalized count of the term occurrences in a document, for example by

$$tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} f_{ij}}$$

where tf_{ik} is the term frequency weight of term k in document D_i, and f_{ik} is the number of occurrences of term k in the document. In the vector space model, normalization is part of the cosine measure. A document collection can contain documents of many different lengths. Although normalization accounts for this to some degree, long documents can have many terms occurring once and others occurring hundreds of times. Retrieval experiments have shown that to reduce the impact of these frequent terms, it is effective to use the logarithm of the number of term occurrences in *tf* weights rather than the raw count.

The inverse document frequency component (idf) reflects the importance of the term in the collection of documents. The more documents that a term occurs in, the less *discriminating* the term is between documents and, consequently, the less useful it will be in retrieval. The typical form of this weight is

$$idf_k = \log \frac{N}{n_k}$$

where idf_k is the inverse document frequency weight for term k, N is the number of documents in the collection, and n_k is the number of documents in which term k occurs. The form of this weight was developed by intuition and experiment, although an argument can be made that idf measures the amount of information carried by the term, as defined in *information theory* (Robertson, 2004).

The effects of these two weights are combined by multiplying them (hence the name *tf.idf*). The reason for combining them this way is, once again, mostly empirical. Given this, the typical form of document term weighting in the vector space model is:

$$d_{ik} = \frac{(\log(f_{ik}) + 1) \cdot \log(N/n_k)}{\sqrt{\sum_{k=1}^{t} [(\log(f_{ik}) + 1.0) \cdot \log(N/n_k)]^2}}$$

The form of query term weighting is essentially the same. Adding 1 to the term frequency component ensures that terms with frequency 1 have a non-zero weight. Note that, in this model, term weights are computed only for terms that occur in the document (or query). Given that the cosine measure normalization is incorporated into the weights, the score for a document is computed using simply the dot product of the document and query vectors.

Although there is no explicit definition of relevance in the vector space model, there is an implicit assumption that relevance is related to the similarity of query and document vectors. In other words, documents "closer" to the query are more likely to be relevant. This is primarily a model of topical relevance, although features related to user relevance could be incorporated into the vector representation. No assumption is made about whether relevance is binary or multivalued.

In the last chapter we described relevance feedback, a technique for query modification based on user-identified relevant documents. This technique was first introduced using the vector space model. The well-known *Rocchio algorithm* (Rocchio, 1971) was based on the concept of an *optimal query*, which maximizes the difference between the average vector representing the relevant documents and the average vector representing the non-relevant documents. Given that only limited relevance information is typically available, the most common (and effective) form of the Rocchio algorithm modifies the initial weights in query vector Q to produce a new query Q' according to

$$q'_j = \alpha.q_j + \beta.\frac{1}{|Rel|}\sum_{D_i \in Rel} d_{ij} - \gamma.\frac{1}{|Nonrel|}\sum_{D_i \in Nonrel} d_{ij}$$

where q_j is the initial weight of query term j, Rel is the set of identified relevant documents, $Nonrel$ is the set of non-relevant documents, $|.|$ gives the size of a set, d_{ij} is the weight of the jth term in document i, and α, β, and γ are parameters that control the effect of each component. Previous studies have shown that the set of non-relevant documents is best approximated by all unseen documents (i.e., all documents not identified as relevant), and that reasonable values for the parameters are 8, 16, and 4 for α, β, and γ, respectively.

This formula modifies the query term weights by adding a component based on the average weight in the relevant documents and subtracting a component based on the average weight in the non-relevant documents. Query terms with weights that are negative are dropped. This results in a longer or expanded query because terms that occur frequently in the relevant documents but not in the original query will be added (i.e., they will have non-zero positive weights in the modified query). To restrict the amount of expansion, typically only a certain number (say, 50) of the terms with the highest average weights in the relevant documents will be added to the query.

7.2 Probabilistic Models

One of the features that a retrieval model should provide is a clear statement about the assumptions upon which it is based. The Boolean and vector space approaches make implicit assumptions about relevance and text representation that impact the design and effectiveness of ranking algorithms. The ideal situation would be to show that, given the assumptions, a ranking algorithm based on the retrieval model will achieve better effectiveness than any other approach. Such proofs are actually very hard to come by in information retrieval, since we are trying to formalize a complex human activity. The validity of a retrieval model generally has to be validated empirically, rather than theoretically.

One early theoretical statement about effectiveness, known as the *Probability Ranking Principle* (Robertson, 1977/1997), encouraged the development of probabilistic retrieval models, which are the dominant paradigm today. These models have achieved this status because *probability theory* is a strong foundation for representing and manipulating the uncertainty that is an inherent part of the information retrieval process. The Probability Ranking Principle, as originally stated, is as follows:

> If a reference retrieval system's[1] response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose, the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data.

Given some assumptions, such as that the relevance of a document to a query is independent of other documents, it is possible to show that this statement is true, in the sense that ranking by probability of relevance will maximize precision, which is the proportion of relevant documents, at any given rank (for example, in the top 10 documents). Unfortunately, the Probability Ranking Principle doesn't tell us *how* to calculate or estimate the probability of relevance. There are many probabilistic retrieval models, and each one proposes a different method for estimating this probability. Most of the rest of this chapter discusses some of the most important probabilistic models.

In this section, we start with a simple probabilistic model based on treating information retrieval as a classification problem. We then describe a popular and effective ranking algorithm that is based on this model.

7.2.1 Information Retrieval as Classification

In any retrieval model that assumes relevance is binary, there will be two sets of documents, the relevant documents and the non-relevant documents, for each query. Given a new document, the task of a search engine could be described as deciding whether the document belongs in the relevant set or the non-relevant[2] set. That is, the system should *classify* the document as relevant or non-relevant, and retrieve it if it is relevant.

Given some way of calculating the *probability* that the document is relevant and the probability that it is non-relevant, then it would seem reasonable to classify the document into the set that has the highest probability. In other words,

[1] A "reference retrieval system" would now be called a search engine.

[2] Note that we never talk about "irrelevant" documents in information retrieval; instead they are "non-relevant."

we would decide that a document D is relevant if $P(R|D) > P(NR|D)$, where $P(R|D)$ is a *conditional* probability representing the probability of relevance given the representation of that document, and $P(NR|D)$ is the conditional probability of non-relevance (Figure 7.3). This is known as the *Bayes Decision Rule*, and a system that classifies documents this way is called a *Bayes classifier*.

In Chapter 9, we discuss other applications of classification (such as spam filtering) and other classification techniques, but here we focus on the ranking algorithm that results from this probabilistic retrieval model based on classification.

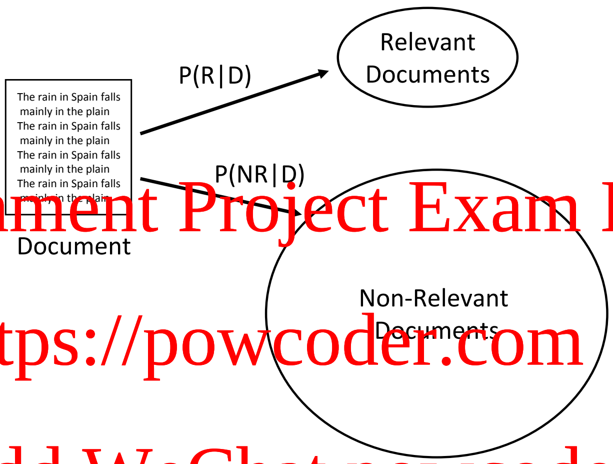
Relevant Documents
P(R|D)
The rain in Spain falls mainly in the plain The rain in Spain falls mainly in the plain The rain in Spain falls mainly in the plain The rain in Spain falls mainly in the plain
P(NR|D)
Document
Non-Relevant Documents

Fig. 7.3. Classifying a document as relevant or non-relevant

The question that faces us now is how to compute these probabilities. To start with, let's focus on $P(R|D)$. It's not clear how we would go about calculating this, but given information about the relevant set, we should be able to calculate $P(D|R)$. For example, if we had information about how often specific words occurred in the relevant set, then, given a new document, it would be relatively straightforward to calculate how likely it would be to see the combination of words in the document occurring in the relevant set. Let's assume that the probability of the word "president" in the relevant set is 0.02, and the probability of "lincoln" is 0.03. If a new document contains the words "president" and "lincoln", we could say that the probability of observing that combination of words in the relevant set is $0.02 \times 0.03 = 0.0006$, assuming that the two words occur independently.[3]

So how does calculating $P(D|R)$ get us to the probability of relevance? It turns out there is a relationship between $P(R|D)$ and $P(D|R)$ that is expressed by *Bayes' Rule*:[4]

$$P(R|D) = \frac{P(D|R)P(R)}{P(D)}$$

where $P(R)$ is the *a priori* probability of relevance (in other words, how likely any document is to be relevant), and $P(D)$ acts as a normalizing constant. Given this, we can express our decision rule in the following way: classify a document as relevant if $P(D|R)P(R) > P(D|NR)P(NR)$. This is the same as classifying a document as relevant if:

$$\frac{P(D|R)}{P(D|NR)} > \frac{P(NR)}{P(R)}$$

The left-hand side of this equation is known as the *likelihood ratio*. In most classification applications, such as spam filtering, the system must decide which class the document belongs to in order to take the appropriate action. For information retrieval, a search engine only needs to rank documents, rather than make that decision (which is hard). If we use the likelihood ratio as a score, the highly ranked documents will be those that have a high likelihood of belonging to the relevant set.

To calculate the document scores, we still need to decide how to come up with values for $P(D|R)$ and $P(D|NR)$. The simplest approach is to make the same assumptions that we made in our earlier example; that is, we represent documents as a combination of words and the relevant and non-relevant sets using word probabilities. In this model, documents are represented as a vector of binary features, $D = (d_1, d_2, \ldots, d_t)$, where $d_i = 1$ if term i is present in the document, and 0 otherwise. The other major assumption we make is *term independence* (also known as the *Naïve Bayes* assumption). This means we can estimate $P(D|R)$ by the product of the individual term probabilities $\prod_{i=1}^{t} P(d_i|R)$ (and similarly for $P(D|NR)$). Because this model makes the assumptions of term independence and binary features in documents, it is known as the *binary independence model*.

[3] Given two events A and B, the joint probability $P(A \cap B)$ is the probability of both events occurring together. In general, $P(A \cap B) = P(A|B)P(B)$. If A and B are independent, this means that $P(A \cap B) = P(A)P(B)$.

[4] Named after Thomas Bayes, a British mathematician.

Words obviously do not occur independently in text. If the word "Microsoft" occurs in a document, it is very likely that the word "Windows" will also occur. The assumption of term independence, however, is a common one since it usually simplifies the mathematics involved in the model. Models that allow some form of dependence between terms will be discussed later in this chapter.

Recall that a document in this model is a vector of 1s and 0s representing the presence and absence of terms. For example, if there were five terms indexed, one of the document representations might be $(1, 0, 0, 1, 1)$, meaning that the document contains terms 1, 4, and 5. To calculate the probability of this document occurring in the relevant set, we need the probabilities that the terms are 1 or 0 in the relevant set. If p_i is the probability that term i occurs (has the value 1) in a document from the relevant set, then the probability of our example document occurring in the relevant set is $p_1 \times (1 - p_2) \times (1 - p_3) \times p_4 \times p_5$. The probability $(1 - p_2)$ is the probability of term 2 *not* occurring in the relevant set. For the non-relevant set, we use s_i to represent the probability of term i occurring.[5]

Going back to the likelihood ratio, using p_i and s_i gives us a score of:

$$\frac{P(D|R)}{P(D|NR)} = \prod_{i:d_i=1} \frac{p_i}{s_i} \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i}$$

where $\prod_{i:d_i=1}$ means that it is a product over the terms that have the value 1 in the document. We can now do a bit of mathematical manipulation to get:

$$\prod_{i:d_i=1} \frac{p_i}{s_i} \cdot \left(\prod_{i:d_i=1} \frac{1-s_i}{1-p_i} \prod_{i:d_i=1} \frac{1-p_i}{1-s_i} \right) \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i}$$

$$= \prod_{i:d_i=1} \frac{p_i(1-s_i)}{s_i(1-p_i)} \cdot \prod_{i} \frac{1-p_i}{1-s_i}$$

The second product is over all terms and is therefore the same for all documents, so we can ignore it for ranking. Since multiplying lots of small numbers can lead to problems with the accuracy of the result, we can equivalently use the logarithm of the product, which means that the scoring function is:

$$\sum_{i:d_i=1} \log \frac{p_i(1-s_i)}{s_i(1-p_i)}$$

[5] In many descriptions of this model, p_i and q_i are used for these probabilities. We use s_i to avoid confusion with the q_i used to represent query terms.

You might be wondering where the query has gone, given that this is a document ranking algorithm for a specific query. In many cases, the query provides us with the only information we have about the relevant set. We can assume that, in the absence of other information, terms that are not in the query will have the same probability of occurrence in the relevant and non-relevant documents (i.e., $p_i = s_i$). In that case, the summation will only be over terms that are both in the query and in the document. This means that, given a query, the score for a document is simply the sum of the term weights for all matching terms.

If we have no other information about the relevant set, we could make the additional assumptions that p_i is a constant and that s_i could be estimated by using the term occurrences in the whole collection as an approximation. We make the second assumption based on the fact that the number of relevant documents is much smaller than the total number of documents in the collection. With a value of 0.5 for p_i in the scoring function described earlier, this gives a term weight for term i of

$$\log \frac{0.5(1 - \frac{n_i}{N})}{\frac{n_i}{N}(1 - 0.5)} = \log \frac{N - n_i}{n_i}$$

where n_i is the number of documents that contain term i, and N is the number of documents in the collection. This shows that, in the absence of information about the relevant documents, the term weight derived from the binary independence model is very similar to an idf weight. There is no tf component, because the documents were assumed to have binary features.

If we do have information about term occurrences in the relevant and non-relevant sets, it can be summarized in a *contingency table*, shown in Table 7.1. This information could be obtained through relevance feedback, where users identify relevant documents in initial rankings. In this table, r_i is the number of relevant documents containing term i, n_i is the number of documents containing term i, N is the total number of documents in the collection, and R is the number of relevant documents for this query.

	Relevant	Non-relevant	Total
$d_i = 1$	r_i	$n_i - r_i$	n_i
$d_i = 0$	$R - r_i$	$N - n_i - R + r_i$	$N - n_i$
Total	R	$N - R$	N

Table 7.1. Contingency table of term occurrences for a particular query

Given this table, the obvious *estimates*[6] for p_i and s_i would be $p_i = r_i/R$ (the number of relevant documents that contain a term divided by the total number of relevant documents) and $s_i = (n_i - r_i)/(N - R)$ (the number of non-relevant documents that contain a term divided by the total number of non-relevant documents). Using these estimates could cause a problem, however, if some of the entries in the contingency table were zeros. If r_i was zero, for example, the term weight would be $\log 0$. To avoid this, a standard solution is to add 0.5 to each count (and 1 to the totals), which gives us estimates of $p_i = (r_i + 0.5)/(R + 1)$ and $s_i = (n_i - r_i + 0.5)/(N - R + 1.0)$. Putting these estimates into the scoring function gives us:

$$\sum_{i:d_i=q_i=1} \log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)}$$

Although this document score sums term weights for just the matching query terms, with relevance feedback the query can be *expanded* to include other important terms from the relevant set. Note that if we have no relevance information, we can set r and R to 0, which would give a p_i value of 0.5, and would produce the *idf*-like term weight discussed before.

So how good is this document score when used for ranking? Not very good, it turns out. Although it does provide a method of incorporating relevance information, in most cases we don't have this information and instead would be using term weights that are similar to *idf* weights. The absence of a tf component makes a significant difference to the effectiveness of the ranking, and most effectiveness measures will drop by about 50% if the ranking ignores this information. This means, for example, that we might see 50% fewer relevant documents in the top ranks if we used the binary independence model ranking instead of the best $tf.idf$ ranking.

It turns out, however, that the binary independence model is the basis for one of the most effective and popular ranking algorithms, known as BM25.[7]

[6] We use the term *estimate* for a probability value calculated using data such as a contingency table because this value is only an estimate for the true value of the probability and would change if more data were available.

[7] BM stands for Best Match, and 25 is just a numbering scheme used by Robertson and his co-workers to keep track of weighting variants (Robertson & Walker, 1994).

7.2.2 The BM25 Ranking Algorithm

BM25 extends the scoring function for the binary independence model to include document and query term weights. The extension is based on probabilistic arguments and experimental validation, but it is not a formal model.

BM25 has performed very well in TREC retrieval experiments and has influenced the ranking algorithms of commercial search engines, including web search engines. There are some variations of the scoring function for BM25, but the most common form is:

$$\sum_{i \in Q} \log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} \cdot \frac{(k_1 + 1)f_i}{K + f_i} \cdot \frac{(k_2 + 1)qf_i}{k_2 + qf_i}$$

where the summation is now over all terms in the query; and N, R, n_i, and r_i are the same as described in the last section, with the additional condition that r and R are set to zero if there is no relevance information; f_i is the frequency of term i in the document; qf_i is the frequency of term i in the query; and k_1, k_2, and K are parameters whose values are set empirically.

The constant k_1 determines how the tf component of the term weight changes as f_i increases. If $k_1 = 0$, the term frequency component would be ignored and only term presence or absence would matter. If k_1 is large, the term weight component would increase nearly linearly with f_i. In TREC experiments, a typical value for k_1 is 1.2, which causes the effect of f_i to be very non-linear, similar to the use of $\log f$ in the term weights discussed in section 7.1.2. This means that after three or four occurrences of a term, additional occurrences will have little impact. The constant k_2 has a similar role in the query term weight. Typical values for this parameter are in the range 0 to 1,000, meaning that performance is less sensitive to k_2 than it is to k_1. This is because query term frequencies are much lower and less variable than document term frequencies.

K is a more complicated parameter that normalizes the *tf* component by document length. Specifically

$$K = k_1((1 - b) + b \cdot \frac{dl}{avdl})$$

where b is a parameter, dl is the length of the document, and $avdl$ is the average length of a document in the collection. The constant b regulates the impact of the length normalization, where $b = 0$ corresponds to no length normalization, and

$b = 1$ is full normalization. In TREC experiments, a value of $b = 0.75$ was found to be effective.

As an example calculation, let's consider a query with two terms, "president" and "lincoln", each of which occurs only once in the query ($qf = 1$). We will consider the typical case where we have no relevance information (r and R are zero). Let's assume that we are searching a collection of 500,000 documents (N), and that in this collection, "president" occurs in 40,000 documents ($n_1 = 40,000$) and "lincoln" occurs in 300 documents ($n_2 = 300$). In the document we are scoring (which is about President Lincoln), "president" occurs 15 times ($f_1 = 15$) and "lincoln" occurs 25 times ($f_2 = 25$). The document length is 90% of the average length ($dl/avdl = 0.9$). The parameter values we use are $k_1 = 1.2, b = 0.75$, and $k_2 = 100$. With these values, $K = 1.2 \cdot (0.25 + 0.75 \cdot 0.9) = 1.11$, and the document score is:

$$
\begin{aligned}
BM25(Q, D) = & \\
& \log \frac{(0+0.5)/(0-0+0.5)}{(40000-0+0.5)/(500000-40000-0+0+0.5)} \\
& \times \frac{(1.2+1)15}{1.11+15} \times \frac{(100+1)1}{100+1} \\
& + \log \frac{(0+0.5)/(0-0+0.5)}{(300-0+0.5)/(500000-300-0+0+0.5)} \\
& \times \frac{(1.2+1)25}{1.11+25} \times \frac{(100+1)1}{100+1}
\end{aligned}
$$

$$
\begin{aligned}
&= \log 460000.5/40000.5 \cdot 33/16.11 \cdot 101/101 \\
&\quad + \log 499700.5/300.5 \cdot 55/26.11 \cdot 101/101 \\
&= 2.44 \cdot 2.05 \cdot 1 + 7.42 \cdot 2.11 \cdot 1 \\
&= 5.00 + 15.66 = 20.66
\end{aligned}
$$

Notice the impact from the first part of the weight that, without relevance information, is nearly the same as an idf weight (as we discussed in section 7.2.1). Because the term "lincoln" is much less frequent in the collection, it has a much higher idf component (7.42 versus 2.44). Table 7.2 gives scores for different numbers of term occurrences. This shows the importance of the "lincoln" term and that even one occurrence of a term can make a large difference in the score. Reducing the number of term occurrences from 25 or 15 to 1 makes a significant but not dramatic difference. This example also demonstrates that it is possible for a document containing a large number of occurrences of a single important term to score higher than a document containing both query terms (15.66 versus 12.74).

Frequency of "president"	Frequency of "lincoln"	BM25 score
15	25	20.66
15	1	12.74
15	0	5.00
1	25	18.2
0	25	15.66

Table 7.2. BM25 scores for an example document

The score calculation may seem complicated, but remember that some of the calculation of term weights can occur at indexing time, before processing any query. If there is no relevance information, scoring a document simply involves adding the weights for matching query terms, with a small additional calculation if query terms occur more than once (i.e., if $qf > 1$). Another important point is that the parameter values for the BM25 ranking algorithm can be tuned (i.e., adjusted to obtain the best effectiveness) for each application. The process of tuning is described further in section 7.7 and Chapter 8.

To summarize, BM25 is an effective ranking algorithm derived from a model of information retrieval viewed as classification. This model focuses on topical relevance and makes an explicit assumption that relevance is binary. In the next section, we discuss another probabilistic model that incorporates term frequency directly in the model, rather than being added in as an extension to improve performance.

7.3 Ranking Based on Language Models

Language models are used to represent text in a variety of *language technologies*, such as speech recognition, machine translation, and handwriting recognition. The simplest form of language model, known as a *unigram* language model, is a *probability distribution* over the words in the language. This means that the language model associates a probability of occurrence with every word in the index vocabulary for a collection. For example, if the documents in a collection contained just five different words, a possible language model for that collection might be $(0.2, 0.1, 0.35, 0.25, 0.1)$, where each number is the probability of a word occurring. If we treat each document as a *sequence* of words, then the probabilities in the language model predict what the next word in the sequence will be. For example, if the five words in our language were "girl", "cat", "the", "boy", and "touched", then the probabilities predict which of these words will be next. These words cover all the possibilities, so the probabilities must add to 1. Because this is a unigram model, the previous words have no impact on the prediction. With this model, for example, it is just as likely to get the sequence "girl cat" (probability 0.2×0.1) as "girl touched" (probability 0.2×0.1).

In applications such as speech recognition, n-gram language models that predict words based on longer sequences are used. An n-gram model predicts a word based on the previous $n - 1$ words. The most common n-gram models are *bigram* (predicting based on the previous word) and *trigram* (predicting based on the previous two words) models. Although bigram models have been used in information retrieval to represent two-word phrases (see section 4.3.5), we focus our discussion on unigram models because they are simpler and have proven to be very effective as the basis for ranking algorithms.

For search applications, we use language models to represent the topical content of a document. A *topic* is something that is talked about often but rarely defined in information retrieval discussions. In this approach, we define a topic as a probability distribution over words (in other words, a language model). For example, if a document is about fishing in Alaska, we would expect to see words associated with fishing and locations in Alaska with high probabilities in the language model. If it is about fishing in Florida, some of the high-probability words will be the same, but there will be more high probability words associated with locations in Florida. If instead the document is about fishing games for computers, most of the high-probability words will be associated with game manufacturers and computer use, although there will still be some important words about fishing. Note that a topic language model, or *topic model* for short, contains probabilities for all words, not just the most important. Most of the words will have "default" probabilities that will be the same for any text, but the words that are important for the topic will have unusually high probabilities.

A language model representation of a document can be used to "generate" new text by sampling words according to the probability distribution. If we imagine the language model as a big bucket of words, where the probabilities determine how many instances of a word are in the bucket, then we can generate text by reaching in (without looking), drawing out a word, writing it down, putting the word back in the bucket, and drawing again. Note that we are not saying that we can generate the original document by this process. In fact, because we are only using a unigram model, the generated text is going to look pretty bad, with no syntactic structure. Important words for the topic of the document will, however, appear often. Intuitively, we are using the language model as a very approximate model for the topic the author of the document was thinking about when he was writing it.

When text is modeled as a finite sequence of words, where at each point in the sequence there are t different possible words, this corresponds to assuming a *multinomial* distribution over words. Although there are alternatives, multinomial language models are the most common in information retrieval.[8] One of the limitations of multinomial models that has been pointed out is that they do not describe text *burstiness* well, which is the observation that once a word is "pulled out of the bucket," it tends to be pulled out repeatedly.

In addition to representing documents as language models, we can also represent the topic of the query as a language model. In this case, the intuition is that the language model is a representation of the topic that the information seeker had in mind when she was writing the query. This leads to three obvious possibilities for retrieval models based on language models: one based on the probability of generating the query text from a document language model, one based on generating the document text from a query language model, and one based on comparing the language models representing the query and document topics. In the next two sections, we describe these retrieval models in more detail.

7.3.1 Query Likelihood Ranking

In the *query likelihood* retrieval model, we rank documents by the probability that the query text could be generated by the document language model. In other words, we calculate the probability that we could pull the query words out of the "bucket" of words representing the document. This is a model of topical relevance, in the sense that the probability of query generation is the measure of how likely it is that a document is about the same topic as the query.

Since we start with a query, we would in general like to calculate $P(D|Q)$ to rank the documents. Using Bayes' Rule, we can calculate this by

[8] We discuss the multinomial model in the context of classification in Chapter 9.

$$p(D|Q) \overset{rank}{=} P(Q|D)P(D)$$

where the symbol $\overset{rank}{=}$, as we mentioned previously, means that the right-hand side is rank equivalent to the left-hand side (i.e., we can ignore the normalizing constant $P(Q)$), $P(D)$ is the prior probability of a document, and $P(Q|D)$ is the query likelihood given the document. In most cases, $P(D)$ is assumed to be *uniform* (the same for all documents), and so will not affect the ranking. Models that assign non-uniform prior probabilities based on, for example, document date or document length can be useful in some applications, but we will make the simpler uniform assumption here. Given that assumption, the retrieval model specifies ranking documents by $P(Q|D)$, which we calculate using the unigram language model for the document

$$P(Q|D) = \prod_{i=1}^{n} P(q_i|D)$$

where q_i is a query word, and there are n words in the query.

To calculate this score, we need to have estimates for the language model probabilities $P(q_i|D)$. The obvious estimate would be

$$P(q_i|D) = \frac{f_{q_i,D}}{|D|}$$

where $f_{q_i,D}$ is the number of times word qi occurs in document D, and $|D|$ is the number of words in D. For a multinomial distribution, this is the *maximum likelihood* estimate, which means this this is the estimate that makes the observed value of $f_{q_i,D}$ most likely. The major problem with this estimate is that if any of the query words are missing from the document, the score given by the query likelihood model for $P(Q|D)$ will be zero. This is clearly not appropriate for longer queries. For example, missing one word out of six should not produce a score of zero. We will also not be able to distinguish between documents that have different numbers of query words missing. Additionally, because we are building a topic model for a document, words associated with that topic should have some probability of occurring, even if they were not mentioned in the document. For example, a language model representing a document about computer games should have some non-zero probability for the word "RPG" even if that word was not mentioned in the document. A small probability for that word will enable the document to receive a non-zero score for the query "RPG computer games", although it will be lower than the score for a document that contains all three words.

Smoothing is a technique for avoiding this estimation problem and overcoming *data sparsity*, which means that we typically do not have large amounts of text to use for the language model probability estimates. The general approach to smoothing is to lower (or *discount*) the probability estimates for words that are seen in the document text, and assign that "leftover" probability to the estimates for the words that are not seen in the text. The estimates for unseen words are usually based on the frequency of occurrence of words in the whole document collection. If $P(q_i|C)$ is the probability for query word i in the *collection language model* for document collection C, then the estimate we use for an unseen word in a document is $\alpha_D P(q_i|C)$, where α_D is a coefficient controlling the probability assigned to unseen words.[9] In general, α_D can depend on the document. In order that the probabilities sum to one, the probability estimate for a word that is seen in a document is $(1 - \alpha_D)P(q_i|D) + \alpha_D P(q_i|C)$.

To make this clear, consider a simple example where there are only three words, w_1, w_2, and w_3, in our index vocabulary. If the collection probabilities for these three words, based on maximum likelihood estimates, are 0.3, 0.5, and 0.2, and the document probabilities based on maximum likelihood estimates are 0.5, 0.5, and 0.0, then the *smoothed* probability estimates for the document language model are:

$$
\begin{aligned}
P(w_1|D) &= (1 - \alpha_D)P(w_1|D) + \alpha_D P(w_1|C) \\
&= (1 - \alpha_D) \cdot 0.5 + \alpha_D \cdot 0.3 \\
P(w_2|D) &= (1 - \alpha_D) \cdot 0.5 + \alpha_D \cdot 0.5 \\
P(w_3|D) &= (1 - \alpha_D) \cdot 0.0 + \alpha_D \cdot 0.2 = \alpha_D \cdot 0.2
\end{aligned}
$$

Note that term w_3 has a non-zero probability estimate, even though it did not occur in the document text. If we add these three probabilities, we get

$$
\begin{aligned}
P(w_1|D) + P(w_2|D) + P(w_3|D) &= (1 - \alpha_D) \cdot (0.5 + 0.5) \\
&\quad + \alpha_D \cdot (0.3 + 0.5 + 0.2) \\
&= 1 - \alpha_D + \alpha_D \\
&= 1
\end{aligned}
$$

which confirms that the probabilities are consistent.

[9] The collection language model probability is also known as the *background* language model probability, or just the background probability.

Different forms of estimation result from specifying the value of α_D. The simplest choice would be to set it to a constant, i.e., $\alpha_D = \lambda$. The collection language model probability estimate we use for word q_i is $c_{q_i}/|C|$, where c_{q_i} is the number of times a query word occurs in the collection of documents, and $|C|$ is the total number of word occurrences in the collection. This gives us an estimate for $P(q_i|D)$ of:

$$p(q_i|D) = (1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|}$$

This form of smoothing is known as the *Jelinek-Mercer* method. Substituting this estimate in the document score for the query-likelihood model gives:

$$P(Q|D) = \prod_{i=1}^{n}((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})$$

As we have said before, since multiplying many small numbers together can lead to accuracy problems, we can use logarithms to turn this score into a rank-equivalent sum as follows:

$$\log P(Q|D) = \sum_{i=1}^{n}\log((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})$$

Small values of λ produce less smoothing, and consequently the query tends to act more like a Boolean AND since the absence of any query word will penalize the score substantially. In addition, the relative weighting of words, as measured by the maximum likelihood estimates, will be important in determining the score. As λ approaches 1, the relative weighting will be less important, and the query acts more like a Boolean OR or a *coordination level match*.[10] In TREC evaluations, it has been shown that values of λ around 0.1 work well for short queries, whereas values around 0.7 are better for much longer queries. Short queries tend to contain only significant words, and a low λ value will favor documents that contain all the query words. With much longer queries, missing a word is much less important, and a high λ places more emphasis on documents that contain a number of the high-probability words.

At this point, it may occur to you that the query likelihood retrieval model doesn't have anything that looks like a *tf.idf* weight, and yet experiments show

[10] A coordination level match simply ranks documents by the number of matching query terms.

that it is as least as effective as the BM25 ranking algorithm. We can, however, demonstrate a relationship to *tf.idf* weights by manipulating the query likelihood score in the following way:

$$
\begin{aligned}
\log P(Q|D) &= \sum_{i=1}^{n} \log((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|}) \\
&= \sum_{i:f_{q_i,D}>0} \log((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|}) + \sum_{i:f_{q_i,D}=0} \log(\lambda\frac{c_{q_i}}{|C|}) \\
&= \sum_{i:f_{q_i,D}>0} \log \frac{((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})}{\lambda\frac{c_{q_i}}{|C|}} + \sum_{i=1}^{n} \log(\lambda\frac{c_{q_i}}{|C|}) \\
&\overset{rank}{=} \sum_{i:f_{q_i,D}>0} \log\left(\frac{((1-\lambda)\frac{f_{q_i,D}}{|D|}}{\lambda\frac{c_{q_i}}{|C|}} + 1\right)
\end{aligned}
$$

In the second line, we split the score into the words that occur in the document and those that don't occur ($f_{q_i,D} = 0$). In the third line, we add

$$\sum_{i:f_{q_i,D}>0} \log(\lambda\frac{c_{q_i}}{|C|})$$

to the last term and subtract it from the first (where it ends up in the denominator), so there is no net effect. The last term is now the same for all documents and can be ignored for ranking. The final expression gives the document score in terms of a "weight" for matching query terms. Although this weight is not identical to a *tf.idf* weight, there are clear similarities in that it is directly proportional to the document term frequency and inversely proportional to the collection frequency.

A different form of estimation, and one that is generally more effective, comes from using a value of α_D that is dependent on document length. This approach is known as *Dirichlet* smoothing, for reasons we will discuss later, and uses

$$\alpha_D = \frac{\mu}{|D| + \mu}$$

where μ is a parameter whose value is set empirically. Substituting this expression for α_D in $(1 - \alpha_D)P(q_i|D) + \alpha_D P(q_i|C)$ results in the probability estimation formula

$$p(q_i|D) = \frac{f_{q_i,D} + \mu\frac{c_{q_i}}{|C|}}{|D| + \mu}$$

which in turn leads to the following document score:

$$\log P(Q|D) = \sum_{i=1}^{n} \log \frac{f_{q_i,D} + \mu\frac{c_{q_i}}{|C|}}{|D| + \mu}$$

Similar to the Jelinek-Mercer smoothing, small values of the parameter (μ in this case) give more importance to the relative weighting of words, and large values favor the number of matching terms. Typical values of μ that achieve the best results in TREC experiments are in the range 1,000 to 2,000 (remember that collection probabilities are very small), and Dirichlet smoothing is generally more effective than Jelinek-Mercer, especially for the short queries that are common in most search applications.

So where does Dirichlet smoothing come from? It turns out that a Dirichlet distribution[11] is the natural way to specify prior knowledge when estimating the probabilities in a multinomial distribution. The process of *Bayesian estimation* determines probability estimates based on this prior knowledge and the observed text. The resulting probability estimate can be viewed as combining actual word counts from the text with *pseudo-counts* from the Dirichlet distribution. If we had no text, the probability estimate for term q_i would be $\mu(c_{q_i}/|C|)/\mu$, which is a reasonable guess based on the collection. The more text we have (i.e., for longer documents), the less influence the prior knowledge will have.

We can demonstrate the calculation of query likelihood document scores using the example given in section 7.2.2. The two query terms are "president" and "lincoln". For the term "president", $f_{q_i,D}$ = 15, and let's assume that c_{q_i} = 160,000. For the term "lincoln", $f_{q_i,D}$ = 25, and we will assume that c_{q_i} = 2,400. The number of word occurrences in the document $|d|$ is assumed to be 1,800, and the number of word occurrences in the collection is 10^9 (500,000 documents times an average of 2,000 words). The value of μ used is 2,000. Given these numbers, the score for the document is:

[11] Named after the German mathematician Johann Peter Gustav Lejeune Dirichlet (the first name used seems to vary).

$$
\begin{aligned}
QL(Q,D) &= \log \frac{15 + 2000 \times (1.6 \times 10^5/10^9)}{1800 + 2000} \\
&\quad + \log \frac{25 + 2000 \times (2400/10^9)}{1800 + 2000} \\
&= \log(15.32/3800) + \log(25.005/3800) \\
&= -5.51 + -5.02 = -10.53
\end{aligned}
$$

A negative number? Remember that we are taking logarithms of probabilities in this scoring function, and the probabilities of word occurrence are small. The important issue is the effectiveness of the rankings produced using these scores. Table 7.3 shows the query likelihood scores for the same variations of term occurrences that were used in Table 7.2. Although the scores look very different for BM25 and QL, the rankings are similar, with the exception that the document containing 15 occurrences of "president" and 1 of "lincoln" is ranked higher than the document containing 0 occurrences of "president" and 25 occurrences of "lincoln" in the QL scores, whereas the reverse is true for BM25.

Frequency of "president"	Frequency of "lincoln"	QL score
15	25	−10.53
15	1	−13.75
15	0	−19.05
1	25	−12.99
0	25	−14.40

Table 7.3. Query likelihood scores for an example document

To summarize, query likelihood is a simple probabilistic retrieval model that directly incorporates term frequency. The problem of coming up with effective term weights is replaced by probability estimation, which is better understood and has a formal basis. The basic query likelihood score with Dirichlet smoothing has similar effectiveness to BM25, although it does do better on most TREC collections. If more sophisticated smoothing based on topic models is used (described further in section 7.6), query likelihood consistently outperforms BM25. This means that instead of smoothing using the collection probabilities for words, we instead use word probabilities from similar documents.

The simplicity of the language model framework, combined with the ability to describe a variety of retrieval applications and the effectiveness of the associated ranking algorithms, make this approach a good choice for a retrieval model based on topical relevance.

7.3.2 Relevance Models and Pseudo-Relevance Feedback

Although the basic query likelihood model has a number of advantages, it is limited in terms of how it models information needs and queries. It is difficult, for example, to incorporate information about relevant documents into the ranking algorithm, or to represent the fact that a query is just one of many possible queries that could be used to describe a particular information need. In this section, we show how this can be done by extending the basic model.

In the introduction to section 7.3, we mentioned that it is possible to represent the topic of a query as a language model. Instead of calling this the query language model, we use the name *relevance model* since it represents the topic covered by relevant documents. The query can be viewed as a very small sample of text generated from the relevance model, and relevant documents are much larger samples of text from the same model. Given some examples of relevant documents for a query, we could estimate the probabilities in the relevance model and then use this model to predict the relevance of new documents. In fact, this is a version of the classification model presented in section 7.2.1, where we interpret $P(D|R)$ as the probability of generating the text in a document given a relevance model. This is also called the *document likelihood* model. Although this model, unlike the binary independence model, directly incorporates term frequency, it turns out that $P(D|R)$ is difficult to calculate and compare across documents. This is because documents contain a large and extremely variable number of words compared to a query. Consider two documents D_a and D_b, for example, containing 5 and 500 words respectively. Because of the large difference in the number of words involved, the comparison of $P(D_a|R)$ and $P(D_b|R)$ for ranking will be more difficult than comparing $P(Q|D_a)$ and $P(Q|D_b)$, which use the same query and smoothed representations for the documents. In addition, we still have the problem of obtaining examples of relevant documents.

There is, however, another alternative. If we can estimate a relevance model from a query, we can compare this language model directly with the model for a document. Documents would then be ranked by the similarity of the document model to the relevance model. A document with a model that is very similar to the relevance model is likely to be on the same topic. The obvious next question is how to compare two language models. A well-known measure from probability theory and information theory, the *Kullback-Leibler divergence* (referred to as

KL-divergence in this book),[12] measures the difference between two probability distributions. Given the *true* probability distribution P and another distribution Q that is an approximation to P, the KL divergence is defined as:

$$KL(P||Q) = \sum_x P(x)\log\frac{P(x)}{Q(x)}$$

Since KL-divergence is always positive and is larger for distributions that are further apart, we use the *negative* KL-divergence as the basis for the ranking function (i.e., smaller differences mean higher scores). In addition, KL-divergence is not symmetric, and it matters which distribution we pick as the true distribution. If we assume the true distribution to be the relevance model for the query (R) and the approximation to be the document language model (D), then the negative KL-divergence can be expressed as

$$\sum_{w\in V} P(w|R)\log P(w|D) - \sum_{w\in V} P(w|R)\log P(w|R)$$

where the summation is over all words w in the vocabulary V. The second term on the right-hand side of this equation does not depend on the document, and can be ignored for ranking. Given a simple maximum likelihood estimate for $P(w|R)$, based on the frequency in the query text ($f_{w,Q}$) and the number of words in the query ($|Q|$), the score for a document will be:

$$\sum_{w\in V}\frac{f_{w,Q}}{|Q|}\log P(w|D)$$

Although this summation is over all words in the vocabulary, words that do not occur in the query have a zero maximum likelihood estimate and will not contribute to the score. Also, query words with frequency k will contribute $k \times \log P(w|D)$ to the score. This means that this score is rank equivalent to the query likelihood score described in the previous section. In other words, query likelihood is a special case of a retrieval model that ranks by comparing a relevance model based on a query to a document language model.

The advantage of the more general model is that it is not restricted to the simple method of estimating the relevance model using query term frequencies. If we

[12] KL-divergence is also called information divergence, information gain, or relative entropy.

regard the query words as a sample from the relevance model, then it seems reasonable to base the probability of a new sample word on the query words we have seen. In other words, the probability of pulling a word w out of the "bucket" representing the relevance model should depend on the n query words we have just pulled out. More formally, we can relate the probability of w to the conditional probability of observing w given that we just observed the query words $q_1 \ldots q_n$ by the approximation:

$$P(w|R) \approx P(w|q_1 \ldots q_n)$$

By definition, we can express the conditional probability in terms of the joint probability of observing w with the query words:

$$P(w|R) \approx \frac{P(w, q_1 \ldots q_n)}{P(q_1 \ldots q_n)}$$

$P(q_1 \ldots q_n)$ is a normalizing constant and is calculated as:

$$P(q_1 \ldots q_n) = \sum_{w \in V} P(w, q_1 \ldots q_n)$$

Now the question is how to estimate the joint probability $P(w, q_1 \ldots q_n)$. Given a set of documents $\mathcal{C}$ represented by language models, we can calculate the joint probability as follows:

$$P(w, q_1 \ldots q_n) = \sum_{D \in \mathcal{C}} p(D) P(w, q_1 \ldots q_n|D)$$

We can also make the assumption that:

$$P(w, q_1 \ldots q_n|D) = P(w|D) \prod_{i=1}^{n} P(q_i|D)$$

When we substitute this expression for $P(w, q_1 \ldots q_n|D)$ into the previous equation, we get the following estimate for the joint probability:

$$P(w, q_1 \ldots q_n) = \sum_{D \in \mathcal{C}} P(D) P(w|D) \prod_{i=1}^{n} P(q_i|D)$$

How do we interpret this formula? The prior probability $P(D)$ is usually assumed to be uniform and can be ignored. The expression $\prod_{i=1}^{n} P(q_i|D)$ is, in fact, the query likelihood score for the document D. This means that the estimate for $P(w, q_1 \ldots q_n)$ is simply a weighted average of the language model probabilities for w in a set of documents, where the weights are the query likelihood scores for those documents.

Ranking based on relevance models actually requires two passes. The first pass ranks documents using query likelihood to obtain the weights that are needed for relevance model estimation. In the second pass, we use KL-divergence to rank documents by comparing the relevance model and the document model. Note also that we are in effect adding words to the query by smoothing the relevance model using documents that are similar to the query. Many words that had zero probabilities in the relevance model based on query frequency estimates will now have non-zero values. What we are describing here is *exactly* the pseudo-relevance feedback process described in section 6.2.4. In other words, relevance models provide a formal retrieval model for pseudo-relevance feedback and query expansion. The following is a summary of the steps involved in ranking using relevance models:

1. Rank documents using the query likelihood score for query Q.
2. Select some number of the top-ranked documents to be the set $\mathcal{C}$.
3. Calculate the relevance model probabilities $P(w|R)$ using the estimate for $P(w, q_1 \ldots q_n)$.
4. Rank documents again using the KL-divergence score:[13]

$$\sum_w P(w|R) \log P(w|D)$$

Some of these steps require further explanation. In steps 1 and 4, the document language model probabilities ($P(w|D)$) should be estimated using Dirichlet smoothing. In step 2, the model allows the set $\mathcal{C}$ to be the whole collection, but because low-ranked documents have little effect on the estimation of $P(w|R)$, usually only 10–50 of the top-ranked documents are used. This also makes the computation of $P(w|R)$ substantially faster.

For similar reasons, the summation in step 4 is not done over all words in the vocabulary. Typically only a small number (10–25) of the highest-probability words are used. In addition, the importance of the original query words is emphasized by combining the original query frequency estimates with the relevance

[13] More accurately, this score is the negative *cross entropy* because we removed the term $\sum_{w \in V} P(w|R) \log P(w|R)$.

model estimates using a similar approach to Jelinek-Mercer, i.e., $\lambda P(w|Q) + (1 - \lambda)P(w|R)$, where λ is a mixture parameter whose value is determined empirically (0.5 is a typical value for TREC experiments). This combination makes it clear that estimating relevance models is basically a process for query expansion and smoothing.

The next important question, as for all retrieval models, is how well it works. Based on TREC experiments, ranking using relevance models is one of the best pseudo-relevance feedback techniques. In addition, relevance models produce a significant improvement in effectiveness compared to query likelihood ranking averaged over a number of queries. Like all current pseudo-relevance feedback techniques, however, the improvements are not consistent, and some queries can produce worse rankings or strange results.

Tables 7.4 and 7.5 show the 16 highest-probability words from relevance models estimated using this technique with some example queries and a large collection of TREC news stories from the 1990s.[14] Table 7.4 uses the top 10 documents from the query likelihood ranking to construct the relevance model, whereas Table 7.5 uses the top 50 documents.

The first thing to notice is that, although the words are reasonable, they are very dependent on the collection of documents that is used. In the TREC news collection, for example, many of the stories that mention Abraham Lincoln are on the topic of the Lincoln Bedroom in the White House, which President Clinton used for guests and President Lincoln used as an office during the Civil War. These types of stories are reflected in the top probability words for the queries "president lincoln" and "abraham lincoln". Expanding the query using these words would clearly favor the retrieval of this type of story rather than more general biographies of Lincoln. The second observation is that there is not much difference between the words based on 10 documents and the words based on 50 documents. The words based on 50 documents are, however, somewhat more general because the larger set of documents contains a greater variety of topics. In the case of the query "tropical fish", the relevance model words based on 10 documents are clearly more related to the topic.

In summary, ranking by comparing a model of the query to a model of the document using KL-divergence is a generalization of query likelihood scoring.

[14] This is a considerably larger collection than was used to generate the term association tables in Chapter 6. Those tables were based on the ROBUST track data, which consists of just over half a million documents. These tables were generated using all the TREC news collections, which total more than six million documents.

president lincoln	*abraham lincoln*	*fishing*	*tropical fish*
lincoln	lincoln	fish	fish
president	america	farm	tropic
room	president	salmon	japan
bedroom	faith	new	aquarium
house	guest	wild	water
white	abraham	water	species
america	new	caught	aquatic
guest	room	catch	fair
serve	christian	tag	china
bed	history	time	coral
washington	public	eat	source
old	bedroom	raise	tank
office	war	city	reef
war	politics	people	animal
long	old	fishermen	tarpon
abraham	national	boat	fishery

Table 7.4. Highest-probability terms from relevance model for four example queries (estimated using top 10 documents)

This generalization allows for more accurate queries that reflect the relative importance of the words in the topic that the information seeker had in mind when he was writing the query. Relevance model estimation is an effective pseudo-relevance feedback technique based on the formal framework of language models, but as with all these techniques, caution must be used in applying relevance model–based query expansion to a specific retrieval application.

Language models provide a formal but straightforward method of describing retrieval models based on topical relevance. Even more sophisticated models can be developed by incorporating term dependence and phrases, for example. Topical relevance is, however, only part of what is needed for effective search. In the next section, we focus on a retrieval model for combining all the pieces of evidence that contribute to user relevance, which is what people who use a search engine really care about.

president lincoln	*abraham lincoln*	*fishing*	*tropical fish*
lincoln	lincoln	fish	fish
president	president	water	tropic
america	america	catch	water
new	abraham	reef	storm
national	war	fishermen	species
great	man	river	boat
white	civil	new	sea
war	new	year	river
washington	history	time	country
clinton	two	bass	tuna
house	room	boat	world
history	booth	world	million
time	time	farm	state
center	politics	angle	time
kennedy	public	fly	japan
room	guest	trout	mile

Table 7.5. Highest-probability terms from relevance model for four example queries (estimated using top 50 documents)

7.4 Complex Queries and Combining Evidence

Effective retrieval requires the combination of many pieces of evidence about a document's potential relevance. In the case of the retrieval models described in previous sections, the evidence consists of word occurrences that reflect topical content. In general, however, there can be many other types of evidence that should be considered. Even considering words, we may want to take into account whether certain words occur near each other, whether words occur in particular document structures, such as section headings or titles, or whether words are related to each other. In addition, evidence such as the date of publication, the document type, or, in the case of web search, the PageRank number will also be important. Although a retrieval algorithm such as query likelihood or BM25 could be extended to include some of these types of evidence, it is difficult not to resort to heuristic "fixes" that make the retrieval algorithm difficult to tune and adapt to new retrieval applications. Instead, what we really need is a framework where we can describe the different types of evidence, their relative importance, and how they should be combined. The *inference network* retrieval model, which has been