UNIT5
Probabilistic Information Retrieval

Probabilistic Information Retrieval- Review of basic probability theory, Probability ranking principle, Binary independence model, Probability estimates, Text Classification- Rocchio classifier, KNearest neighbor classifier, Linear and nonlinear classifiers. Text Clustering- Clustering in information retrieval, Evaluation of clustering

Review of basic probability theory

We hope that the reader has seen a little basic probability theory previously. We will give a very quick review; some references for further reading appear at the end of the chapter. A variable A represents an event (a subset of the space of possible outcomes). Equivalently, we can represent the subset via a *random variable* , which is a function from outcomes to real numbers; the subset is the domain over which the random variable A has a particular value. Often we will not know with certainty whether an event is true in the world. We can ask the probability of the event $0 \leq P(A) \leq 1$. For two events A and B, the joint event of both events occurring is described by the joint probability $P(A,B)$. The conditional probability $P(A|B)$ expresses the probability of event A given that event B occurred. The fundamental relationship between joint and conditional probabilities is given by the *chain rule* :

$$P(A,B) = P(A \cap B) = P(A|B)P(B) = P(B|A)P(A) \tag{56}$$

Without making any assumptions, the probability of a joint event equals the probability of one of the events multiplied by the probability of the other event conditioned on knowing the first event happened.

Writing $P(\overline{A})$ for the complement of an event, we similarly have:

$$P(\overline{A},B) = P(B|\overline{A})P(\overline{A}) \tag{57}$$

Probability theory also has a *partition rule* , which says that if an event B can be divided into an exhaustive set of disjoint subcases, then the probability of B is the sum of the probabilities of the subcases. A special case of this rule gives that:

$$P(B) = P(A,B) + P(\overline{A},B) \tag{58}$$

From these we can derive *Bayes' Rule* for inverting conditional probabilities:

$$P(A \mid B) = P(B \mid A)P(A)\frac{}{P(B)=\left[\frac{P(B|A)}{\sum_{X\in\{A,\overline{A}\}} P(B|X)P(X)}\right]P(A)} \qquad (59)$$

This equation can also be thought of as a way of updating probabilities. We start off with an initial estimate of how likely the event A is when we do not have any other information; this is the *prior probability* $P(A)$. Bayes' rule lets us derive a *posterior probability* $P(A|B)$ after having seen the evidence B, based on the *likelihood* of B occurring in the two cases that A does or does not hold.

Finally, it is often useful to talk about the *odds* of an event, which provide a kind of multiplier for how probabilities change:

$$\text{Odds:} \qquad O(A) = \frac{P(A)}{P(\overline{A})} = \frac{P(A)}{1 - P(A)}$$

The Probability Ranking Principle

The 1/0 loss case

We assume a ranked retrieval setup as in Section 6.3 , where there is a collection of documents, the user issues a query, and an ordered list of documents is returned. We also assume a binary notion of relevance as in Chapter 8 . For a query q and a document d in the collection, let $R_{d,q}$ be an indicator random variable that says whether d is relevant with respect to a given query q. That is, it takes on a value of 1 when the document is relevant and 0 otherwise. In context we will often write just R for $R_{d,q}$.

Using a probabilistic model, the obvious order in which to present documents to the user is to rank documents by their estimated probability of relevance with respect to the information need: $P(R=1|d,q)$. This is the basis of the *Probability Ranking Principle* (PRP) (van Rijsbergen, 1979, 113-114):

``If a reference retrieval system's 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.''

In the simplest case of the PRP, there are no retrieval costs or other utility concerns that would differentially weight actions or errors. You lose a point for either returning a nonrelevant document or failing to return a relevant document (such a binary situation where you are evaluated on your *accuracy* is called *1/0 loss*). The goal is to return the best possible results as the top k documents, for any value of k the user chooses to examine. The PRP then says to simply rank all documents in decreasing order of $P(R=1|d,q)$. If a set of retrieval results is to be returned, rather than an ordering, the *Bayes Optimal Decision Rule* , the decision which minimizes the risk of loss, is to simply return documents that are more likely relevant than nonrelevant:

$$d \text{ is relevant iff } P(R=1|d,q) > P(R=0|d,q) \qquad (61)$$

Theorem. The PRP is optimal, in the sense that it minimizes the expected loss (also known as the *Bayes risk*) under 1/0 loss.
End theorem.

The PRP with retrieval costs

Suppose, instead, that we assume a model of retrieval costs. Let C_1 be the cost of not retrieving a relevant document and C_0 the cost of retrieval of a nonrelevant document. Then the Probability Ranking Principle says that if for a specific document d and for all documents d' not yet retrieved

$$C_0 \cdot P(R=0|d) - C_1 \cdot P(R=1|d) \leq C_0 \cdot P(R=0|d') - C_1 \cdot P(R=1|d') \qquad (62)$$

then d is the next document to be retrieved. Such a model gives a formal framework where we can model differential costs of false positives and false negatives and even system performance issues at the modeling stage, rather than simply at the evaluation stage, as we did in Section 8.6 (page). However, we will not further consider loss/utility models in this chapter.

The Binary Independence Model

The *Binary Independence Model* (BIM) we present in this section is the model that has traditionally been used with the PRP. It introduces some simple assumptions, which make estimating the probability function $P(R|d,q)$ practical. Here, ``binary'' is equivalent to Boolean: documents and queries are both represented as binary term incidence vectors. That is, a document d is represented by the vector $\vec{x} = (x_1, \ldots, x_M)$ where $x_t = 1$ if term t is present in document d and $x_t = 0$ if t is not present in d. With this representation, many possible documents have the same vector representation. Similarly, we represent q by the incidence vector $\vec{q}$ (the distinction between q and $\vec{q}$ is less central since commonly q is in the form of a set of words). ``Independence'' means that terms are modeled as occurring in documents independently. The model recognizes no association between terms. This assumption is far from correct, but it nevertheless often gives satisfactory results in practice; it is the ``naive'' assumption of Naive Bayes models, discussed further in Section 13.4 (page). Indeed, the Binary Independence Model is exactly the same as the multivariate Bernoulli Naive Bayes model presented in Section 13.3 (page). In a sense this assumption is equivalent to an assumption of the vector space model, where each term is a dimension that is orthogonal to all other terms.

We will first present a model which assumes that the user has a single step information need. As discussed in Chapter 9 , seeing a range of results might let the user refine their information need. Fortunately, as mentioned there, it is straightforward to extend the Binary Independence Model so as to provide a framework for relevance feedback, and we present this model in Section 11.3.4 .

To make a probabilistic retrieval strategy precise, we need to estimate how terms in documents contribute to relevance, specifically, we wish to know how term frequency, document frequency, document length, and other statistics that we can compute influence judgments about document relevance, and how they can be reasonably combined to estimate the probability of document relevance. We then order documents by decreasing estimated probability of relevance.

We assume here that the relevance of each document is independent of the relevance of other documents. As we noted in Section 8.5.1 (page), this is incorrect: the assumption is especially harmful in practice if it allows a system to return duplicate or near duplicate documents. Under the BIM, we model the probability $P(R|d,q)$ that a document is relevant via the probability in terms of term incidence vectors $P(R|\vec{x},\vec{q})$. Then, using Bayes rule, we have:

$$P(R=1|\vec{x},\vec{q}) = \frac{P(\vec{x}|R=1,\vec{q})P(R=1|\vec{q})}{P(\vec{x}|\vec{q})} \tag{63}$$

$$P(R=0|\vec{x},\vec{q}) = \frac{P(\vec{x}|R=0,\vec{q})P(R=0|\vec{q})}{P(\vec{x}|\vec{q})} \tag{64}$$

Here, $P(\vec{x}|R=1,\vec{q})$ and $P(\vec{x}|R=0,\vec{q})$ are the probability that if a relevant or nonrelevant, respectively, document is retrieved, then that document's representation is $\vec{x}$. You should think of this quantity as defined with respect to a space of possible documents in a domain. How do we compute all these probabilities? We never know the exact probabilities, and so we have to use estimates: Statistics about the actual document collection are used to estimate these probabilities. $P(R=1|\vec{q})$ and $P(R=0|\vec{q})$ indicate the prior probability of retrieving a relevant or nonrelevant document respectively for a query $\vec{q}$. Again, if we knew the percentage of relevant documents in the collection, then we could use this number to estimate $P(R=1|\vec{q})$ and $P(R=0|\vec{q})$. Since a document is either relevant or nonrelevant to a query, we must have that:

$$P(R=1|\vec{x},\vec{q}) + P(R=0|\vec{x},\vec{q}) = 1 \tag{65}$$

Deriving a ranking function for query terms

Given a query q, we wish to order returned documents by descending $P(R=1|d,q)$. Under the BIM, this is modeled as ordering by $P(R=1|\vec{x},\vec{q})$. Rather than estimating this probability directly, because we are interested only in the ranking of documents, we work with some other quantities which are easier to compute and which give the same ordering of documents. In particular, we can rank documents by their odds of relevance (as the odds of relevance is monotonic with the probability of relevance). This makes things easier, because we can ignore the common denominator in Rxq-bayes, giving:

$$O(R|\vec{x},\vec{q}) = \frac{P(R=1|\vec{x},\vec{q})}{P(R=0|\vec{x},\vec{q})} = \frac{\frac{P(R=1|\vec{q})P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|\vec{q})}}{\frac{P(R=0|\vec{q})P(\vec{x}|R=0,\vec{q})}{P(\vec{x}|\vec{q})}} = \frac{P(R=1|\vec{q})}{P(R=0|\vec{q})} \cdot \frac{P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|R=0,\vec{q})} \tag{66}$$

The left term in the rightmost expression of Equation 66 is a constant for a given query. Since we are only ranking documents, there is thus no need for us to estimate it. The right-hand term does, however, require estimation, and this initially appears to be difficult: How can we accurately estimate the probability of an entire term incidence vector occurring? It is at this point that we make the *Naive Bayes conditional independence assumption* that the presence or absence of a word in a document is independent of the presence or absence of any other word (given the query):

$$\frac{P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|R=0,\vec{q})} = \prod_{t=1}^{M} \frac{P(x_t|R=1,\vec{q})}{P(x_t|R=0,\vec{q})} \tag{67}$$

So:

$$O(R|\vec{x},\vec{q}) = O(R|\vec{q}) \cdot \prod_{t=1}^{M} \frac{P(x_t|R=1,\vec{q})}{P(x_t|R=0,\vec{q})} \tag{68}$$

Since each x_t is either 0 or 1, we can separate the terms to give:

$$O(R|\vec{x},\vec{q}) = O(R|\vec{q}) \cdot \prod_{t:x_t=1} \frac{P(x_t=1|R=1,\vec{q})}{P(x_t=1|R=0,\vec{q})} \cdot \prod_{t:x_t=0} \frac{P(x_t=0|R=1,\vec{q})}{P(x_t=0|R=0,\vec{q})} \tag{69}$$

Henceforth, let $p_t = P(x_t=1|R=1,\vec{q})$ be the probability of a term appearing in a document relevant to the query, and $u_t = P(x_t=1|R=0,\vec{q})$ be the probability of a term appearing in a nonrelevant document. These quantities can be visualized in the following

contingency table where the columns add to 1:
(H)

document		relevant ($R=1$)	nonrelevant ($R=0$)
Term present	$x_t = 1$	p_t	u_t
Term absent	$x_t = 0$	$1 - p_t$	$1 - u_t$

Let us make an additional simplifying assumption that terms not occurring in the query are equally likely to occur in relevant and nonrelevant documents: that is, if $q_t = 0$ then $p_t = u_t$. (This assumption can be changed, as when doing relevance feedback in Section 11.3.4 .) Then we need only consider terms in the products that appear in the query, and so,

$$O(R|\vec{q},\vec{x}) = O(R|\vec{q}) \cdot \prod_{t:x_t=q_t=1} \frac{p_t}{u_t} \cdot \prod_{t:x_t=0,q_t=1} \frac{1-p_t}{1-u_t} \tag{70}$$

The left product is over query terms found in the document and the right product is over query terms not found in the document.

We can manipulate this expression by including the query terms found in the document into the right product, but simultaneously dividing through by them in the left product, so the value is unchanged. Then we have:

$$O(R|\vec{q},\vec{x}) = O(R|\vec{q}) \cdot \prod_{t:x_t=q_t=1} \frac{p_t(1-u_t)}{u_t(1-p_t)} \cdot \prod_{t:q_t=1} \frac{1-p_t}{1-u_t} \tag{71}$$

The left product is still over query terms found in the document, but the right product is now over all query terms. That means that this right product is a constant for a particular query, just like the odds $O(R|\vec{q})$. So the only quantity that needs to be estimated to rank documents for relevance to a query is the left product. We can equally rank documents by the logarithm of this term, since log is a monotonic function. The resulting quantity used for ranking is called the *Retrieval Status Value* (RSV) in this model:

$$RSV_d = \log \prod_{t:x_t=q_t=1} \frac{p_t(1-u_t)}{u_t(1-p_t)} = \sum_{t:x_t=q_t=1} \log \frac{p_t(1-u_t)}{u_t(1-p_t)} \tag{72}$$

So everything comes down to computing the RSV. Define c_t:

$$c_t = \log \frac{p_t(1-u_t)}{u_t(1-p_t)} = \log \frac{p_t}{(1-p_t)} + \log \frac{1-u_t}{u_t} \tag{73}$$

The c_t terms are log odds ratios for the terms in the query. We have the odds of the term appearing if the document is relevant ($p_t/(1-p_t)$) and the odds of the term appearing if the document is nonrelevant ($u_t/(1-u_t)$). The *odds ratio* is the ratio of two such odds, and then we finally take the log of that quantity. The value will be 0 if a term has equal odds of appearing in relevant and nonrelevant documents, and positive if it is more likely to appear in relevant documents. The c_t quantities function as term weights in the model, and the document score for a query is $RSV_d = \sum_{x_t=q_t=1} c_t$. Operationally, we sum them in accumulators for query terms appearing in documents, just as for the vector space model calculations discussed in Section 7.1 (page). We now turn to how we estimate these c_t quantities for a particular collection and query.

Probability estimates in theory

For each term t, what would these c_t numbers look like for the whole collection? odds-ratio-ct-contingency gives a contingency table of counts of documents in the collection, where df_t is the number of documents that contain term t:

(I)

	documents	relevant	nonrelevant	Total
Term present	$x_t = 1$	s	$\mathrm{df}_t - s$	df_t
Term absent	$x_t = 0$	$S - s$	$(N - \mathrm{df}_t) - (S - s)$	$N - \mathrm{df}_t$
	Total	S	$N - S$	N

Using this, $p_t = s/S$ and $u_t = (\mathrm{df}_t - s)/(N - S)$ and

$$c_t = K(N, \mathrm{df}_t, S, s) = \log \frac{s/(S-s)}{(\mathrm{df}_t - s)/((N - \mathrm{df}_t) - (S - s))} \tag{74}$$

To avoid the possibility of zeroes (such as if every or no relevant document has a particular term) it is fairly standard to *add* $\frac{1}{2}$ to each of the quantities in the center 4 terms of odds-ratio-ct-contingency, and then to adjust the marginal counts (the totals) accordingly (so, the bottom right cell totals $N+2$). Then we have:

$$\hat{c}_t = K(N, \mathrm{df}_t, S, s) = \log \frac{(s+\frac{1}{2})/(S-s+\frac{1}{2})}{(\mathrm{df}_t - s + \frac{1}{2})/(N - \mathrm{df}_t - S + s + \frac{1}{2})} \tag{75}$$

Adding $\frac{1}{2}$ in this way is a simple form of smoothing. For trials with categorical outcomes (such as noting the presence or absence of a term), one way to estimate the probability of an event from data is simply to count the number of times an event occurred divided by the total number of trials. This is referred to as the *relative frequency* of the event. Estimating the probability as the relative frequency is the *maximum likelihood estimate* (or *MLE*), because this value makes the observed data maximally likely. However, if we simply use the MLE, then the probability given to events we happened to see is usually too high, whereas other events may be completely unseen and giving them as a probability estimate their relative frequency of 0 is both an underestimate, and normally breaks our models, since anything multiplied by 0 is 0. Simultaneously decreasing the estimated probability of seen events and increasing the probability of unseen events is referred to as *smoothing* . One simple way of smoothing is to *add a number* α to each of the observed counts.
These *pseudocounts* correspond to the use of a uniform distribution over the vocabulary as a *Bayesian prior* , following Equation 59. We initially assume a uniform distribution over events, where the size of α denotes the strength of our belief in uniformity, and we then update the probability based on observed events. Since our belief in uniformity is weak, we use $\alpha = \frac{1}{2}$. This is a form of *maximum a posteriori* (*MAP*) estimation, where we choose the most likely point value for probabilities based on the prior and the observed evidence, following Equation 59. We will further discuss methods of smoothing estimated counts to give probability models in Section 12.2.2 (page ❑); the simple method of *adding* $\frac{1}{2}$ to each observed count will do for now.

Probability estimates in practice

Under the assumption that relevant documents are a very small percentage of the collection, it is plausible to approximate statistics for nonrelevant documents by statistics from the whole collection. Under this assumption, u_t (the probability of term occurrence in nonrelevant documents for a query) is df_t/N and

$$\log[(1-u_t)/u_t] = \log[(N-\mathrm{df}_t)/\mathrm{df}_t] \approx \log N/\mathrm{df}_t \qquad (76)$$

In other words, we can provide a theoretical justification for the most frequently used form of *idf* weighting, which we saw in Section 6.2.1 .

The approximation technique in Equation 76 cannot easily be extended to relevant documents. The quantity p_t can be estimated in various ways:

1. We can use the frequency of term occurrence in known relevant documents (if we know some). This is the basis of probabilistic approaches to relevance feedback weighting in a feedback loop, discussed in the next subsection.
2. Croft and Harper (1979) proposed using a constant in their combination match model. For instance, we might assume that p_t is constant over all terms x_t in the query and that $p_t = 0.5$. This means that each term has even odds of appearing in a relevant document, and so the p_t and $(1-p_t)$ factors cancel out in the expression for RSV. Such an estimate is weak, but doesn't disagree violently with our hopes for the search terms appearing in many but not all relevant documents. Combining this method with our earlier approximation for u_t, the document ranking is determined simply by which query terms occur in documents scaled by their idf weighting. For short documents (titles or abstracts) in situations in which iterative searching is undesirable, using this weighting term alone can be quite satisfactory, although in many other circumstances we would like to do better.
3. Greiff (1998) argues that the constant estimate of p_t in the Croft and Harper (1979) model is theoretically problematic and not observed empirically: as might be expected, p_t is shown to rise with df_t. Based on his data analysis, a plausible proposal would be to use the estimate $p_t = \frac{1}{3} + \frac{2}{3}\mathrm{df}_t/N$.

Iterative methods of estimation, which combine some of the above ideas, are discussed in the next subsection.

Probabilistic approaches to relevance feedback

We can use (pseudo-)relevance feedback, perhaps in an iterative process of estimation, to get a more accurate estimate of p_t. The probabilistic approach to relevance feedback works as follows:

1. Guess initial estimates of p_t and u_t. This can be done using the probability estimates of the previous section. For instance, we can assume that p_t is constant over all x_t in the query, in particular, perhaps taking $p_t = \frac{1}{2}$.
2. Use the current estimates of p_t and u_t to determine a best guess at the set of relevant documents $R = \{d : R_{d,q} = 1\}$. Use this model to retrieve a set of candidate relevant documents, which we present to the user.
3. We interact with the user to refine the model of R. We do this by learning from the user relevance judgments for some subset of documents V. Based on relevance judgments, V is partitioned into two subsets: $VR = \{d \in V, R_{d,q} = 1\} \subset R$ and $VNR = \{d \in V, R_{d,q} = 0\}$, which is disjoint from R.
4. We reestimate p_t and u_t on the basis of known relevant and nonrelevant documents. If the sets VR and VNR are large enough, we may be able to estimate these quantities directly from these documents as maximum likelihood estimates:

$$p_t = |VR_t|/|VR| \tag{77}$$

5.

6. (where VR_t is the set of documents in VR containing x_t). In practice, we usually need to smooth these estimates. We can do this by *adding* $\frac{1}{2}$ to both the count $|VR_t|$ and to the number of relevant documents not containing the term, giving:

$$p_t = \frac{|VR_t| + \frac{1}{2}}{|VR| + 1} \tag{78}$$

7.

8. However, the set of documents judged by the user (V) is usually very small, and so the resulting statistical estimate is quite unreliable (noisy), even if the estimate is smoothed. So it is often better to combine the new information with the original guess in a process of *Bayesian updating* . In this case we have:

$$p_t^{(k+1)} = \frac{|VR_t| + \kappa p_t^{(k)}}{|VR| + \kappa} \tag{79}$$

9.

10. Here $p_t^{(k)}$ is the k^{th} estimate for p_t in an iterative updating process and is used as a Bayesian prior in the next iteration with a weighting of κ. Relating this equation back to Equation 59 requires a bit more probability theory than we have presented here (we need to use a beta distribution prior, conjugate to the Bernoulli random variable X_t). But the form of the resulting equation is quite straightforward: rather than uniformly distributing pseudocounts, we now distribute a total of κ pseudocounts according to the previous estimate, which acts as the prior distribution. In the absence of other evidence (and assuming that the user is perhaps indicating roughly 5 relevant or nonrelevant documents) then a value of around $\kappa = 5$ is perhaps appropriate. That is, the prior is strongly weighted so that the estimate does not change too much from the evidence provided by a very small number of documents.
11. Repeat the above process from step 2, generating a succession of approximations to R and hence p_t, until the user is satisfied.

It is also straightforward to derive a pseudo-relevance feedback version of this algorithm, where we simply pretend that $VR = V$. More briefly:

1. Assume initial estimates for p_t and u_t as above.
2. Determine a guess for the size of the relevant document set. If unsure, a conservative (too small) guess is likely to be best. This motivates use of a fixed size set V of highest ranked documents.

3. Improve our guesses for p_t and u_t. We choose from the methods of and 79 for re-estimating p_t, except now based on the set V instead of VR. If we let V_t be the subset of documents in V containing x_t and use *add* $\frac{1}{2}$ *smoothing* , we get:

$$p_t = \frac{|V_t| + \frac{1}{2}}{|V| + 1} \tag{80}$$

4.

5. and if we assume that documents that are not retrieved are nonrelevant then we can update our u_t estimates as:

$$u_t = \frac{\mathrm{df}_t - |V_t| + \frac{1}{2}}{N - |V| + 1} \tag{81}$$

6.

7. Go to step 2 until the ranking of the returned results converges.

Once we have a real estimate for p_t then the c_t weights used in the RSV value look almost like a tf-idf value. For instance, using Equation 73, Equation 76, and Equation 80, we have:

$$c_t = \log\left[\frac{p_t}{1-p_t} \cdot \frac{1-u_t}{u_t}\right] \approx \log\left[\frac{|V_t| + \frac{1}{2}}{|V| - |V_t| + 1} \cdot \frac{N}{\mathrm{df}_t}\right] \tag{82}$$

But things aren't quite the same: $p_t/(1-p_t)$ measures the (estimated) proportion of relevant documents that the term t occurs in, not term frequency. Moreover, if we apply log identities:

$$c_t = \log\frac{|V_t| + \frac{1}{2}}{|V| - |V_t| + 1} + \log\frac{N}{\mathrm{df}_t} \tag{83}$$

we see that we are now *adding* the two log scaled components rather than multiplying them.

An appraisal and some extensions

An appraisal of probabilistic models

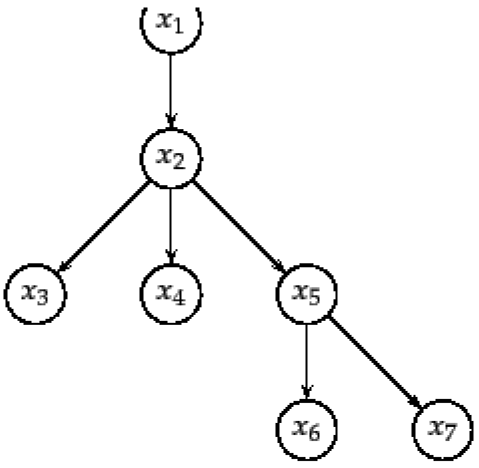
Probabilistic methods are one of the oldest formal models in IR. Already in the 1970s they were held out as an opportunity to place IR on a firmer theoretical footing, and with the resurgence of probabilistic methods in computational linguistics in the 1990s, that hope has returned, and probabilistic methods are again one of the currently hottest topics in IR. Traditionally, probabilistic IR has had neat ideas but the methods have never won on performance. Getting reasonable approximations of the needed probabilities for a probabilistic IR model is possible, but it requires some major assumptions. In the BIM these are:

- a Boolean representation of documents/queries/relevance
- term independence
- terms not in the query don't affect the outcome
- document relevance values are independent

It is perhaps the severity of the modeling assumptions that makes achieving good performance difficult. A general problem seems to be that probabilistic models either require partial relevance information or else only allow for deriving apparently inferior term weighting models.

Things started to change in the 1990s when the BM25 weighting scheme, which we discuss in the next section, showed very good performance, and started to be adopted as a term weighting scheme by many groups. The difference between ``vector space'' and ``probabilistic'' IR systems is not that great: in either case, you build an information retrieval scheme in the exact same way that we discussed in Chapter 7 . For a probabilistic IR system, it's just that, at the end, you score queries not by cosine similarity and tf-idf in a vector space, but by a slightly different formula motivated by probability theory. Indeed, sometimes people have changed an existing vector-space IR system into an effectively probabilistic system simply by adopted term weighting formulas from probabilistic models. In this section, we briefly present three extensions of the traditional probabilistic model, and in the next chapter, we look at the somewhat different probabilistic language modeling approach to IR.

Tree-structured dependencies between terms



► **Figure 11.1** A tree of dependencies between terms. In this graphical model representation, a term x_i is directly dependent on a term x_k if there is an arrow $x_k \rightarrow x_i$.

Some of the assumptions of the BIM can be removed. For example, we can remove the assumption that terms are independent. This assumption is very far from true in practice. A case that particularly violates this assumption is term pairs like Hong and Kong, which are strongly dependent. But dependencies can occur in various complex configurations, such as between the set of terms New, York, England, City, Stock, Exchange, and University. van Rijsbergen (1979) proposed a simple, plausible model which allowed a tree structure of term dependencies, as in Figure 11.1 . In this model each term can be directly dependent on only one other term, giving a tree structure of dependencies. When it was invented in the 1970s, estimation problems held back the practical success of this model, but the idea was reinvented as the Tree Augmented Naive Bayes model by Friedman and Goldszmidt (1996), who used it with some success on various machine learning data sets

Okapi BM25: a non-binary model

The BIM was originally designed for short catalog records and abstracts of fairly consistent length, and it works reasonably in these contexts, but for modern full-text search collections, it seems clear that a model should pay attention to term frequency and document length, as in Chapter 6 . The *BM25 weighting scheme* , often called *Okapi weighting* , after the system in which it was first implemented, was developed as a way of building a probabilistic model sensitive to these quantities while not introducing too many additional parameters into the model (Spärck Jones et al., 2000). We will not develop the full theory behind the model here, but just present a series of forms that build up to the standard form now used for document scoring. The simplest score for document d is just idf weighting of the query terms present, as in Equation 76:

$$RSV_d = \sum_{t \in q} \log \frac{N}{\mathrm{df}_t} \tag{84}$$

Sometimes, an alternative version of *idf* is used. If we start with the formula in Equation 75 but in the absence of relevance feedback information we estimate that $S = s = 0$, then we get an alternative idf formulation as follows:

$$RSV_d = \sum_{t \in q} \log \frac{N - \mathrm{df}_t + \frac{1}{2}}{\mathrm{df}_t + \frac{1}{2}} \tag{85}$$

This variant behaves slightly strangely: if a term occurs in over half the documents in the collection then this model gives a negative term weight, which is presumably undesirable. But, assuming the use of a stop list, this normally doesn't happen, and the value for each summand can be given a floor of 0.

We can improve on Equation 84 by factoring in the frequency of each term and document length:

$$RSV_d = \sum_{t \in q} \log \left[\frac{N}{\mathrm{df}_t} \right] \cdot \frac{(k_1 + 1)\mathrm{tf}_{td}}{k_1((1-b) + b \times (L_d / L_{ave})) + \mathrm{tf}_{td}} \tag{86}$$

Here, tf_{td} is the frequency of term t in document d, and L_d and L_{ave} are the length of document d and the average document length for the whole collection. The variable k_1 is a positive tuning parameter that calibrates the document term frequency scaling. A k_1 value of 0 corresponds to a binary model (no term frequency), and a large value corresponds to using raw term frequency. b is another tuning parameter ($0 \leq b \leq 1$) which determines the scaling by document length: $b = 1$ corresponds to fully scaling the term weight by the document length, while $b = 0$ corresponds to no length normalization.

If the query is long, then we might also use similar weighting for query terms. This is appropriate if the queries are paragraph long information needs, but unnecessary for short queries.

$$RSV_d = \sum_{t \in q} \left[\log \frac{N}{\mathrm{df}_t} \right] \cdot \frac{(k_1 + 1)\mathrm{tf}_{td}}{k_1((1 - b) + b \times (L_d / L_{ave})) + \mathrm{tf}_{td}} \cdot \frac{(k_3 + 1)\mathrm{tf}_{tq}}{k_3 + \mathrm{tf}_{tq}} \quad (87)$$

with tf_{tq} being the frequency of term t in the query q, and k_3 being another positive tuning parameter that this time calibrates term frequency scaling of the query. In the equation presented, there is no length normalization of queries (it is as if $b = 0$ here). Length normalization of the query is unnecessary because retrieval is being done with respect to a single fixed query. The tuning parameters of these formulas should ideally be set to optimize performance on a development test collection (see page 8.1). That is, we can search for values of these parameters that maximize performance on a separate development test collection (either manually or with optimization methods such as grid search or something more advanced), and then use these parameters on the actual test collection. In the absence of such optimization, experiments have shown reasonable values are to set k_1 and k_3 to a value between 1.2 and 2 and $b = 0.75$.

If we have relevance judgments available, then we can use the full form of smoothed-rf in place of the approximation $\log(N/\mathrm{df}_t)$ introduced in prob-idf:

$$RSV_d = \sum_{t \in q} \log \left[\left[\frac{(|VR_t| + \frac{1}{2})/(|VNR_t| + \frac{1}{2})}{(\mathrm{df}_t - |VR_t| + \frac{1}{2})/(N - \mathrm{df}_t - |VR| + |VR_t| + \frac{1}{2})} \right] \right. \quad (88)$$

$$\left. \times \frac{(k_1 + 1)\mathrm{tf}_{td}}{k_1((1-b) + b(L_d/L_{ave})) + \mathrm{tf}_{td}} \times \frac{(k_3 + 1)\mathrm{tf}_{tq}}{k_3 + \mathrm{tf}_{tq}} \right] \quad (89)$$

Here, VR_t, NVR_t, and VR are used as in Section 11.3.4 . The first part of the expression reflects relevance feedback (or just idf weighting if no relevance information is available), the second implements document term frequency and document length scaling, and the third considers term frequency in the query.

Rather than just providing a term weighting method for terms in a user's query, relevance feedback can also involve augmenting the query (automatically or with manual review) with some (say, 10-20) of the top terms in the known-relevant documents as ordered by the relevance factor $\hat{c}_t$ from Equation 75, and the above formula can then be used with such an augmented query vector $\vec{q}$.

The BM25 term weighting formulas have been used quite widely and quite successfully across a range of collections and search tasks. Especially in the TREC evaluations, they performed well and were widely adopted by many groups. See Spärck Jones et al. (2000) for extensive motivation and discussion of experimental results.

Bayesian network approaches to IR

Turtle and Croft (1989;1991) introduced into information retrieval the use of *Bayesian networks* (Jensen and Jensen, 2001), a form of probabilistic graphical model. We skip the details because fully introducing the formalism of Bayesian networks would require much too much space, but conceptually, Bayesian networks use directed graphs to show probabilistic dependencies between variables, as in Figure 11.1 , and have led to the development of sophisticated algorithms for propagating influence so as to allow learning and inference with arbitrary knowledge within arbitrary directed acyclic graphs. Turtle and Croft used a sophisticated network to better model the complex dependencies between a document and a user's information need.

The model decomposes into two parts: a document collection network and a query network. The document collection network is large, but can be precomputed: it maps from documents to terms to concepts. The concepts are a thesaurus-based expansion of the terms appearing in the document. The query network is relatively small but a new network needs to be built each time a query comes in, and then attached to the document network. The query network maps from query terms, to query subexpressions (built using probabilistic or ``noisy'' versions of AND and OR operators), to the user's information need.

The result is a flexible probabilistic network which can generalize various simpler Boolean and probabilistic models. Indeed, this is the primary case of a statistical ranked retrieval model that naturally supports structured query operators. The system allowed efficient large-scale retrieval, and was the basis of the InQuery text retrieval system, built at the University of Massachusetts. This system performed very well in TREC evaluations and for a time was sold commercially. On the other hand, the model still used various approximations and independence assumptions to make parameter estimation and computation possible. There has not been much follow-on work along these lines, but we would note that this model was actually built very early on in the modern era of using Bayesian networks, and there have been many subsequent developments in the theory, and the time is perhaps right for a new generation of Bayesian network-based information retrieval systems.