12 Language models for information retrieval

A common suggestion to users for coming up with good queries is to think of words that would likely appear in a relevant document, and to use those words as the query. The language modeling approach to IR directly models that idea: a document is a good match to a query if the document model is likely to generate the query, which will in turn happen if the document contains the query words often. This approach thus provides a different realization of some of the basic ideas for document ranking which we saw in Section 6.2 (page 117). Instead of overtly modeling the probability $P(R = 1|q, d)$ of relevance of a document d to a query q, as in the traditional probabilistic approach to IR (Chapter 11), the basic language modeling approach instead builds a probabilistic language model M_d from each document d, and ranks documents based on the probability of the model generating the query: $P(q|M_d)$.

In this chapter, we first introduce the concept of language models (Section 12.1) and then describe the basic and most commonly used language modeling approach to IR, the Query Likelihood Model (Section 12.2). After some comparisons between the language modeling approach and other approaches to IR (Section 12.3), we finish by briefly describing various extensions to the language modeling approach (Section 12.4).

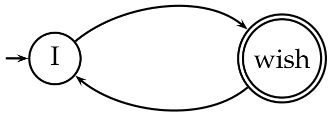
12.1 Language models

12.1.1 Finite automata and language models

What do we mean by a document model generating a query? A traditional *generative model* of a language, of the kind familiar from formal language theory, can be used either to recognize or to generate strings. For example, the finite automaton shown in Figure 12.1 can generate strings that include the examples shown. The full set of strings that can be generated is called the *language* of the automaton.[1]

GENERATIVE MODEL

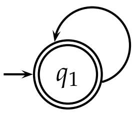
LANGUAGE



I wish
I wish I wish
I wish I wish I wish
I wish I wish I wish I wish I wish I wish
...

CANNOT GENERATE: wish I wish

► **Figure 12.1** A simple finite automaton and some of the strings in the language it generates. → shows the start state of the automaton and a double circle indicates a (possible) finishing state.



$P(\text{STOP}|q_1) = 0.2$

the	0.2
a	0.1
frog	0.01
toad	0.01
said	0.03
likes	0.02
that	0.04
...	...

► **Figure 12.2** A one-state finite automaton that acts as a unigram language model. We show a partial specification of the state emission probabilities.

If instead each node has a probability distribution over generating different terms, we have a language model. The notion of a language model is inherently probabilistic. A *language model* is a function that puts a probability measure over strings drawn from some vocabulary. That is, for a language model M over an alphabet Σ:

LANGUAGE MODEL

$$\sum_{s \in \Sigma^*} P(s) = 1 \tag{12.1}$$

One simple kind of language model is equivalent to a probabilistic finite automaton consisting of just a single node with a single probability distribution over producing different terms, so that $\sum_{t \in V} P(t) = 1$, as shown in Figure 12.2. After generating each word, we decide whether to stop or to loop around and then produce another word, and so the model also requires a probability of stopping in the finishing state. Such a model places a probability distribution over any sequence of words. By construction, it also provides a model for generating text according to its distribution.

[1] Finite automata can have outputs attached to either their states or their arcs; we use states here, because that maps directly on to the way probabilistic automata are usually formalized.

Model M_1		Model M_2	
the	0.2	the	0.15
a	0.1	a	0.12
frog	0.01	frog	0.0002
toad	0.01	toad	0.0001
said	0.03	said	0.03
likes	0.02	likes	0.04
that	0.04	that	0.04
dog	0.005	dog	0.01
cat	0.003	cat	0.015
monkey	0.001	monkey	0.002
...	...	...	...

► **Figure 12.3** Partial specification of two unigram language models.

Example 12.1: To find the probability of a word sequence, we just multiply the probabilities which the model gives to each word in the sequence, together with the probability of continuing or stopping after producing each word. For example,

(12.2)
$$\begin{aligned} P(\text{frog said that toad likes frog}) &= (0.01 \times 0.03 \times 0.04 \times 0.01 \times 0.02 \times 0.01) \\ &\quad \times (0.8 \times 0.8 \times 0.8 \times 0.8 \times 0.8 \times 0.8 \times 0.2) \\ &\approx 0.000000000001573 \end{aligned}$$

As you can see, the probability of a particular string/document, is usually a very small number! Here we stopped after generating *frog* the second time. The first line of numbers are the term emission probabilities, and the second line gives the probability of continuing or stopping after generating each word. An explicit stop probability is needed for a finite automaton to be a well-formed language model according to Equation (12.1). Nevertheless, most of the time, we will omit to include STOP and $(1 - \text{STOP})$ probabilities (as do most other authors). To compare two models for a data set, we can calculate their *likelihood ratio*, which results from simply dividing the probability of the data according to one model by the probability of the data according to the other model. Providing that the stop probability is fixed, its inclusion will not alter the likelihood ratio that results from comparing the likelihood of two language models generating a string. Hence, it will not alter the ranking of documents.[2] Nevertheless, formally, the numbers will no longer truly be probabilities, but only proportional to probabilities. See Exercise 12.4.

LIKELIHOOD RATIO

Example 12.2: Suppose, now, that we have two language models M_1 and M_2, shown partially in Figure 12.3. Each gives a probability estimate to a sequence of

[2] In the IR context that we are leading up to, taking the stop probability to be fixed across models seems reasonable. This is because we are generating queries, and the length distribution of queries is fixed and independent of the document from which we are generating the language model.

terms, as already illustrated in Example 12.1. The language model that gives the higher probability to the sequence of terms is more likely to have generated the term sequence. This time, we will omit STOP probabilities from our calculations. For the sequence shown, we get:

(12.3)

s	frog	said	that	toad	likes	that	dog
M_1	0.01	0.03	0.04	0.01	0.02	0.04	0.005
M_2	0.0002	0.03	0.04	0.0001	0.04	0.04	0.01

$P(s|M_1) = 0.000000000000048$
$P(s|M_2) = 0.000000000000000384$

and we see that $P(s|M_1) > P(s|M_2)$. We present the formulas here in terms of products of probabilities, but, as is common in probabilistic applications, in practice it is usually best to work with sums of log probabilities (cf. page 258).

12.1.2 Types of language models

How do we build probabilities over sequences of terms? We can always use the chain rule from Equation (11.1) to decompose the probability of a sequence of events into the probability of each successive event conditioned on earlier events:

$$P(t_1t_2t_3t_4) = P(t_1)P(t_2|t_1)P(t_3|t_1t_2)P(t_4|t_1t_2t_3) \quad (12.4)$$

The simplest form of language model simply throws away all conditioning context, and estimates each term independently. Such a model is called a *unigram language model*:

UNIGRAM LANGUAGE MODEL

$$P_{\text{uni}}(t_1t_2t_3t_4) = P(t_1)P(t_2)P(t_3)P(t_4) \quad (12.5)$$

BIGRAM LANGUAGE MODEL

There are many more complex kinds of language models, such as *bigram language models*, which condition on the previous term,

$$P_{\text{bi}}(t_1t_2t_3t_4) = P(t_1)P(t_2|t_1)P(t_3|t_2)P(t_4|t_3) \quad (12.6)$$

and even more complex grammar-based language models such as probabilistic context-free grammars. Such models are vital for tasks like speech recognition, spelling correction, and machine translation, where you need the probability of a term conditioned on surrounding context. However, most language-modeling work in IR has used unigram language models. IR is not the place where you most immediately need complex language models, since IR does not directly depend on the structure of sentences to the extent that other tasks like speech recognition do. Unigram models are often sufficient to judge the topic of a text. Moreover, as we shall see, IR language models are frequently estimated from a single document and so it is questionable whether there is enough training data to do more. Losses from data sparseness (see the discussion on page 260) tend to outweigh any gains from richer models. This is an example of the bias-variance tradeoff (cf. Section 14.6, page 308): With limited training data, a more constrained model tends to perform better. In addition, unigram models are more efficient to estimate and apply than higher-order models. Nevertheless, the importance of phrase and proximity queries in IR in general suggests that future work should make use of more sophisticated language models, and some has begun to (see Section 12.5, page 252). Indeed, making this move parallels the model of van Rijsbergen in Chapter 11 (page 231).

12.1.3 Multinomial distributions over words

Under the unigram language model the order of words is irrelevant, and so such models are often called "bag of words" models, as discussed in Chapter 6 (page 117). Even though there is no conditioning on preceding context, this model nevertheless still gives the probability of a particular ordering of terms. However, any other ordering of this bag of terms will have the same probability. So, really, we have a *multinomial distribution* over words. So long as we stick to unigram models, the language model name and motivation could be viewed as historical rather than necessary. We could instead just refer to the model as a multinomial model. From this perspective, the equations presented above do not present the multinomial probability of a bag of words, since they do not sum over all possible orderings of those words, as is done by the multinomial coefficient (the first term on the right-hand side) in the standard presentation of a multinomial model:

MULTINOMIAL DISTRIBUTION

$$P(d) = \frac{L_d!}{\mathrm{tf}_{t_1,d}!\mathrm{tf}_{t_2,d}!\cdots\mathrm{tf}_{t_M,d}!} P(t_1)^{\mathrm{tf}_{t_1,d}} P(t_2)^{\mathrm{tf}_{t_2,d}} \cdots P(t_M)^{\mathrm{tf}_{t_M,d}} \tag{12.7}$$

Here, $L_d = \sum_{1 \le i \le M} \mathrm{tf}_{t_i,d}$ is the length of document d, M is the size of the term vocabulary, and the products are now over the terms in the vocabulary, not the positions in the document. However, just as with STOP probabilities, in practice we can also leave out the multinomial coefficient in our calculations, since, for a particular bag of words, it will be a constant, and so it has no effect on the likelihood ratio of two different models generating a particular bag of words. Multinomial distributions also appear in Section 13.2 (page 258).

The fundamental problem in designing language models is that we do not know what exactly we should use as the model M_d. However, we do generally have a sample of text that is representative of that model. This problem makes a lot of sense in the original, primary uses of language models. For example, in speech recognition, we have a training sample of (spoken) text. But we have to expect that, in the future, users will use different words and in different sequences, which we have never observed before, and so the model has to generalize beyond the observed data to allow unknown words and sequences. This interpretation is not so clear in the IR case, where a document is finite and usually fixed. The strategy we adopt in IR is as follows. We pretend that the document d is only a representative sample of text drawn from a model distribution, treating it like a fine-grained topic. We then estimate a language model from this sample, and use that model to calculate the probability of observing any word sequence, and, finally, we rank documents according to their probability of generating the query.

?

Exercise 12.1 [⋆]

Including stop probabilities in the calculation, what will the sum of the probability estimates of all strings in the language of length 1 be? Assume that you generate a word and then decide whether to stop or not (i.e., the null string is not part of the language).

Exercise 12.2 [⋆]

If the stop probability is omitted from calculations, what will the sum of the scores assigned to strings in the language of length 1 be?

Exercise 12.3 [⋆]

What is the likelihood ratio of the document according to M_1 and M_2 in Example 12.2?

Exercise 12.4 [⋆]

No explicit STOP probability appeared in Example 12.2. Assuming that the STOP probability of each model is 0.1, does this change the likelihood ratio of a document according to the two models?

Exercise 12.5 [⋆⋆]

How might a language model be used in a spelling correction system? In particular, consider the case of context-sensitive spelling correction, and correcting incorrect usages of words, such as *their* in *Are you their?* (See Section 3.5 (page 65) for pointers to some literature on this topic.)

12.2 The query likelihood model

12.2.1 Using query likelihood language models in IR

QUERY LIKELIHOOD MODEL

Language modeling is a quite general formal approach to IR, with many variant realizations. The original and basic method for using language models in IR is the *query likelihood model*. In it, we construct from each document d in the collection a language model M_d. Our goal is to rank documents by $P(d|q)$, where the probability of a document is interpreted as the likelihood that it is relevant to the query. Using Bayes rule (as introduced in Section 11.1, page 220), we have:

$$P(d|q) = P(q|d)P(d)/P(q)$$

$P(q)$ is the same for all documents, and so can be ignored. The prior probability of a document $P(d)$ is often treated as uniform across all d and so it can also be ignored, but we could implement a genuine prior which could include criteria like authority, length, genre, newness, and number of previous people who have read the document. But, given these simplifications, we return results ranked by simply $P(q|d)$, the probability of the query q under the language model derived from d. The Language Modeling approach thus attempts to model the query generation process: Documents are ranked by the probability that a query would be observed as a random sample from the respective document model.

The most common way to do this is using the multinomial unigram language model, which is equivalent to a multinomial Naive Bayes model (page 263), where the documents are the classes, each treated in the estimation as a separate "language". Under this model, we have that:

$$P(q|M_d) = K_q \prod_{t \in V} P(t|M_d)^{\mathrm{tf}_{t,d}} \tag{12.8}$$

where, again $K_q = L_d!/(\mathrm{tf}_{t_1,d}!\mathrm{tf}_{t_2,d}! \cdots \mathrm{tf}_{t_M,d}!)$ is the multinomial coefficient for the query q, which we will henceforth ignore, since it is a constant for a particular query.

For retrieval based on a language model (henceforth LM), we treat the generation of queries as a random process. The approach is to

1. Infer a LM for each document.
2. Estimate $P(q|M_{d_i})$, the probability of generating the query according to each of these document models.
3. Rank the documents according to these probabilities.

The intuition of the basic model is that the user has a prototype document in mind, and generates a query based on words that appear in this document. Often, users have a reasonable idea of terms that are likely to occur in documents of interest and they will choose query terms that distinguish these documents from others in the collection.[3] Collection statistics are an integral part of the language model, rather than being used heuristically as in many other approaches.

12.2.2 Estimating the query generation probability

In this section we describe how to estimate $P(q|M_d)$. The probability of producing the query given the LM M_d of document d using maximum likelihood

[3] Of course, in other cases, they do not. The answer to this within the language modeling approach is translation language models, as briefly discussed in Section 12.4.

estimation (MLE) and the unigram assumption is:

$$\hat{P}(q|M_d) = \prod_{t \in q} \hat{P}_{\text{mle}}(t|M_d) = \prod_{t \in q} \frac{\text{tf}_{t,d}}{L_d} \tag{12.9}$$

where M_d is the language model of document d, $\text{tf}_{t,d}$ is the (raw) term frequency of term t in document d, and L_d is the number of tokens in document d. That is, we just count up how often each word occurred, and divide through by the total number of words in the document d. This is the same method of calculating an MLE as we saw in Section 11.3.2 (page 226), but now using a multinomial over word counts.

The classic problem with using language models is one of estimation (the ˆ symbol on the P's is used above to stress that the model is estimated): terms appear very sparsely in documents. In particular, some words will not have appeared in the document at all, but are possible words for the information need, which the user may have used in the query. If we estimate $\hat{P}(t|M_d) = 0$ for a term missing from a document d, then we get a strict conjunctive semantics: documents will only give a query non-zero probability if all of the query terms appear in the document. Zero probabilities are clearly a problem in other uses of language models, such as when predicting the next word in a speech recognition application, because many words will be sparsely represented in the training data. It may seem rather less clear whether this is problematic in an IR application. This could be thought of as a human-computer interface issue: vector space systems have generally preferred more lenient matching, though recent web search developments have tended more in the direction of doing searches with such conjunctive semantics. Regardless of the approach here, there is a more general problem of estimation: occurring words are also badly estimated; in particular, the probability of words occurring once in the document is normally overestimated, since their one occurrence was partly by chance. The answer to this (as we saw in Section 11.3.2, page 226) is smoothing. But as people have come to understand the LM approach better, it has become apparent that the role of smoothing in this model is not only to avoid zero probabilities. The smoothing of terms actually implements major parts of the term weighting component (Exercise 12.8). It is not just that an unsmoothed model has conjunctive semantics; an unsmoothed model works badly because it lacks parts of the term weighting component.

Thus, we need to smooth probabilities in our document language models: to discount non-zero probabilities and to give some probability mass to unseen words. There's a wide space of approaches to smoothing probability distributions to deal with this problem. In Section 11.3.2 (page 226), we already discussed adding a number (1, 1/2, or a small α) to the observed counts and renormalizing to give a probability distribution.[4] In this section we will mention a couple of other smoothing methods, which involve combining observed counts with a more general reference probability distribution. The general approach is that a non-occurring term should be possible in a query, but its probability should be somewhat close to but no more likely than would be expected by chance from the whole collection. That is, if $\text{tf}_{t,d} = 0$ then

$$\hat{P}(t|M_d) \leq \text{cf}_t/T$$

where cf_t is the raw count of the term in the collection, and T is the raw size (number of tokens) of the entire collection. A simple idea that works well in practice is to use a mixture between a document-specific multinomial distribution and a multinomial distribution estimated from the entire collection:

$$\hat{P}(t|d) = \lambda \hat{P}_{\text{mle}}(t|M_d) + (1-\lambda)\hat{P}_{\text{mle}}(t|M_c) \tag{12.10}$$

where $0 < \lambda < 1$ and M_c is a language model built from the entire document collection. This mixes the probability from the document with the general collection frequency of the word. Such a model is referred to as a *linear interpolation* language model.[5] Correctly setting λ is important to the good performance of this model.

LINEAR INTERPOLATION

An alternative is to use a language model built from the whole collection as a prior distribution in a *Bayesian updating process* (rather than a uniform distribution, as we saw in Section 11.3.2). We then get the following equation:

BAYESIAN SMOOTHING

$$\hat{P}(t|d) = \frac{\text{tf}_{t,d} + \alpha \hat{P}(t|M_c)}{L_d + \alpha} \tag{12.11}$$

Both of these smoothing methods have been shown to perform well in IR experiments; we will stick with the linear interpolation smoothing method for the rest of this section. While different in detail, they are both conceptually similar: in both cases the probability estimate for a word present in the document combines a discounted MLE and a fraction of the estimate of its prevalence in the whole collection, while for words not present in a document, the estimate is just a fraction of the estimate of the prevalence of the word in the whole collection.

The role of smoothing in LMs for IR is not simply or principally to avoid estimation problems. This was not clear when the models were first proposed, but it is now understood that smoothing is essential to the good properties

[4] In the context of probability theory, (re)normalization refers to summing numbers that cover an event space and dividing them through by their sum, so that the result is a probability distribution which sums to 1. This is distinct from both the concept of term normalization in Chapter 2 and the concept of length normalization in Chapter 6, which is done with a L_2 norm.

[5] It is also referred to as Jelinek-Mercer smoothing.

of the models. The reason for this is explored in Exercise 12.8. The extent of smoothing in these two models is controlled by the λ and α parameters: a small value of λ or a large value of α means more smoothing. This parameter can be tuned to optimize performance using a line search (or, for the linear interpolation model, by other methods, such as the expectation maximimization algorithm; see Section 16.5, page 368). The value need not be a constant. One approach is to make the value a function of the query size. This is useful because a small amount of smoothing (a "conjunctive-like" search) is more suitable for short queries, while a lot of smoothing is more suitable for long queries.

To summarize, the retrieval ranking for a query q under the basic LM for IR we have been considering is given by:

$$P(d|q) \propto P(d) \prod_{t \in q} ((1-\lambda)P(t|M_c) + \lambda P(t|M_d)) \tag{12.12}$$

This equation captures the probability that the document that the user had in mind was in fact d.

Example 12.3: Suppose the document collection contains two documents:

- d_1: Xyzzy reports a profit but revenue is down
- d_2: Quorus narrows quarter loss but revenue decreases further

The model will be MLE unigram models from the documents and collection, mixed with $\lambda = 1/2$.

Suppose the query is *revenue down*. Then:

$$\begin{aligned} P(q|d_1) &= [(1/8+2/16)/2] \times [(1/8+1/16)/2] \\ &= 1/8 \times 3/32 = 3/256 \\ P(q|d_2) &= [(1/8+2/16)/2] \times [(0/8+1/16)/2] \\ &= 1/8 \times 1/32 = 1/256 \end{aligned} \tag{12.13}$$

So, the ranking is $d_1 > d_2$.

12.2.3 Ponte and Croft's Experiments

Ponte and Croft (1998) present the first experiments on the language modeling approach to information retrieval. Their basic approach is the model that we have presented until now. However, we have presented an approach where the language model is a mixture of two multinomials, much as in (Miller et al. 1999, Hiemstra 2000) rather than Ponte and Croft's multivariate Bernoulli model. The use of multinomials has been standard in most subsequent work in the LM approach and experimental results in IR, as well as evidence from text classification which we consider in Section 13.3

	Precision			
Rec.	tf-idf	LM	%chg	
0.0	0.7439	0.7590	+2.0	
0.1	0.4521	0.4910	+8.6	
0.2	0.3514	0.4045	+15.1	*
0.3	0.2761	0.3342	+21.0	*
0.4	0.2093	0.2572	+22.9	*
0.5	0.1558	0.2061	+32.3	*
0.6	0.1024	0.1405	+37.1	*
0.7	0.0451	0.0760	+68.7	*
0.8	0.0160	0.0432	+169.6	*
0.9	0.0033	0.0063	+89.3	
1.0	0.0028	0.0050	+76.9	
Ave	0.1868	0.2233	+19.55	*

► **Figure 12.4** Results of a comparison of tf-idf with language modeling (LM) term weighting by Ponte and Croft (1998). The version of tf-idf from the INQUERY IR system includes length normalization of tf. The table gives an evaluation according to 11-point average precision with significance marked with a * according to a Wilcoxon signed rank test. The language modeling approach always does better in these experiments, but note that where the approach shows significant gains is at higher levels of recall.

(page 263), suggests that it is superior. Ponte and Croft argued strongly for the effectiveness of the term weights that come from the language modeling approach over traditional tf-idf weights. We present a subset of their results in Figure 12.4 where they compare tf-idf to language modeling by evaluating TREC topics 202–250 over TREC disks 2 and 3. The queries are sentence-length natural language queries. The language modeling approach yields significantly better results than their baseline tf-idf based term weighting approach. And indeed the gains shown here have been extended in subsequent work.

Exercise 12.6 [$\star$]

Consider making a language model from the following training text:

the martian has landed on the latin pop sensation ricky martin

a. Under a MLE-estimated unigram probability model, what are $P(\text{the})$ and $P(\text{martian})$?

b. Under a MLE-estimated bigram model, what are $P(\text{sensation}|\text{pop})$ and $P(\text{pop}|\text{the})$?

Exercise 12.7 [⋆⋆]

Suppose we have a collection that consists of the 4 documents given in the below table.

docID	Document text
1	click go the shears boys click click click
2	click click
3	metal here
4	metal shears click here

Build a query likelihood language model for this document collection. Assume a mixture model between the documents and the collection, with both weighted at 0.5. Maximum likelihood estimation (mle) is used to estimate both as unigram models. Work out the model probabilities of the queries click, shears, and hence click shears for each document, and use those probabilities to rank the documents returned by each query. Fill in these probabilities in the below table:

Query	Doc 1	Doc 2	Doc 3	Doc 4
click				
shears				
click shears				

What is the final ranking of the documents for the query click shears?

Exercise 12.8 [⋆⋆]

Using the calculations in Exercise 12.7 as inspiration or as examples where appropriate, write one sentence each describing the treatment that the model in Equation (12.10) gives to each of the following quantities. Include whether it is present in the model or not and whether the effect is raw or scaled.

a. Term frequency in a document

b. Collection frequency of a term

c. Document frequency of a term

d. Length normalization of a term

Exercise 12.9 [⋆⋆]

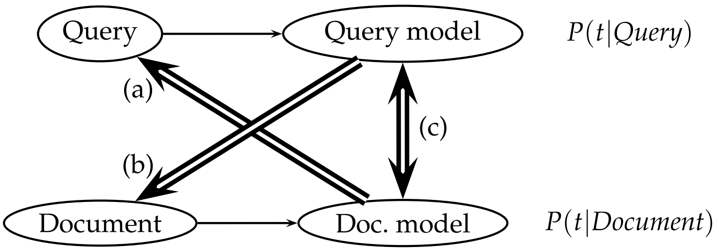
In the mixture model approach to the query likelihood model (Equation (12.12)), the probability estimate of a term is based on the term frequency of a word in a document, and the collection frequency of the word. Doing this certainly guarantees that each term of a query (in the vocabulary) has a non-zero chance of being generated by each document. But it has a more subtle but important effect of implementing a form of term weighting, related to what we saw in Chapter 6. Explain how this works. In particular, include in your answer a concrete numeric example showing this term weighting at work.

12.3 Language modeling versus other approaches in IR

The language modeling approach provides a novel way of looking at the problem of text retrieval, which links it with a lot of recent work in speech and language processing. As Ponte and Croft (1998) emphasize, the language modeling approach to IR provides a different approach to scoring matches between queries and documents, and the hope is that the probabilistic language modeling foundation improves the weights that are used, and hence the performance of the model. The major issue is estimation of the document model, such as choices of how to smooth it effectively. The model has achieved very good retrieval results. Compared to other probabilistic approaches, such as the BIM from Chapter 11, the main difference initially appears to be that the LM approach does away with explicitly modeling relevance (whereas this is the central variable evaluated in the BIM approach). But this may not be the correct way to think about things, as some of the papers in Section 12.5 further discuss. The LM approach assumes that documents and expressions of information needs are objects of the same type, and assesses their match by importing the tools and methods of language modeling from speech and natural language processing. The resulting model is mathematically precise, conceptually simple, computationally tractable, and intuitively appealing. This seems similar to the situation with XML retrieval (Chapter 10): there the approaches that assume queries and documents are objects of the same type are also among the most successful.

On the other hand, like all IR models, you can also raise objections to the model. The assumption of equivalence between document and information need representation is unrealistic. Current LM approaches use very simple models of language, usually unigram models. Without an explicit notion of relevance, relevance feedback is difficult to integrate into the model, as are user preferences. It also seems necessary to move beyond a unigram model to accommodate notions of phrase or passage matching or Boolean retrieval operators. Subsequent work in the LM approach has looked at addressing some of these concerns, including putting relevance back into the model and allowing a language mismatch between the query language and the document language.

The model has significant relations to traditional tf-idf models. Term frequency is directly represented in tf-idf models, and much recent work has recognized the importance of document length normalization. The effect of doing a mixture of document generation probability with collection generation probability is a little like idf: terms rare in the general collection but common in some documents will have a greater influence on the ranking of documents. In most concrete realizations, the models share treating terms as if they were independent. On the other hand, the intuitions are probabilistic rather than geometric, the mathematical models are more principled rather than heuristic, and the details of how statistics like term frequency and document length are used differ. If you are concerned mainly with performance numbers, recent work has shown the LM approach to be very effective in retrieval experiments, beating tf-idf and BM25 weights. Nevertheless, there is



► **Figure 12.5** Three ways of developing the language modeling approach: (a) query likelihood, (b) document likelihood, and (c) model comparison.

perhaps still insufficient evidence that its performance so greatly exceeds that of a well-tuned traditional vector space retrieval system as to justify changing an existing implementation.

12.4 Extended language modeling approaches

In this section we briefly mention some of the work that extends the basic language modeling approach.

There are other ways to think of using the language modeling idea in IR settings, and many of them have been tried in subsequent work. Rather than looking at the probability of a document language model M_d generating the query, you can look at the probability of a query language model M_q generating the document. The main reason that doing things in this direction and creating a *document likelihood model* is less appealing is that there is much less text available to estimate a language model based on the query text, and so the model will be worse estimated, and will have to depend more on being smoothed with some other language model. On the other hand, it is easy to see how to incorporate relevance feedback into such a model: you can expand the query with terms taken from relevant documents in the usual way and hence update the language model M_q (Zhai and Lafferty 2001a). Indeed, with appropriate modeling choices, this approach leads to the BIM model of Chapter 11. The relevance model of Lavrenko and Croft (2001) is an instance of a document likelihood model, which incorporates pseudo-relevance feedback into a language modeling approach. It achieves very strong empirical results.

DOCUMENT LIKELIHOOD MODEL

Rather than directly generating in either direction, we can make a language model from both the document and query, and then ask how different these two language models are from each other. Lafferty and Zhai (2001) lay out these three ways of thinking about the problem, which we show in Figure 12.5, and develop a general risk minimization approach for document retrieval. For instance, one way to model the risk of returning a document d as relevant to a query q is to use the *Kullback-Leibler (KL) divergence* between their respective language models:

KULLBACK-LEIBLER DIVERGENCE

$$R(d;q) = KL(M_d \| M_q) = \sum_{t \in V} P(t|M_q) \log \frac{P(t|M_q)}{P(t|M_d)} \tag{12.14}$$

KL divergence is an asymmetric divergence measure originating in information theory, which measures how bad the probability distribution M_q is at modeling M_d (Cover and Thomas 1991, Manning and Schütze 1999). Lafferty and Zhai (2001) present results suggesting that a model comparison approach outperforms both query-likelihood and document-likelihood approaches. One disadvantage of using KL divergence as a ranking function is that scores are not comparable across queries. This does not matter for ad hoc retrieval, but is important in other applications such as topic tracking. Kraaij and Spitters (2003) suggest an alternative proposal which models similarity as a normalized log-likelihood ratio (or, equivalently, as a difference between cross-entropies).

Basic LMs do not address issues of alternate expression, that is, synonymy, or any deviation in use of language between queries and documents. Berger and Lafferty (1999) introduce translation models to bridge this query-document gap. A *translation model* lets you generate query words not in a document by translation to alternate terms with similar meaning. This also provides a basis for performing cross-language IR. We assume that the translation model can be represented by a conditional probability distribution $T(\cdot|\cdot)$ between vocabulary terms. The form of the translation query generation model is then:

TRANSLATION MODEL

$$P(q|M_d) = \prod_{t \in q} \sum_{v \in V} P(v|M_d) T(t|v) \tag{12.15}$$

The term $P(v|M_d)$ is the basic document language model, and the term $T(t|v)$ performs translation. This model is clearly more computationally intensive and we need to build a translation model. The translation model is usually built using separate resources (such as a traditional thesaurus or bilingual dictionary or a statistical machine translation system's translation dictionary), but can be built using the document collection if there are pieces of text that naturally paraphrase or summarize other pieces of text. Candidate examples are documents and their titles or abstracts, or documents and anchor-text pointing to them in a hypertext environment.

Building extended LM approaches remains an active area of research. In general, translation models, relevance feedback models, and model comparison approaches have all been demonstrated to improve performance over the basic query likelihood LM.

12.5 References and further reading

For more details on the basic concepts of probabilistic language models and techniques for smoothing, see either Manning and Schütze (1999, Chapter 6) or Jurafsky and Martin (2008, Chapter 4).

The important initial papers that originated the language modeling approach to IR are: (Ponte and Croft 1998, Hiemstra 1998, Berger and Lafferty 1999, Miller et al. 1999). Other relevant papers can be found in the next several years of SIGIR proceedings. (Croft and Lafferty 2003) contains a collection of papers from a workshop on language modeling approaches and Hiemstra and Kraaij (2005) review one prominent thread of work on using language modeling approaches for TREC tasks. Zhai and Lafferty (2001b) clarify the role of smoothing in LMs for IR and present detailed empirical comparisons of different smoothing methods. Zaragoza et al. (2003) advocate using full Bayesian predictive distributions rather than MAP point estimates, but while they outperform Bayesian smoothing, they fail to outperform a linear interpolation. Zhai and Lafferty (2002) argue that a two-stage smoothing model with first Bayesian smoothing followed by linear interpolation gives a good model of the task, and performs better and more stably than a single form of smoothing. A nice feature of the LM approach is that it provides a convenient and principled way to put various kinds of prior information into the model; Kraaij et al. (2002) demonstrate this by showing the value of link information as a prior in improving web entry page retrieval performance. As briefly discussed in Chapter 16 (page 353), Liu and Croft (2004) show some gains by smoothing a document LM with estimates from a cluster of similar documents; Tao et al. (2006) report larger gains by doing document-similarity based smoothing.

Hiemstra and Kraaij (2005) present TREC results showing a LM approach beating use of BM25 weights. Recent work has achieved some gains by going beyond the unigram model, providing the higher order models are smoothed with lower order models (Gao et al. 2004, Cao et al. 2005), though the gains to date remain modest. Spärck Jones (2004) presents a critical viewpoint on the rationale for the language modeling approach, but Lafferty and Zhai (2003) argue that a unified account can be given of the probabilistic semantics underlying both the language modeling approach presented in this chapter and the classical probabilistic information retrieval approach of Chapter 11. The Lemur Toolkit (http://www.lemurproject.org/) provides a flexible open source framework for investigating language modeling approaches to IR.