12 Language models for information retrieval

In the traditional probabilistic approach to IR, the user has an information need, and determines a query $q$ which is run against documents $d$, and we try to determine the probability of relevance $P(R|q,d)$. The original language modeling approach bypasses overtly modeling the concept of relevance. It 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: $P(q|M_d)$.

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 your 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.

What do we mean by a document model generating a query? A traditional generative model of 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.

If instead each node has a probability distribution over generating different words, we have a language model. A (stochastic or probabilistic) language model is a function that puts a probability measure over strings drawn from some vocabulary. One simple kind of language model is equivalent to a probabilistic finite automaton consisting of just a single node with a single probability distribution of producing different words, as shown in Figure 12.2, coupled with a probability of stopping when in a finish 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. To find the probability of a word sequence, we just multiply the probabilities
I wish
I wish I wish
I wish I wish I wish
I wish I wish I wish I wish I wish I wish

*wish I wish

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

◮ Figure 12.2 A one-state finite automaton that acts as a unigram language model together with a partial specification of the state emission probabilities.

which it gives to each word in the sequence. For example,

\[
(12.1) \quad P(\text{frog said that toad likes frog}) = 0.01 \times 0.03 \times 0.04 \times 0.02 \times 0.01 = 0.0000000024
\]

Here we omit the probability of stopping after frog. An explicit stop probability is needed for the finite automaton to generate and give probabilities to finite strings, but we will in general omit mention of it, since, if fixed, it does not alter the ranking of documents.

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 words, as shown in the example. The language model that gives the higher probability to the sequence of words is more likely to have generated the word sequence. For the sequence shown, we get:
Figure 12.3 Partial specification of two unigram language models.

<table>
<thead>
<tr>
<th>Model $M_1$</th>
<th>Model $M_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>the</strong> 0.2</td>
<td><strong>the</strong> 0.15</td>
</tr>
<tr>
<td><strong>a</strong> 0.1</td>
<td><strong>a</strong> 0.12</td>
</tr>
<tr>
<td><strong>frog</strong> 0.01</td>
<td><strong>frog</strong> 0.0002</td>
</tr>
<tr>
<td><strong>toad</strong> 0.01</td>
<td><strong>toad</strong> 0.0001</td>
</tr>
<tr>
<td><strong>said</strong> 0.03</td>
<td><strong>said</strong> 0.03</td>
</tr>
<tr>
<td><strong>likes</strong> 0.02</td>
<td><strong>likes</strong> 0.04</td>
</tr>
<tr>
<td><strong>that</strong> 0.04</td>
<td><strong>that</strong> 0.04</td>
</tr>
<tr>
<td><strong>dog</strong> 0.005</td>
<td><strong>dog</strong> 0.01</td>
</tr>
<tr>
<td><strong>cat</strong> 0.003</td>
<td><strong>cat</strong> 0.015</td>
</tr>
<tr>
<td><strong>monkey</strong> 0.001</td>
<td><strong>monkey</strong> 0.002</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(12.2) $s$ 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

$P(s|M_1) = 0.00000000000048$
$P(s|M_2) = 0.00000000000000384$

and we see that $P(s|M_1) > P(s|M_2)$.

How do people build probabilities over word sequences? We can always use the chain rule to decompose the probability of a sequence of events into the probability of each successive events conditioned on earlier events:

$$P(w_1w_2w_3w_4) = P(w_1)P(w_2|w_1)P(w_3|w_2w_1)P(w_4|w_1w_2w_3)$$

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

$$P_{\text{uni}}(w_1w_2w_3w_4) = P(w_1)P(w_2)P(w_3)P(w_4)$$

Under this model the order of words is irrelevant, and so such models are sometimes called “bag of words” models as discussed in Chapter 6 (page 88). There are many more complex kinds of language models, such as bigram language models, which condition on the previous word,

$$P_{\text{bi}}(w_1w_2w_3w_4) = P(w_1)P(w_2|w_1)P(w_3|w_2)P(w_4|w_3)$$

and even more complex grammar-based language models such as probabilistic context-free grammars. However, most language-modeling work in IR...
has used unigram language models, and IR is probably not the most productive place to try using complex language models, since IR does not directly depend on the structure of sentences to the extent that other tasks like speech recognition do. Moreover, since, as we shall see, IR language models are frequently estimated from a single document, there is often not enough training data and losses from sparseness outweigh any gains from richer models.

The fundamental problem in designing language models is that we generally 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 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. However, we pretend that the document $d$ is only a representative sample of text drawn from a model distribution, we estimate a language model from this sample, use that model to calculate the probability of observing any word sequence, and finally rank documents according to their probability of generating the query.

### 12.1 The Query Likelihood Model

#### 12.1.1 Using Query Likelihood Language Models in IR

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, we have:

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)}$$

$P(q)$ is the same for all documents, and so can be ignored. The prior $P(d)$ 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 $q$ given by 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 202), where the documents are the classes, each treated in the estimation as a separate “language”. Under this model, we have that:

\[
P(q|M_d) = \prod_{w \in V} P(w|M_d)^{c(w)}
\]

(12.3)

Usually a unigram estimate of words is used in IR. There is some work on bigrams, paralleling the discussion of van Rijsbergen in Chapter 11 (page 176), but it hasn’t been found very necessary. While modeling term cooccurrence should improve estimates somewhat, IR is different to tasks like speech recognition: word order and sentence structure are not very necessary to modeling the topical content of documents.

For retrieval based on a probabilistic language model, we treat the generation of queries as a random process. The approach is to:

1. Infer a language model for each document.
2. Estimate the probability of generating the query according to each of these models.
3. Rank the documents according to these probabilities.

The intuition 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. Collection statistics are an integral part of the language model, rather than being used heuristically as in many other approaches.

12.1.2 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 language model \( M_d \) of document \( d \) using maximum likelihood estimation (MLE) and given the unigram assumption is:

\[
\hat{P}(q|M_d) = \prod_{t \in q} \hat{P}_{\text{mle}}(t|M_d) = \prod_{t \in q} \frac{tf_{t,d}}{dl_d}
\]

(12.4)

where \( M_d \) is the language model of document \( d \), \( tf_{t,d} \) is the (raw) term frequency of term \( t \) in document \( d \), and \( dl_d \) is the number of tokens in document \( d \).

The classic problem with such models is one of estimation (the \( \hat{\cdot} \) is used above to stress that the model is estimated). 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. This may or may not be undesirable: it is partly 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. But regardless of one’s 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 there one occurrence was partly by chance.

This problem of insufficient data and a zero probability preventing any non-zero match score for a document can spell disaster. We need to smooth probabilities: to discount non-zero probabilities and to give some probability mass to unseen things. There’s a wide space of approaches to smoothing probability distributions to deal with this problem, such as adding a number (1, 1/2, or a small \( \epsilon \)) to counts and renormalizing, discounting, Dirichlet priors and interpolation methods. A simple idea that works well in practice is to use a mixture between the document multinomial and the collection multinomial distribution.

The general approach is that a non-occurring term is possible in a query, but no more likely than would be expected by chance from the whole collection. That is, if \( tf_{t,d} = 0 \) then

\[
\hat{P}(t|M_d) \leq \frac{cf_t}{cs}
\]

where \( cf_t \) is the raw count of the term in the collection, and \( cs \) is the raw size (number of tokens) of the entire collection. We can guarantee this by mixing together a document-specific model with a whole collection model:

\[
\hat{P}(w|d) = \lambda \hat{P}_{\text{mle}}(w|M_d) + (1 - \lambda) \hat{P}_{\text{mle}}(w|M_c)
\]

(12.5)

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. Correctly setting \( \lambda \) is important to the good performance of this model. A high value of lambda makes the search “conjunctive-like” – suitable for short queries. A low value is more suitable for long queries. One can tune \( \lambda \) to optimize performance, including not having it be constant but a function of document size.

So, the general formulation of the basic LM for IR is:

\[
P(q|d) \propto P(d) \prod_{t \in q} \left( (1 - \lambda) P(t|M_c) + \lambda P(t|M_d) \right)
\]
The equation represents the probability that the document that the user had in mind was in fact this one.

**Example 12.1:** Suppose the document collection contains two documents:

- $d_1$: Xyz reports a profit but revenue is down
- $d_2$: Qrs 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:

\[
P(q|d_1) = \frac{[(1/8 + 2/16)/2] \times [(1/8 + 1/16)/2]}{2} = \frac{1/8 \times 3/32}{2} = 3/256
\]

\[
P(q|d_2) = \frac{[(1/8 + 2/16)/2] \times [(0/8 + 1/16)/2]}{2} = \frac{1/8 \times 1/32}{2} = 1/256
\]

So, the ranking is $d_1 > d_2$.

### 12.2 Ponte and Croft’s Experiments

Ponte and Croft (1998) present the first experiments on the language modeling approach to information retrieval. Their basic approach where each document defines a language model 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 evidence from text categorization (see Chapter 13) 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 evaluated on 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.

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

**Figure 12.4** Results of a comparison of tf-idf to 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 Wilcoxon signed rank test. While the language modeling approach always does better in these experiments, note that where the approach shows significant gains is at higher levels of recall.

As Ponte and Croft (1998) emphasize, the language modeling approach to IR provides a different form of 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. It has achieved very good retrieval results. Compared to other probabilistic approaches, such as BIM from Chapter 11, the main difference is that the LM approach attempts to do away with explicitly modeling relevance (whereas this is the central variable evaluated in the BIM approach). The LM approach assumes that documents and expressions of information problems 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.

On the other hand, like all IR models, one can also raise objections to the model. The assumption of equivalence between document and information problem 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 or priors over document relevance. It also isn’t easy to see how to accommodate notions of phrasal matching 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 some relation to traditional tf-idf models. Term frequency is directly 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 one is concerned mainly with performance numbers, while the LM approach has been proven quite effective in retrieval experiments, there is little evidence that its performance exceeds a well-tuned traditional ranked retrieval system.

12.4 Extended language modeling approaches

In this section we briefly note some of the work that has taken place that extends the basic language modeling approach.

There are other ways that one could 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 generating the query, you can look at the probability of a query language model generating the document. The main reason that doing things in this direction is less appealing is that there is much less text available to estimate a query language model, 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: one can expand the query with terms taken from relevant documents in the usual way and hence update the query language model (Zhai and Lafferty 2001a). Indeed, with appropriate modeling choices, this approach leads to the BIR model of Chapter 11.

Rather than directly generating in either direction, one 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
For instance, one way to model the risk of returning a document \( d \) as relevant to a query \( q \) is to use the \textit{Kullback-Leibler divergence} between their respective language models:

\[
R(d; q) = KL(d \| q) = \sum_w P(w|M_q) \log \frac{P(w|M_q)}{P(w|M_d)}
\]

This asymmetric divergence measure coming from information theory shows how bad the probability distribution \( M_q \) is at modeling \( M_d \). Lafferty and Zhai (2001) present results suggesting that a model comparison approach outperforms both query-likelihood and document-likelihood approaches.

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-lingual IR. Assuming a probabilistic lexicon \( \text{Lex} \) which gives information on synonymy or translation pairs, the nature of the translation query generation model is:

\[
P(q|M_d) = \prod_w \sum_v P(v|M_d) T(w|v)
\]
The left term on the right hand side is the basic document language model, and the right term performs translation. This model is clearly more computationally intensive and one needs to build a translation model, usually using separate resources (such as a dictionary or a statistical machine translation system’s lexicon).

12.5 References and further reading

For more details on the basic concepts and smoothing methods for probabilistic language models, see either Manning and Schütze (1999, Ch. 6) or Jurafsky and Martin (2000, Ch. 6).

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) reviews one prominent thread of work on using language modeling approaches for TREC tasks. System implementers should consult Zhai and Lafferty (2001b), Zaragoza et al. (2003) for detailed empirical comparisons of different smoothing methods for language models in IR. Additionally, 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). For a critical viewpoint on the rationale for the language modeling approach, see Spärck Jones (2004).