CS4051- Information Retrieval

Muhammad Rafi
April 15, 2024

Language Model for IR

Agenda

- IR Approaches (IR models)
- Language Model –Basic
- Examples
- Types of Language Model
- Maximum Likelihood Estimates
- Extended Language Models
- Vector Space Vs. Language Model
- Conclusion

IR Approaches

- Boolean model
  - Based on the notion of sets
  - Documents are retrieved only if they satisfy Boolean conditions specified in the query
  - Does not impose a ranking on retrieved documents
  - Exact match
- Vector space model
  - Based on geometry, the notion of vectors in high dimensional space
  - Documents are ranked based on their similarity to the query (ranked retrieval)
  - Best/partial match

IR Approaches

- Probabilistic models
  - Based on the notion of probabilities present in the text
  - Document are ranked based on the probabilities that the text present in document based on query terms.
  - Best/partial match
- Language models
  - Based on the notion of probabilities and processes for generating text
  - Documents are ranked based on the probability that they generated the query
  - Best/partial match

Language Model –Basic Idea

- 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.

LM vs. PM

- 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
- 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: $P(q|M_d)$.

Language Model

- 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 S:

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

- 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 $\sum_{t \in V} P(t) = 1$,

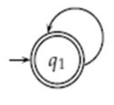
Example 1

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

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

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

Example 1



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

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.

Example 2

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 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:

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

$P(s|M_1) = 0.00000000000048$
$P(s|M_2) = 0.000000000000000384$

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 239).

Example 2

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.

Language Model

- Each document is treated as (the basis for) a language model.
- Given a query *q*
- Rank documents based on *P*(*d*|*q*)

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

- *P*(*q*) is the same for all documents, so ignore
- *P*(*d*) is the prior – often treated as the same for all *d*
    - But we can give a prior to “high-quality” documents, e.g., those with high PageRank.
- *P*(*q*|*d*) is the probability of *q* given *d.*
- So to rank documents according to relevance to *q*, ranking according to *P*(*q*|*d*) and *P*(*d*|*q*) is equivalent.

Language Model

- p(d) = prior probability of hypothesis A – PRIOR
- p(q) = prior probability of query q – EVIDENCE
- p(q|d) = probability of q given d - LIKELIHOOD
- P(d|q) = probability of d given q – POSTERIOR

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

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 202), 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

Query Likelihood Model

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 to use the multinomial unigram language model, which is equivalent to a multinomial naive Bayes model (page 243), where the documents are the classes, each treated in the estimation as a separate "language." Under this model, we have that:

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

Query Likelihood Model

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

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 henceforth ignore because 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.

Query Likelihood Model

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.

Query Likelihood Model

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

estimation (MLE) and the unigram assumption is:

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

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.

Query Likelihood Model

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,

Query Likelihood Model

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 $\mathrm{tf}_{t,d} = 0$ then

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

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:

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

Query Likelihood Model

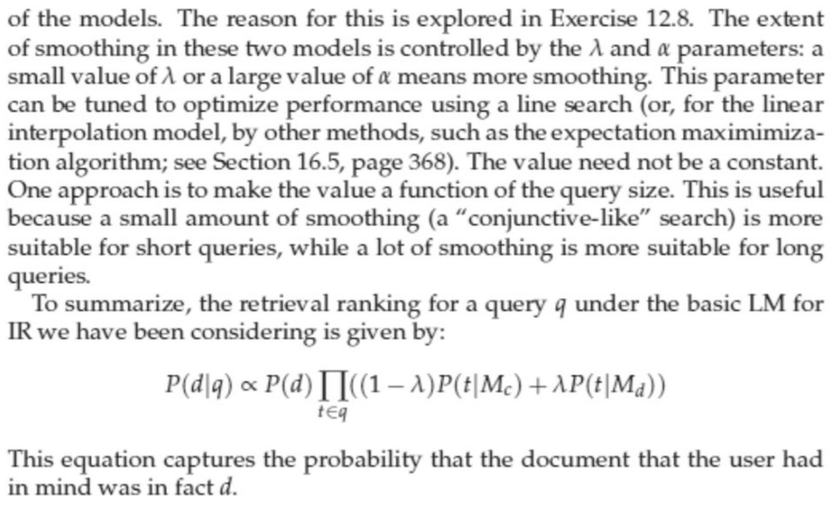
where $0 < \lambda < 1$ and M_c is a language model built from the entire document collection. This mixes the probability from the document with the general collection frequency of the word. Such a model is referred to as a *linear interpolation* language model.[5] Correctly setting λ is important to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

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:

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

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.

Query Likelihood Model



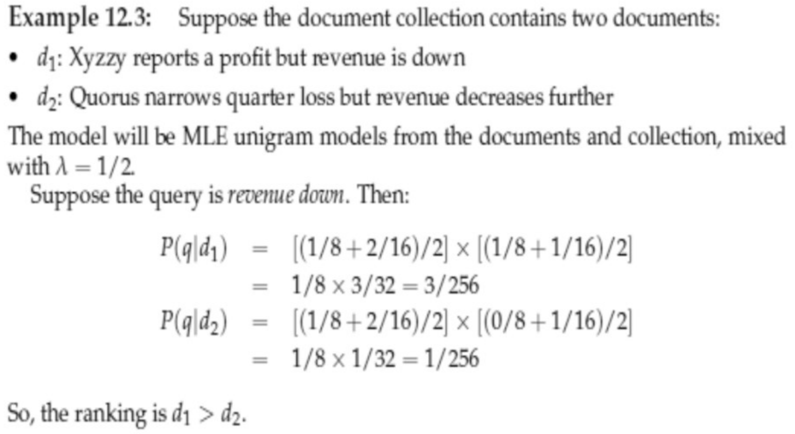
of the models. The reason for this is explored in Exercise 12.8. The extent of smoothing in these two models is controlled by the λ and α parameters: a small value of λ or a large value of α means more smoothing. This parameter can be tuned to optimize performance using a line search (or, for the linear 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:

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

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

Example



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:

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

So, the ranking is $d_1 > d_2$.

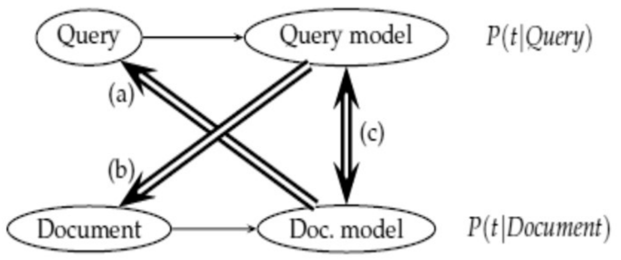
Smoothing- Language Model

- Jelinek-Mercer smoothing
- Dirichlet smoothing
- Good-Turing smoothing

Extended Language Model

- 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. (Smoothing factors)

Extended Language Model



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

Kullback Leibler Divergence Model

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).

KL Divergence Model

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

LM vs. VSM

- LMs have some things in common with vector space models.
- Term frequency is directed in the model.
  - But it is not scaled in LMs.
- Probabilities are inherently “length-normalized”.
  - Cosine normalization does something similar for vector space.
- Mixing document and collection frequencies has an effect similar to idf.
  - Terms rare in the general collection, but common in some documents will have a greater influence on the ranking.

LM vs. VSM

- LMs vs. vector space model: commonalities
  - Term frequency is directly in the model.
  - Probabilities are inherently "length-normalized".
  - Mixing document and collection frequencies has an effect similar to idf.
- LMs vs. vector space model: differences
  - LMs: based on probability theory
  - Vector space: based on similarity, a geometric/ linear algebra notion
  - Collection frequency vs. document frequency
  - Details of term frequency, length normalization etc.

Assumptions of LM

- Simplifying assumption: Queries and documents are objects of same type. Not true!
  - There are other LMs for IR that do not make this assumption.
  - The vector space model makes the same assumption.
- Simplifying assumption: Terms are conditionally independent.
  - Again, vector space model (and Naive Bayes) makes the same assumption.
- Cleaner statement of assumptions than vector space
- Thus, better theoretical foundation than vector space
  - … but "pure" LMs perform much worse than "tuned" LMs.