Introduction to

Information Retrieval

Language Models for IR

Overview

- Introduction
- Language models
- The query likelihood model
- Ponte and Croft’s Experiments
- Language modeling versus other approaches in IR
- Extended language modeling approaches

Introduction

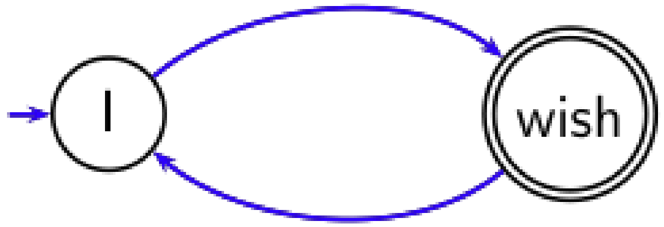
- A good query - is to think of words that would likely appear in a relevant document
- The language modeling approach to IR directly models this idea:
  - a document is a good match to a query if the document model is likely to generate the query, which will in turn happen if the document contains the query words often.
- Instead of 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 , the basic language modeling approach instead builds a probabilistic language model *Md* from each document *d*, and ranks documents based on the probability of the model generating the query: $P(q|Md)$.

Finite automata and language models

- What do we mean by a document model generating a query?
- *Generative model* of a language - formal language - can be used either to recognize or to generate strings.
- The full set of strings that can be generated is called the *language* of the automaton.

What is a language model?

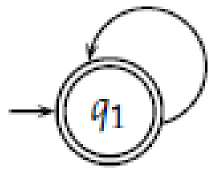
We can view a finite state automaton as a deterministic language model.



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.

I wish I wish I wish I wish . . . Cannot generate: “wish I wish” or “I wish I”. Our basic model: each document was generated by a different automaton like this except that these automata are probabilistic.

Language Model



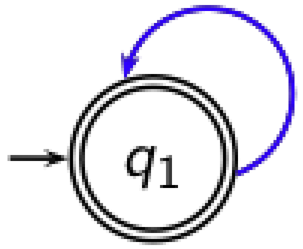
$P(\text{STOP}|q_1) = 0.2$

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

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:

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

A probabilistic language model



w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

- This is a one-state probabilistic finite-state automaton – a unigram language model – and the state emission distribution for its one state q_1. STOP is not a word, but a special symbol indicating that the automaton stops. Query string is “frog said that toad likes frog “STOP

- 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

- 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,

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

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.

To compare two models

- Nevertheless, most of the time, we will omit to include STOP and (1 − STOP) probabilities .
- LIKELIHOOD RATIO:
- 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.

A different language model for each document

Example

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

- s: frog said that toad likes that dog
- $p(s|M_1) = 0.01 \times 0.03 \times 0.04 \times 0.01 \times 0.02 \times 0.04 \times 0.005 = 0.0000000000048$
- $p(s|M_2) = 0.000000000000000384$
- $p(s|M_1) > p(s|M_2)$: M_1 is more likely to generate s

Types of language models

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

Basic LM using chain rule

- $P(t1\ t2\ t3\ t4) = P(t1)\ P(t2|t1)\ P(\ t3|t1t2)\ P(t4|t1t2t3)$

The probability of a sequence $s = t_1 t_2 \ldots t_n$ is

$$\begin{aligned} p(t_1 \ldots t_n) &= P(t_1)P(t_2|t_1)P(t_3|t_1 t_2)\ldots P(t_n|t_1 \ldots t_{n-1}) \\ &= P(t_1)P(t_2|t_1)\prod_{i=3}^{n} P(t_i|t_1 \ldots t_{i-1}) \end{aligned}$$

- Unigram language model: the condition parts are completely ignored

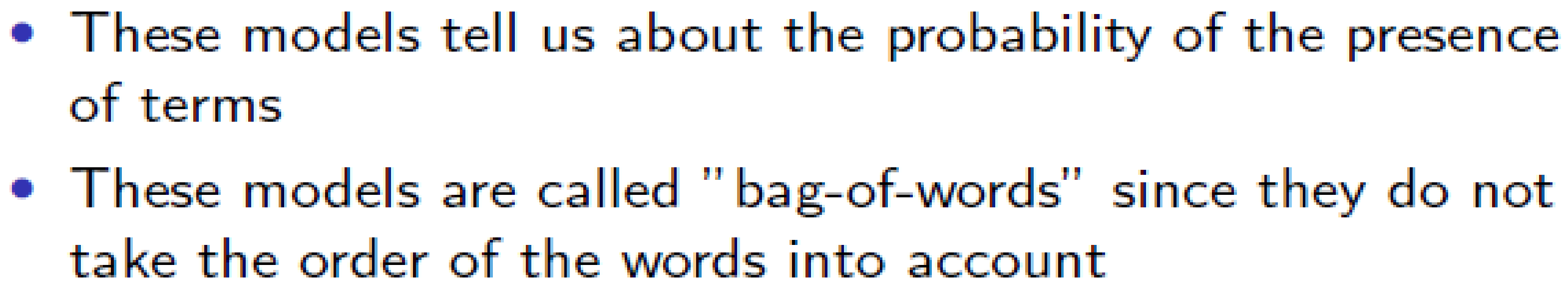
$$p(t_1 \ldots t_n) = \prod_{i=1}^{n} P(t_i)$$

- Bigram language model: the probabilities are estimated based on the previous term

$$p(t_1 \ldots t_n) = P(t_1) \prod_{i=2}^{n} P(t_i | t_{i-1})$$

- 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.
- 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.



- These models tell us about the probability of the presence of terms
- These models are called "bag-of-words" since they do not take the order of the words into account

Multinomial distributions over words

- From the point of view of a unigram language model, a document is a bag of words
- Even though there is no conditioning on preceding context, this model nevertheless still gives the probability of a particular ordering of terms.
- So, really, we have a *multinomial distribution* MULTINOMIAL over words.
- The probability of a document d of length d is:

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

Here, $L_d = \sum_{1\leq i\leq M} \mathrm{tf}_{t_i,d}$ 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

However, we can 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.

$$P(d) = \prod_{i=1}^{M} P(t_i)^{\mathrm{tf}_{t_i,d}}$$

probability of that document appearing in the corpus. So, generally, you could have the probability Md/NMd/N, where MdMd is the number of times the specific document appears

QUERY LIKELIHOOD MODEL

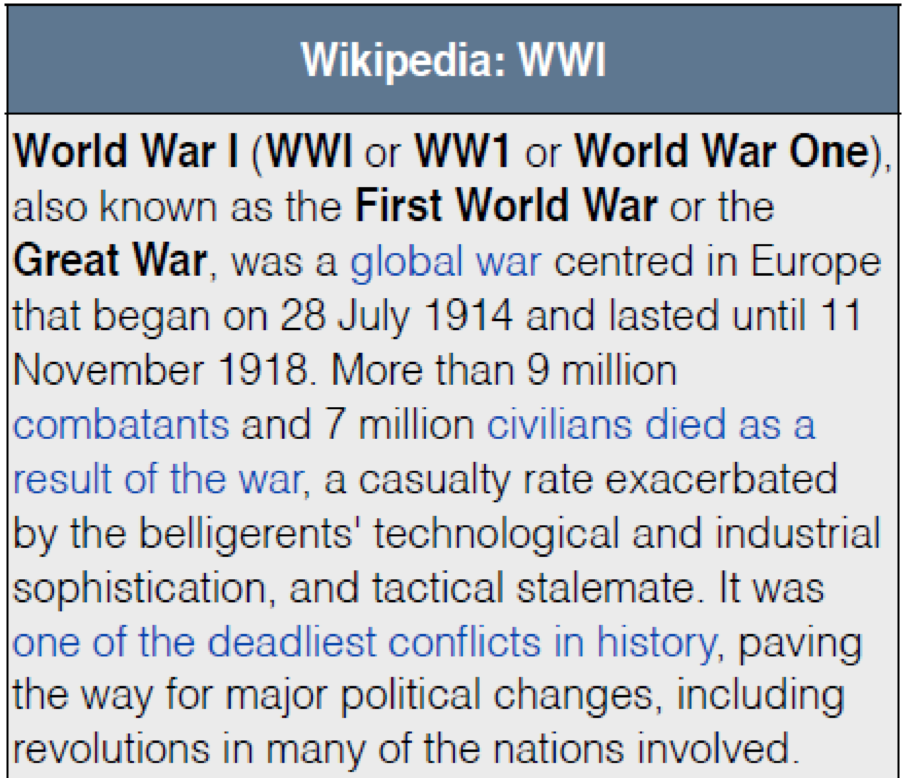
- The original and basic method for using language models in IR is the *query likelihood model*.
- We construct from each document *d* in the collection a language model M_d.
- Our goal is to rank documents by $P(d|q)$, where the probability of a document is interpreted as the likelihood that it is relevant to the query. Using Bayes rule we have: $P(d|q) = P(q|d)P(d)/P(q)$
- $P(q)$ 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.
- Probability of a document appearing in the corpus, P(d), is the probability Md/N. where Md is the number of times the specific document appears in the corpus and N is the total number of documents

$$P(d|q) \approx P(q|d)$$

The probability $P(q|d)$ measures the extent to which the query q can be generated from the document d

Query Likelihood Model

- In the *Query Likelihood Model*, we rank documents based on the probability of generating the query from the documents' language models.



Wikipedia: WWI

World War I (**WWI** or **WW1** or **World War One**), also known as the **First World War** or the **Great War**, was a global war centred in Europe that began on 28 July 1914 and lasted until 11 November 1918. More than 9 million combatants and 7 million civilians died as a result of the war, a casualty rate exacerbated by the belligerents' technological and industrial sophistication, and tactical stalemate. It was one of the deadliest conflicts in history, paving the way for major political changes, including revolutions in many of the nations involved.

Query: "deadliest war in history"

Term	P(w|D)	log P(w|D)
deadliest	1/94 = 0.011	-1.973
war	6/94 = 0.063	-1.195
in	3/94 = 0.032	-1.496
history	1/94 = 0.011	-1.973
	Π = 2.30e-7	**Σ = -6.637**

Query Likelihood Model

Wikipedia: Taiping Rebellion

The **Taiping Rebellion** was a massive civil war in southern China from 1850 to 1864, against the ruling Manchu Qing dynasty. It was a millenarian movement led by Hong Xiuquan, who announced that he had received visions, in which he learned that he was the younger brother of Jesus. At least 20 million people died, mainly civilians, in one of the deadliest military conflicts in history.

Query: "deadliest war in history"

Term	P(w|D)	log P(w|D)
deadliest	1/56 = 0.017	-1.748
war	1/56 = 0.017	-1.748
in	2/56 = 0.035	-1.447
history	1/56 = 0.017	-1.748
	Π = 2.56e-8	**Σ = –6.691**

The Retrieval Procedure

- Infer a language model for each document d_i, $i = 1, \ldots, n$
- For each query q, estimate the probability of generating the query given each of the document models, $P(q|M_{d_i})$
- Rank the documents according to the 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
- The most common way to do this is using the multinomial unigram language model. Under this model, we have

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

How to compute *P*(*q*|*d*)

- We will make the same conditional independence assumption as for Naive Bayes.

$$P(q|M_d) = P(\langle t_1, \ldots, t_{|q|}\rangle | M_d) = \prod_{1 \leq k \leq |q|} P(t_k|M_d)$$

(|*q*|: length ofr *q*; t_k : the token occurring at position k in q)

- This is equivalent to:

$$P(q|M_d) = \prod_{\text{distinct term } t \text{ in } q} P(t|M_d)^{\mathrm{tf}_{t,q}}$$

- $\mathrm{tf}_{t,q}$: term frequency (# occurrences) of *t* in *q*
- Multinomial model (omitting constant factor)

Estimating the query generation probability : Parameter estimation

- Missing piece: Where do the parameters $P(t|M_d)$. come from?
- Start with maximum likelihood estimates

$$\hat{P}(t|M_d) = \frac{\mathrm{tf}_{t,d}}{|d|}$$

($|d|$: length of d; $\mathrm{tf}_{t,d}$: # occurrences of t in d)

- A single t with $P(t|M_d) = 0$ will make $P(q|M_d) = \prod P(t|M_d)$ zero.
- For example, for query [Michael Jackson top hits] a document about "top songs" (but not using the word "hits") would have $P(t|M_d) = 0$.
- We need to smooth the estimates to avoid zeros.

Smoothing

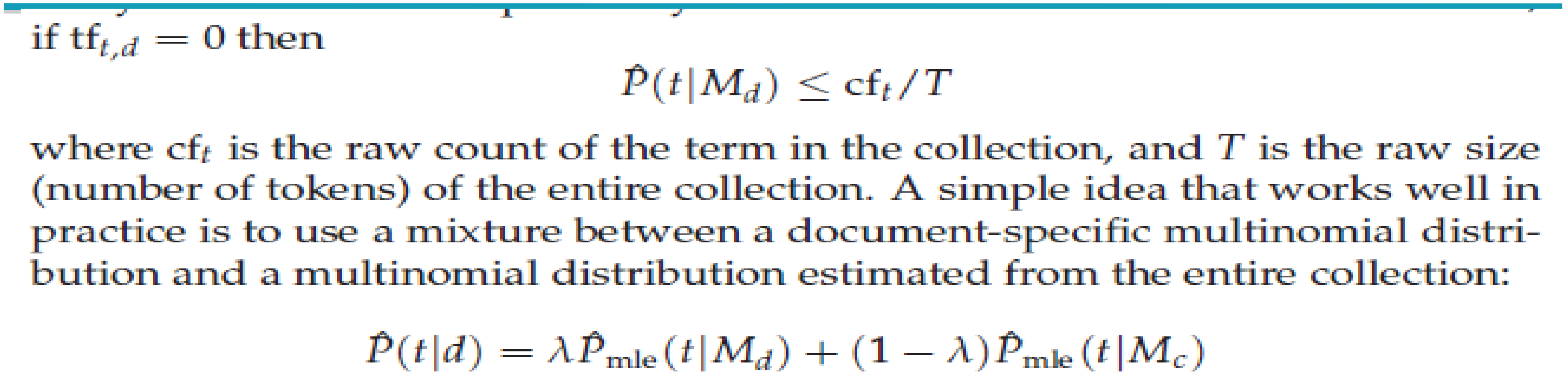
- Key intuition: A non occurring term is possible (even though it didn't occur), . . .

- Notation: M_c: the collection model; cf_t: the number of occurrences of *t* in the collection; $T = \sum_t \mathrm{cf}_t$: the total number of tokens in the collection.

$$\hat{P}(t|M_d) = \frac{\mathrm{tf}_{t,d}}{|d|}$$

- We will use $\hat{P}(t|M_c)$ to "smooth" $P(t|d)$ away from zero.

Mixture model- *linear interpolation* language model



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:

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

where $0 < \lambda < 1$ and *Mc* 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.

- Mixes the probability from the document with the general collection frequency of the word.
- High value of λ: "conjunctive-like" search – tends to retrieve documents containing all query words.
- Low value of λ: more disjunctive, suitable for long queries
- Correctly setting λ is very important for good performance.

Mixture model: Summary

$$P(q|d) \propto \prod_{1 \leq k \leq |q|} (\lambda P(t_k|M_d) + (1-\lambda)P(t_k|M_c))$$

- What we model: The user has a document in mind and generates the query from this document.
- The equation represents the probability that the document that the user had in mind was in fact this one.

Example

- Collection: d_1 and d_2
- d_1 : Jackson was one of the most talented entertainers of all time
- d_2: Michael Jackson anointed himself King of Pop
- Query q: Michael Jackson
- Use mixture model with $\lambda = 1/2$
- $P(q|d_1) = [(0/11 + 1/18)/2] \cdot [(1/11 + 2/18)/2] \approx 0.003$
- $P(q|d_2) = [(1/7 + 1/18)/2] \cdot [(1/7 + 2/18)/2] \approx 0.013$
- Ranking: $d_2 > d_1$

Exercise:

- Collection: d_1 and d_2
- d_1 : Xerox reports a profit but revenue is down
- d_2: Lucene narrows quarter loss but decreases further
- Query q: revenue down
- Use mixture model with $\lambda = 1/2$
- $P(q|d_1) = [(1/8 + 2/16)/2] \cdot [(1/8 + 1/16)/2] = 1/8 \cdot 3/32 = 3/256$
- $P(q|d_2) = [(1/8 + 2/16)/2] \cdot [(0/8 + 1/16)/2] = 1/8 \cdot 1/32 = 1/256$
- Ranking: $d_2 > d_1$

- We have a collection of two documents:
  - d_1 : How the coronavirus took advantage of humanity's essential weakness ?
  - d_2 : The economy is drastically affected by the coronavirus pandemic .
- Let $q = \texttt{coronavirus advantage}$ be a query, $\lambda = \frac{1}{2}$, and $P(d_1) = P(d_2)$
- Since the prior probabilities are equal, we ignore them
- $P(q|d_1) = \frac{1}{2}(\frac{1}{10} + \frac{2}{20})^1 \times \frac{1}{2}(\frac{1}{10} + \frac{1}{20})^1 = \frac{3}{400} = 0.0075$
- $P(q|d_2) = \frac{1}{2}(\frac{1}{10} + \frac{2}{20})^1 \times \frac{1}{2}(\frac{0}{10} + \frac{1}{20})^1 = \frac{1}{400} = 0.0025$
- $P(q|d_1) > P(q|d_2)$

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 $d_1 > d_2$.

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

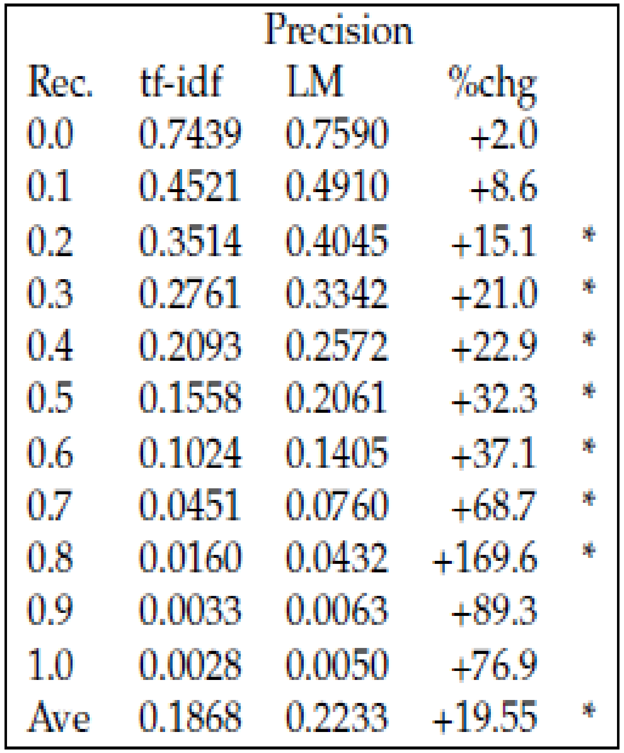
Build a query likelihood language model for this document collection. Assume a mixture model between the documents and the collection, with both weighted at 0.5. Maximum likelihood estimation (mle) is used to estimate both as unigram models. Work out the model probabilities of the queries click, shears, 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?

Ponte and Croft's Experiments

- Ponte and Croft (1998) present the first experiments on the language modeling approach to information retrieval.
- Their approach is **the query likelihood model approach (not the mixture model)**



	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	*

Results of a comparison of tf-idf with language modeling (LM) term weighting by Ponte and Croft (1998). The table gives an evaluation according to 11-point average precision

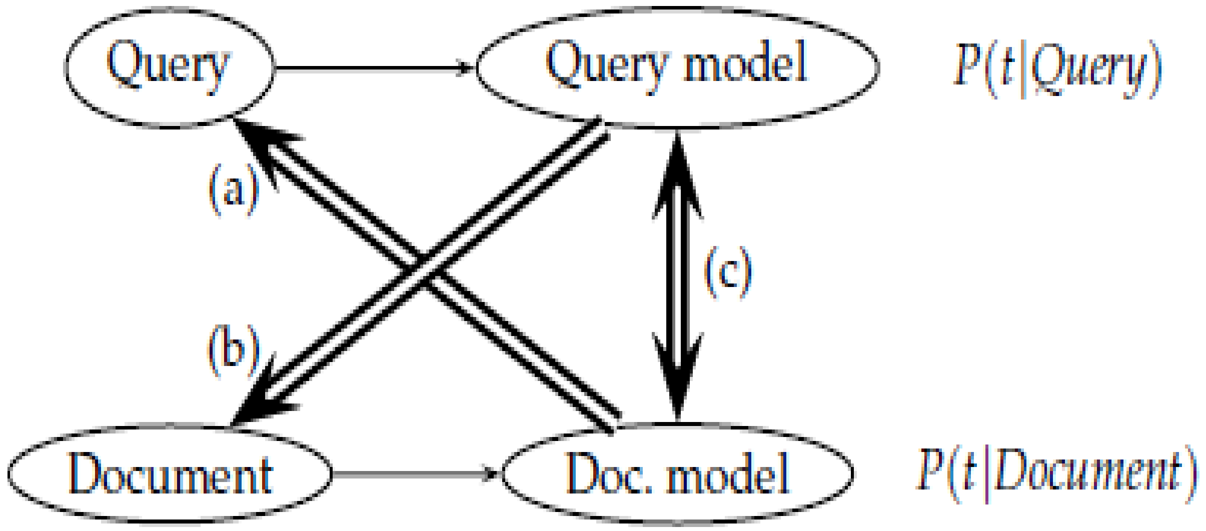
Language modeling versus other approaches in IR

- LM approach does away with explicitly modeling relevance.
- The LM approach assumes that documents and expressions of information needs are objects of the same type
- The model has significant relations to traditional tf-idf models. 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.
- The resulting model is mathematically precise, conceptually simple, computationally tractable, and intuitively appealing.
- Objections:
  - The assumption of equivalence between document and information need representation is unrealistic
  - Current LM approaches use very simple models of language, usually unigram models
- Recent work has shown the LM approach to be very effective in retrieval experiments, beating tf-idf and other weights

Extended language modeling approaches

- There are other ways to think of using the language modeling idea in IR settings.
- It is easy to incorporate relevance feedback into a model: you can expand the query with terms taken from relevant documents in the usual way and hence update the language model *Mq.*
- Rather than looking at the probability of a document language model *Md* generating the query, you can look at the probability of a query language model *Mq* generating the document.
  - creating a *document likelihood model* is less appealing, there is much less text available to estimate a language model based on the query text, and so the model will be worse estimated.
- 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

To develop a general risk minimization approach for document retrieval.



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

- 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:

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

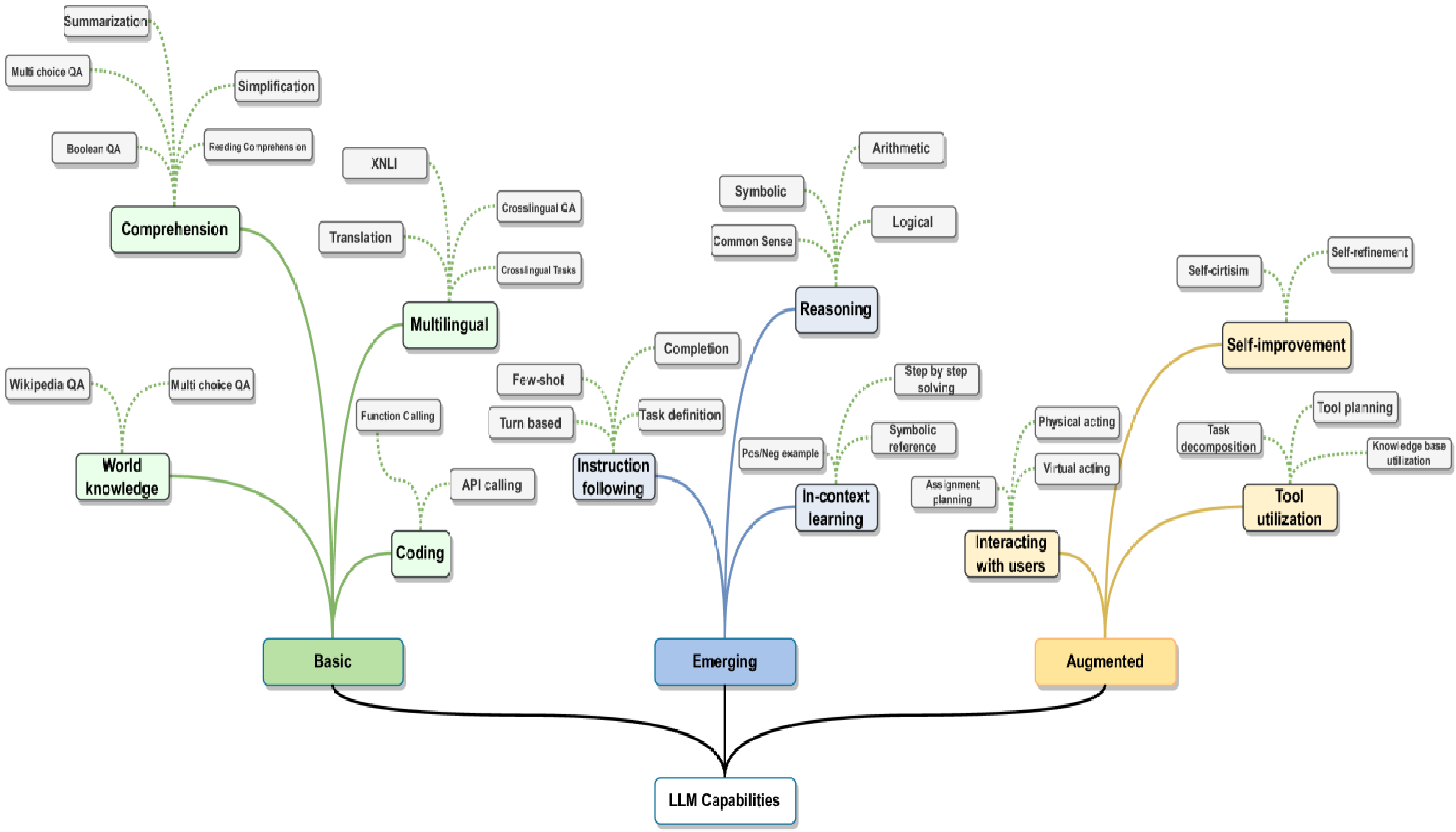
- One disadvantage of using KL divergence as a ranking function is that scores are not comparable across queries. This does not matter for adhoc retrieval, but is important in other applications such as topic tracking.

- Basic LMs do not address issues of alternate expression, that is, synonyms, or any deviation in use of language between queries and documents.
- Berger and Lafferty (1999) introduced 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.
- The form of the translation query generation model is then:

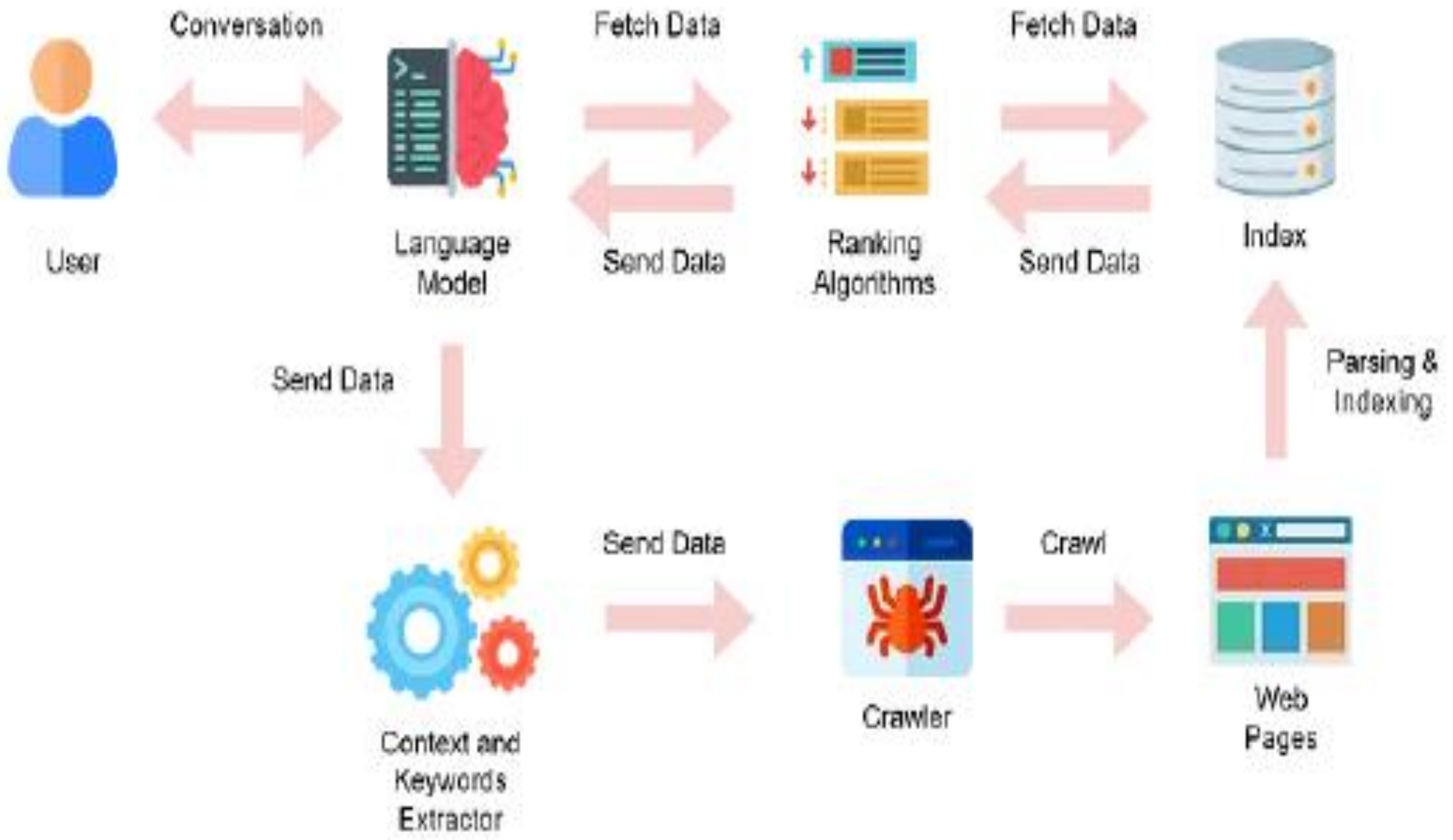
$$P(q|M_d) = \prod_{t \in q} \sum_{v \in V} P(v|M_d)T(t|v)$$

- The term *P*(*v*|*Md*) is the basic document language model, and the term *T*(*t*|*v*) performs translation.
- 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

LLMs


Summarization
Multi choice QA
Simplification
Boolean QA
Reading Comprehension
Comprehension
XNLI
Crosslingual QA
Translation
Crosslingual Tasks
Multilingual
Wikipedia QA
Multi choice QA
World knowledge
Function Calling
API calling
Coding
Arithmetic
Symbolic
Logical
Common Sense
Reasoning
Completion
Few-shot
Turn based
Task definition
Instruction following
Step by step solving
Symbolic reference
Pos/Neg example
In-context learning
Self-cirtisim
Self-refinement
Self-improvement
Physical acting
Virtual acting
Assignment planning
Interacting with users
Tool planning
Task decomposition
Knowledge base utilization
Tool utilization
Basic
Emerging
Augmented
LLM Capabilities

Large Language Models leveraging search engines to serve relevant information to the user.



- The essential role of a search engine in this novel architecture is not only to crawl and index the Internet, but also to find and present the most relevant information to the LLM, based on the specific user needs and preferences interpreted by the model.
- Search engines no longer need to directly show users the ranked results of a query.
- Search engines need to send the most pertinent information to the Language Model (perhaps supported by URLs), which they can then synthesise, summarise, and present in a more consumable way for the end user.
- The crawling and ranking functions of the search engine need to work seamlessly with the synthesis capabilities of the LLM

Advantages of LLMs over Search Engines as the front page of the Web

- A fundamental benefit of LLMs over search engines lies in their capacity to comprehend intricate queries and produce personalised responses.
- LLMs possess the capability to grasp and interpret natural language queries, resulting in more precise and pertinent replies.
- LLMs can also tailor their responses based on user preferences and prior interactions, enhancing the usefulness and relevance of the information provided.
- LLMs have capacity to learn and adapt to novel tasks and domains through their existing knowledge.
- LLMs possess the ability to assimilate new data and user interactions, enabling them to enhance their accuracy and relevance progressively as time unfolds.
- LLMs utility can potentially expand beyond web-based content
- Users can engage with LLMs in their preferred language, even if the underlying information the model has been trained on was not initially available in that specific language

Challenges and Limitations of Large Language Models

- Accuracy and validity of the information they present is a concern
- LLMs generate responses based on their training data, which may occasionally lack accuracy, correctness and may be affected by bias - users being exposed to erroneous or incomplete information.
- Privacy concerns represent another significant challenge tied to LLMs - The data employed for training LLMs could potentially include personal or sensitive information, thereby raising apprehensions regarding data privacy and security.
  - Data generated during interactions between LLMs and users may also give rise to privacy concerns over sensitive data.

- AI LLMs may continue to be used to enhance search engines in the future to:
  - Generate more informative and comprehensive search results.
  - Personalize search results based on a user's individual needs and interests.
  - Identify and filter out spam and misinformation.
  - Provide users with a more natural and conversational search experience.
- The future of search will likely be a hybrid of traditional search engines and AI LLMs.

Search Engine v/s LLMs

- **Up-to-date information**: Search engines offer access to the latest information available on the web. LLMs might not have real-time data unless they're consistently updated or access the internet.
- **Different use cases**: LLMs are great for conversational AI, tutoring, content generation, coding, and more, while search engines are essential for research, news, and broad information discovery. With the incorporation of LLMs into the search results, this will continue to change over time.

Search Engines that are powered by AI LLMs