Information Retrieval
CS 6900

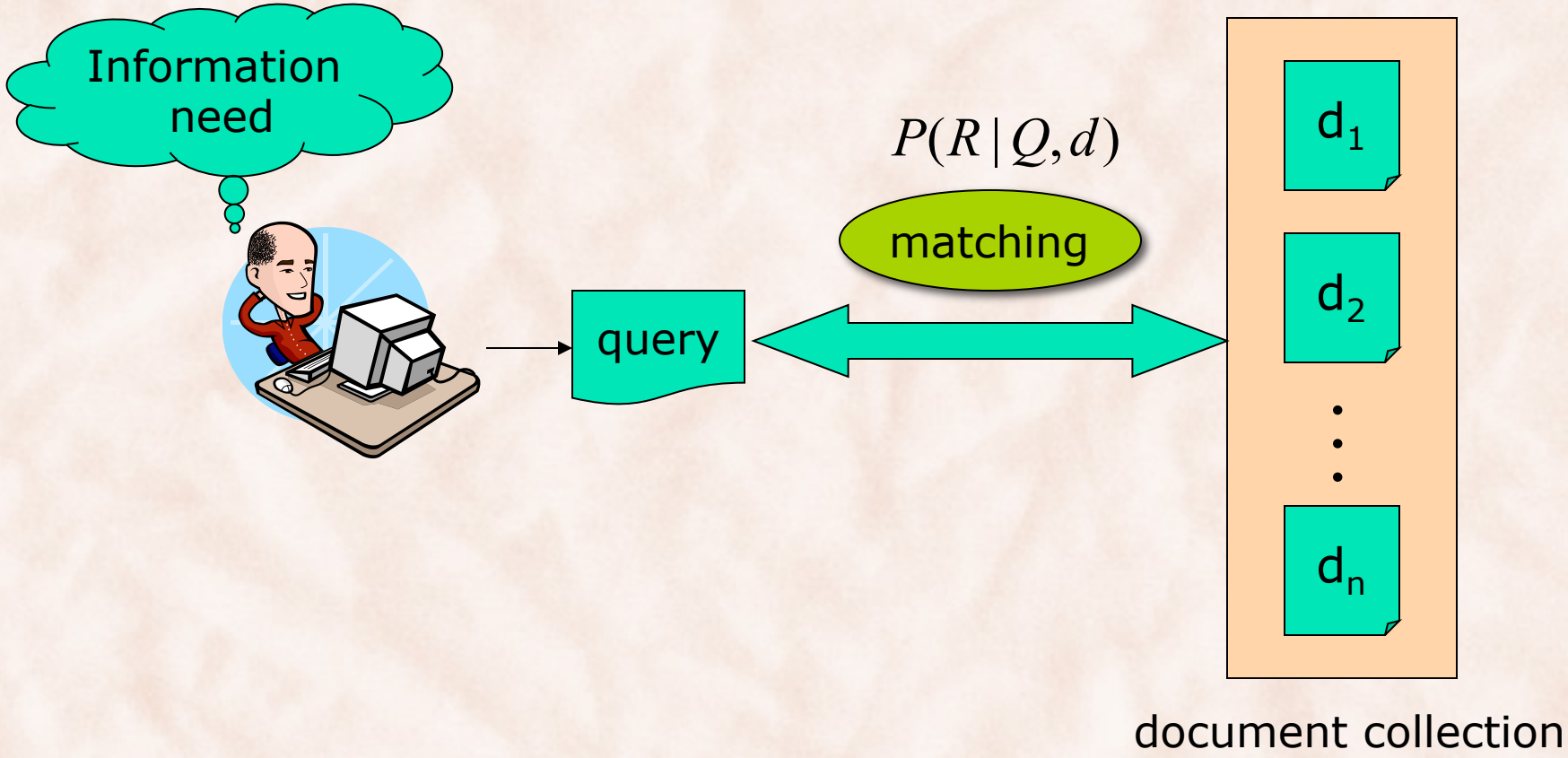
Lecture 11

Razvan C. Bunescu

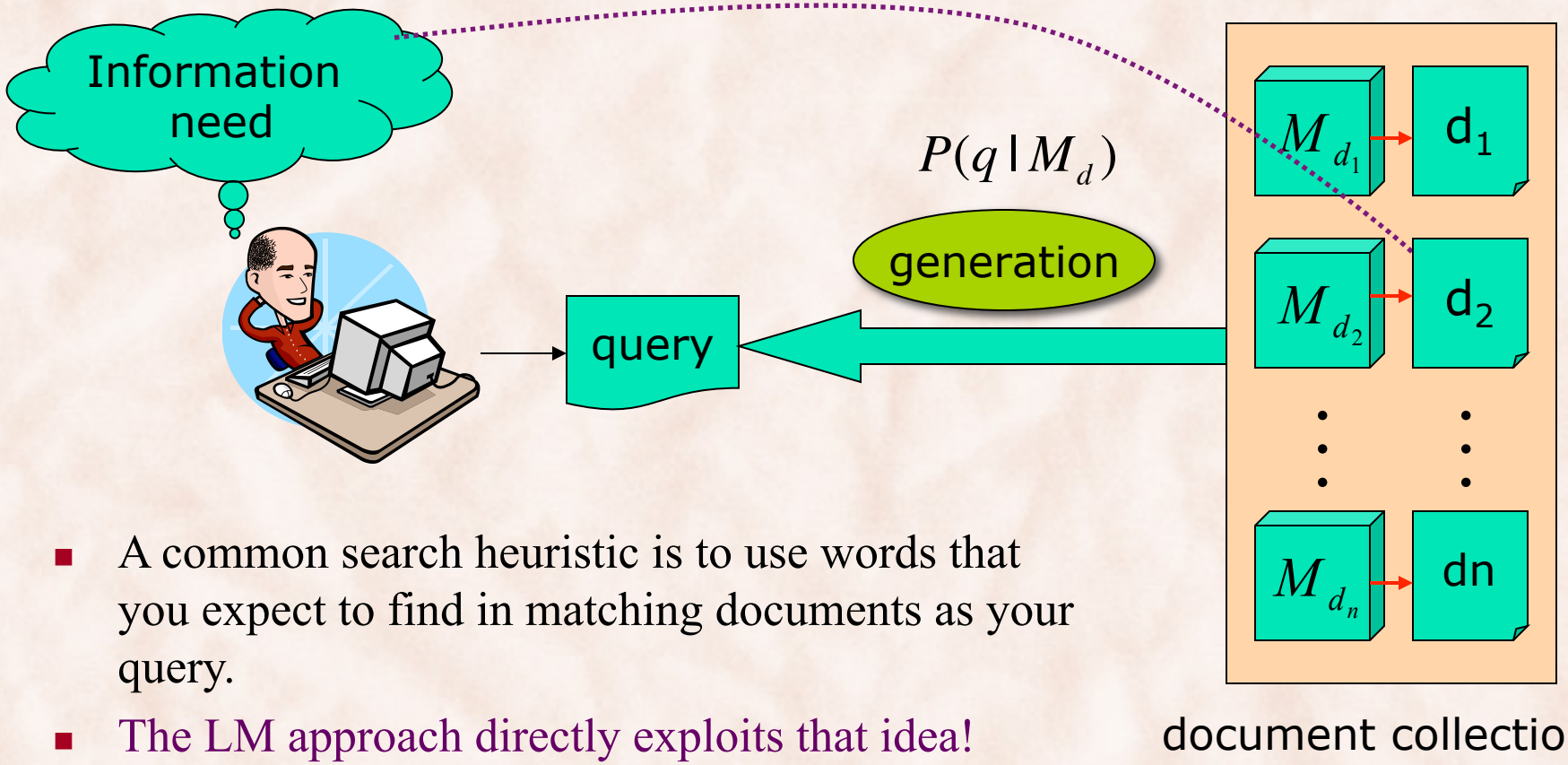
School of Electrical Engineering and Computer Science

bunescu@ohio.edu

Standard Probabilistic IR



Language Models for IR



- A common search heuristic is to use words that you expect to find in matching documents as your query.
- The LM approach directly exploits that idea!

Language Models for IR

- Assumption: The user has a prototype document in mind and **generates** a query based on words that appear in this document.
- Approach: View each document as a *language model* and compute the probability that it **generates** the query:
  - Use actual document to estimate the LM's parameters.
  - Use smoothing to avoid zeros and to include collection statistics.
  - Compute probability of generating the query.
  - Find documents most likely to have generated the query.
  - Present ranked documents to the user.

Language Models i.e. N-gram Models

- **Next word prediction**:
  - Sue swallowed the large green _____
    - tree, car, desk, pill, frog, alien, …
  - How? Predict next word w^* given history $w_1, \ldots, w_{n-1}$:

$$w^* = \arg\max_{w_n \in V} p(w_n \mid w_1, \ldots, w_{n-1})$$

- **Sentence probability estimation**:
  - p("Sue swallowed the large green pill") = ?
  - (related to) / (reformulation of) **next word prediction**.

$$p(w_1, \ldots, w_n) = p(w_1)p(w_2 \mid w_1)p(w_3 \mid w_1, w_2)\ldots p(w_n \mid w_1, \ldots, w_{n-1})$$

N-Gram Models

- Predict next word w^* given history $w_1, \ldots, w_{n-1}$:

$$w^* = \arg\max_{w_n \in V} p(w_n \mid w_1, \ldots, w_{n-1})$$

history

  - Unigram Model: $p(w_n)$
  - Bigram Model: $p(w_n \mid w_{n-1})$
  - Trigram Model: $p(w_n \mid w_{n-2}, w_{n-1})$

- Building N-Gram models:
  - How can we estimate $p(w_n|w_1, \ldots, w_{n-1})$?

N-gram Models: Statistical Estimation

- Reduce to estimating the probability distribution of *n*-grams:

$$p(w_n \mid w_1, \ldots, w_{n-1}) = \frac{p(w_1, \ldots, w_n)}{p(w_1, \ldots, w_{n-1})}$$

- Use a training text to estimate n-gram distributions:
  - Assume sequence of *N* words:
    - pad with *n*-1 dummy symbols to the beginning ⇒ *N* *n*-grams.
  - Assume the occurrence of each particular n-gram is a random variable with Bernoulli distribution (quite untrue, but usable!).
  - Use **Maximum Likelihood Estimate** (here, *relative frequency*):

$$p(w_1, \ldots, w_n) = \frac{C(w_1, \ldots, w_n)}{N}$$

freq. of n-gram in training text

N-gram Models: MLE

$$p(w_1,\ldots,w_n) = \frac{C(w_1,\ldots,w_n)}{N}$$

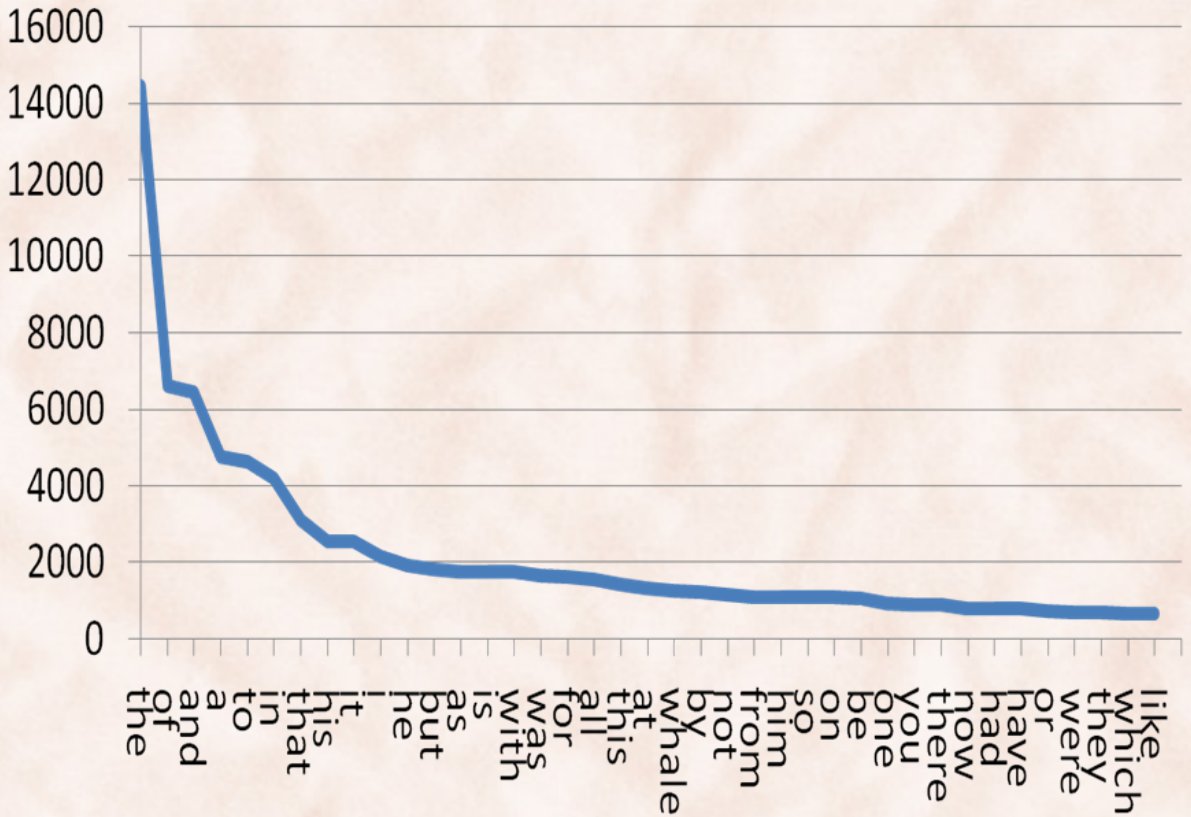
Why is this only an estimate?

$$\Rightarrow p(w_n \mid w_1,\ldots,w_{n-1}) = \frac{C(w_1,\ldots,w_n)}{C(w_1,\ldots,w_{n-1})}$$

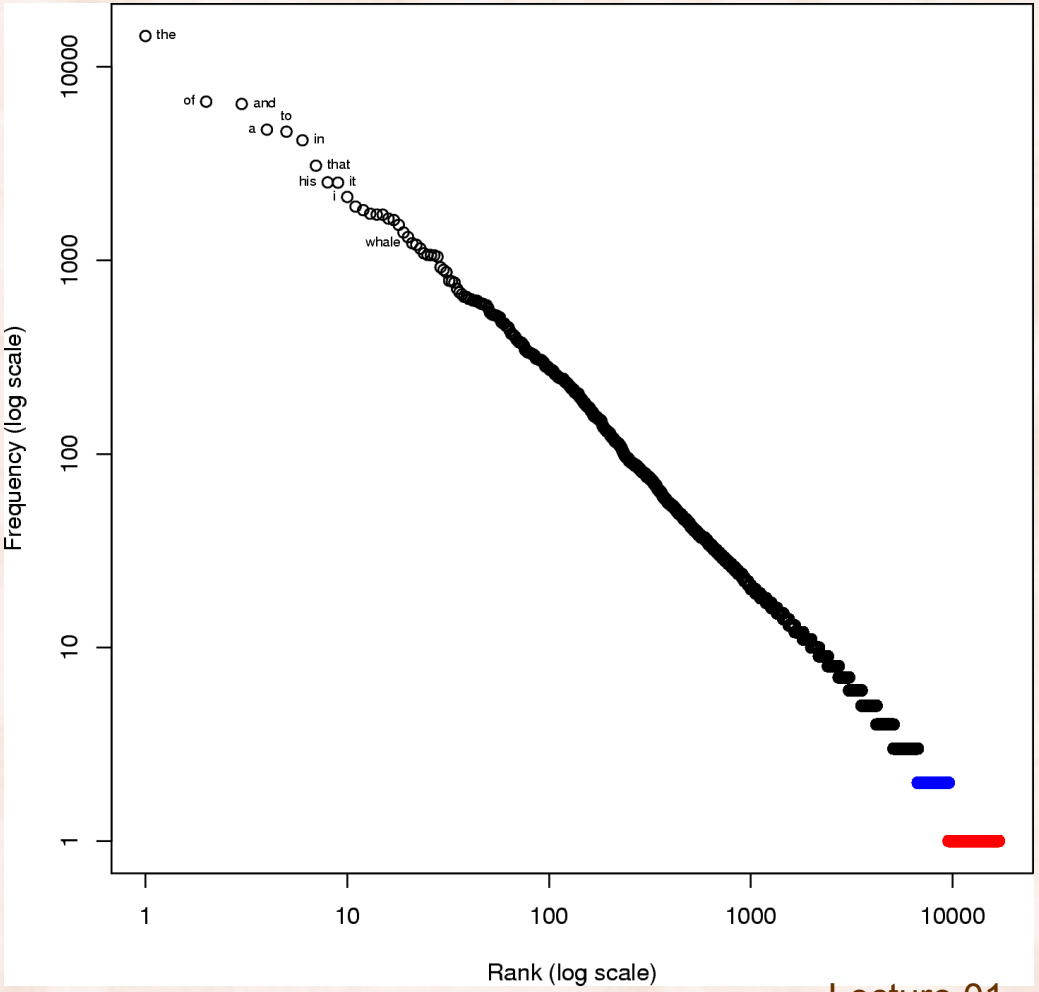
- MLE is in general unsuitable for statistical inference in NLP:
  - Sparsity leads to unreliable estimates for rare events!
  - Can alleviate sparsity effects (but not eliminate):
    - Increase *N* (text size) – how much for sufficient trigram counts?
    - Decrease *n* (history size) – how much for accurate probabilities?
      - go from unlimited to limited history/memory.

Zipf's Law

Frequency vs. rank for all words in Moby Dick



Zipf's Law (Log Scale)



Moby Dick:

- 44% *hapax legomena*
- 17% *dis legomena*

Markov Chains

- Assume next word depends only on *k* previous words:
  - k[th] order Markov assumption

    $p(w_n \mid w_1, \ldots, w_{n-1}) \approx p(w_n \mid w_{n-k}, \ldots, w_{n-1})$
  - 1[st] order Markov model:

    $p(w_n \mid w_1, \ldots, w_{n-1}) \approx p(w_n \mid w_{n-1}) \Rightarrow p(w_1, \ldots, w_n) \approx ?$

- Assume vocabulary |V| = 30K:
  - 0-gram LM (uniform) $\Rightarrow$ 1 params: $p(w)=1/|V|$,
  - 1-gram LM (unigram) $\Rightarrow$ $3*10^4$ params: $p(w_n)$
  - 2-gram LM (bigram) $\Rightarrow$ $9*10^8$ params: $p(w_n \mid w_{n-1})$
  - 3-gram LM (trigram) $\Rightarrow$ $2.7*10^{13}$ params: $p(w_n \mid w_{n-2}, w_{n-1})$

Out Of Vocabulary (OOV) Words

It is not practical to calculate n-grams for all words.

Possible approaches:

1) Use only the most common words:
   - replace 1-count words (*hapax legomena*) by <UNK>.
     - half of the types, but only a fraction of the tokens (Zipf's law).

2) Use a predefined vocabulary.
   - replace OOV words by <UNK>.

3) Replace the first occurrence of every word type in the training data by <UNK>

Language Models for IR

- Build a **different language model for each document**:
  ⇒ Text to train parameters = 1 document.

- Most work in IR uses **unigram** language models:
  - 1 document is not enough for training higher order LMs.
  - IR does not depend on sentence structure as other applications of language models (e.g. speech recognition, machine translation):
  - Unigram LMs are often sufficient to judge the topic of the text.

A different language model for each document

Language model of document d_1:

w	a	the	that	frog	toad	dog	said	likes
$P(w\|M_1)$	0.1	0.2	0.04	0.01	0.01	0.005	0.03	0.02

Language model of document d_2:

w	a	the	that	frog	toad	dog	said	likes
$P(w\|M_2)$	0.1	0.2	0.04	0.0002	0.0001	0.01	0.03	0.04

$q =$ *frog said that toad likes that dog*

$P(q|M_1) = 0.01 \times 0.03 \times 0.04 \times 0.005 \times 0.02 \times 0.04 \times 0.005 = 0.00000000000048$

$P(q|M_2) = 0.0002 \times 0.03 \times 0.04 \times 0.0001 \times 0.04 \times 0.04 \times 0.01 = 0.000000000000000384$

$\Rightarrow P(q|M_1) > P(q|M_2)$

$\Rightarrow$ document d_1 is more relevant than document d_2 for query q.

Why use $P(q|d)$ for ranking documents?

- Each document is treated as (basis for) a language model.
- Given a query q, rank documents d based on $P(d|q)$:

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

  - $P(q)$ is the same for all documents, so ignore.
  - $P(d)$ is the prior, often treated as uniform distribution:
    - But could give a higher prior to high quality documents, using criteria such as *authority* (e.g. PageRank), *length*, *genre*.
  - $P(q|d)$ is the posterior probability of q given d.

$\Rightarrow$ for uniform prior, ranking documents according to $P(d|q)$ or $P(q|d)$ is equivalent.

The Query Likelihood Model

- Use unigram language model M_d (trained on d) to estimate:

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

  - $|q|$ is the length of q; t_k is the token occurring at position k in q.
  - Same conditional independence assumptions as in **Naive Bayes**.

- This is equivalent to:

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

  - $\text{tf}_{t,q}$ is the term frequency of t in q.
  - Equivalent with a **multinomial model** (up to a constant factor).

Estimating Parameters $P(t|M_d)$

- Start with maximum likelihood estimates (training text = document *d*):

$$P(t \mid M_d) = \frac{tf_{t,d}}{|d|}$$

- Problem: zero probabilities.
  - A single term with $P(t|M_d) = 0$ would make $P(q|M_d) = 0$.
    - results in conjunctive semantics.
    - a single term has "veto power".
- Solution: *smooth* the estimates.
  - make a non-occurring term possible.
  - but no more likely than expected by chance in the entire collection.

Jelinek-Mercer Smoothing

- Create a language model M_C for the entire collection:

$$P(t \mid M_c) = \frac{\text{cf}_t}{T} \text{ , where } T = \sum_{t \in V} \text{cf}_t$$

  - cf_t is the number of occurrences of term t in the collection.
  - T is the total number of tokens in the collection.

- Use $P(t|M_c)$ to smooth $P(t|d)$ away from zero:

$$P(t \mid d) = \lambda P(t \mid M_d) + (1 - \lambda) P(t \mid M_c)$$

  - Tuning λ is important for good performance:
    - high values for λ = conjunctive-like search => tends to retrieve documents containing all query words.
    - low values for λ = disjunctive => suitable for long queries.

Jelinek-Mercer Smoothing

- Query Likelihood Model:

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

- Assumption: The user has a prototype document in mind and **generates** a query based on words that appear in this document.
- Approach: View each document as a *language model* and compute the probability that it **generates** the query:
  - The probability that the document that the user had in mind is *d*.

Document LM

Collection LM

Example

- Document collection has 2 documents:
  - d_1: Xerox reports a profit but revenue is down
  - d_2: Lucent narrows quarter loss but revenue decreases further
- Query q = *revenue down*.
- Unigram language model:
  - MLEs, smoothed with $\lambda = ½$.
  - $P(q|d_1) = [(1/8 + 2/16)/2] \times [(1/8 + 1/16)/2]$
    $= 1/8 \times 3/32 = 3/256$
  - $P(q|d_2) = [(1/8 + 2/16)/2] \times [(0 + 1/16)/2]$
    $= 1/8 \times 1/32 = 1/256$
- Ranking: $d_1 > d_2$

Dirichlet Smoothing

- Use background distribution $P(t|M_c)$ as a prior distribution in a *Bayesian updating process*:

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

  - Before seeing any part of the document, we start with the background distribution as our estimate:
    - Rather than the uniform estimate used in Laplace smoothing.
  - As we read the document and count terms, we update the posterior distribution.
  - Parameter α determines trade-off between document effect and collection effect.

Jelinek-Mercer vs. Dirichlet

- Dirichlet performs better for keyword queries.
- Jelinek-Mercer performs better for verbose queries.

- Both models are sensitive to the smoothing parameter:
  $\Rightarrow$ it is very important to do proper parameter tuning.

Ponte and Croft's Experiments

	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	*

- TREC topics 202-250 on TREC disks 2 and 3:
  - Natural language queries consisting of one sentence.
- Compared LM with TF.IDF that includes length normalization.
  - Statistical significance marked with * (Wilcoxon).
  - Improvements especially at high recall levels.

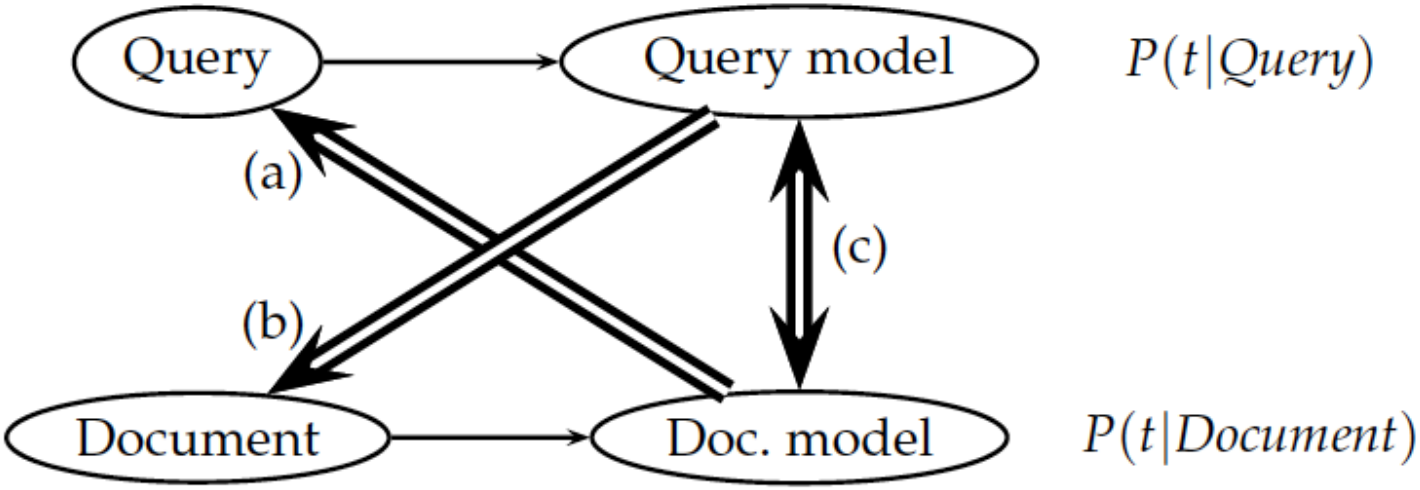
VSM vs. BM25 vs. LM

- BM25 and LM are both based on probability theory.
- Term frequency is used directly in all three:
  - Raw term frequencies in LM, more complex in VSM and BM25.
- Length normalization:
  - VSM: cosine or pivot normalization.
  - BM25: parameterized length normalization.
  - LM: probabilities are inherently normalized.
- Document (VSM, BM25) vs. collection frequency (LM):
  - LM mixes term and collection frequencies, with similar effect.
  - Terms rare in the general collection but common in some documents will have a greater influence on the ranking.e

LM: Pros and Cons

- LM approach does away with explicitly modeling relevance:
  - Relevance feedback and user preferences are difficult to exploit.
- Based on probability theory:
  - More principled than VSM, which is rather heuristic.
- Shown to outperform VSM and BM25:
  - Still, insufficient evidence that performance gains vs. well tuned VSM justify chaning an existing implementation.
- Assumes documents and queries are of the same type.
  - May be unrealistic, but same is true for VSM.
- Assumes terms are conditionally independent:
  - But same is true for VSM.

Language Modeling for IR



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

Language Modeling for IR

1) **Query Likelihood Model**:
   - Rank using $P(q|M_d)$.
   - Estimate M_q from document, smooth with collection model.
   - Problems dealing with relevance feedback, phrase queries, ...

2) **Document Likelihood Model:**
   - Rank using $P(d|M_q)$,
   - Hard to estimate M_q from query => need strong smoothing.
   - Easy to incorporate relevance feedback:
     - expand query with terms from relevant docs, and update M_q.

Language Modeling for IR

3) **Model Comparison**:

- Create language models for both query and document.
- Ask how different they are from each other, and use it to rank.
  - Use KL divergence:

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

- Lafferty and Zhai (2001) show that it outperforms both query and document likelihood models.

Extensions to the LM Approach

- Berger and Lafferty (1999) introduce **translation models**:
  - Bridge lexical gap between query and document:
    - synonymy, different word choices.
  - Build a translation model as $T(t|v)$ between vocabulary terms:
    - use thesaurus, or bilingual dictionary, or SMT translation dict.
    - can also be built using document collection, using pieces of text that naturally paraphrase or summarize other pieces of text:
      - documents and their titles, abstracts, or inlink anchor texts.
  - Build a translation query generation model from $T(t|v)$ and $P(v|M_d)$:

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

Implementations of LM Approaches to IR

- **The Lemur Toolkit**:
  - www.lemurproject.org
  - Supports major language modeling approaches such as Indri and KL-divergence:
    - see class lemur::langmod::UnigramLM and its subclasses.
  - Also TF.IDF and Okapi BM25.
- **Apache Lucene Core**:
  - http://lucene.apache.org
  - See class lucene.search.similarity.LMSimilarity and its subclasses:
    - LMJelinekMercerSimilarity, LMDirichletSimilarity.
  - Also TFIDFSimilarity and BM25Similarity.