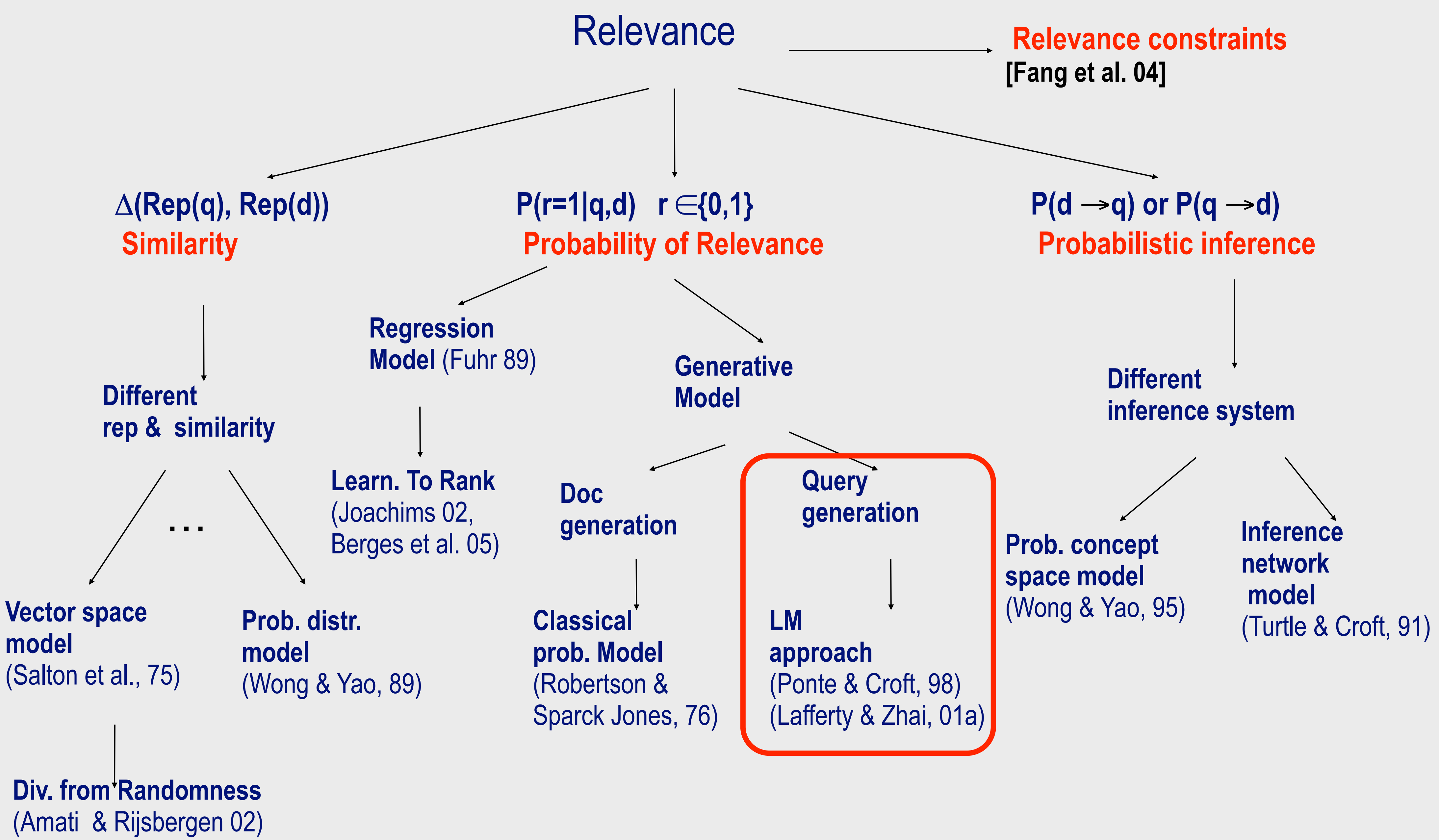
Web Search and Mining

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

Language Model for Information Retrieval (IIR 12)

The Notion of Relevance



Recall

- Basic idea
- Compute the odd of *O(R=1|Q,D)* using Bayes' rule

$$O(R=1|Q,D) = \frac{P(R=1|Q,D)}{P(R=0|Q,D)} = \frac{P(Q,D|R=1)}{P(Q,D|R=0)} \boxed{\frac{P(R=1)}{P(R=0)}}$$ ignored for ranking D

- Special cases
  - How to define *P(Q, D|R)*
  - Document "generation": *P(Q, D | R) = P(D | Q,R) P(Q | R)*
  - Query "generation": *P(Q, D | R) = P(Q | D,R) P(D | R)*

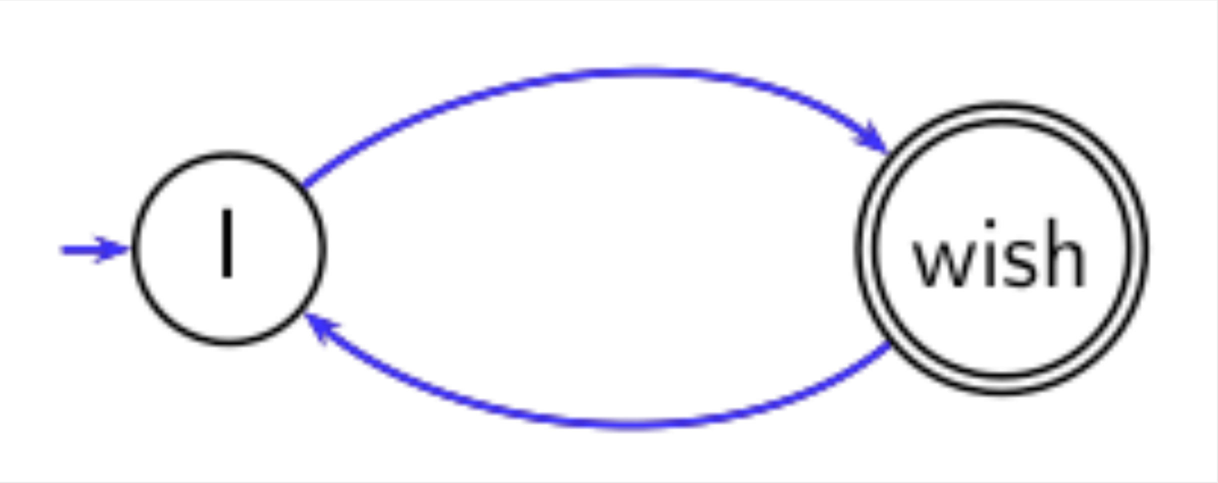
Using language models (LMs) for IR

- LM = language model
- We view the document as a generative model that generates the query.
- What we need to do:
  - Define the generative model of each document
    - Estimate parameters (different parameters for each document's model)
    - Smooth to avoid zeros
  - Apply to each document model to calculate the probability of generating the query
  - Present most likely document(s) to user

Language Model

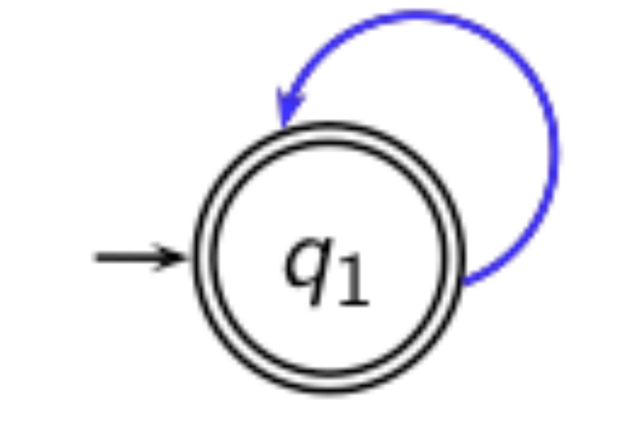
What is a language model?

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



- Generate: I wish I wish I wish I wish . . .
- Our basic model: each document was generated by a different automaton like this except that these automata are probabilistic.

A probabilistic language model



w	$P(w \mid q_1)$	w	$P(w \mid q_1)$
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.
- string = frog said that toad likes frog STOP
- P(string) = 0.01 · 0.03 · 0.04 · 0.01 · 0.02 · 0.01 · 0.2 = 0.0000000000048

A different language model for each document

language model of d_1			
w	$P(w\|.)$	w	$P(w\|.)$
STOP	.2	toad	.01
the	.2	said	.03
a	.1	likes	.02
frog	.01	that	.04
		...	...

language model of d_2			
w	$P(w\|.)$	w	$P(w\|.)$
STOP	.2	toad	.02
the	.15	said	.03
a	.08	likes	.02
frog	.01	that	.05
		...	...

- string = frog said that toad likes frog STOP
    - P(string|M_{d1}) = 0.01 · 0.03 · 0.04 · 0.01 · 0.02 · 0.01 · 0.2 = 0.0000000000048 = $4.8 \cdot 10^{-12}$
    - P(string|M_{d2}) = 0.01 · 0.03 · 0.05 · 0.02 · 0.02 · 0.01 · 0.2 = 0.0000000000120 = $12 \cdot 10^{-12}$
- P(string | M_{d1}) < P(string | M_{d2})
    - Thus, document d2 is “more relevant” to the string than d1 is.

Language Model for IR (1)

Where we are

- In the LM approach to IR, we attempt to model the query generation process.
  - Each document is treated as (the basis for) a language model.
- Then we rank documents by the probability that a query would be observed as a random sample from the a document model.
- That is, we rank according to $P(Q \mid D)$.
- Next: how do we compute $P(Q \mid D)$?

How to compute $P(q|d)$

- Multinomial model
- 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 of q; t_k : the token occurring at position k in q)

- This is equivalent to:

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

- $tf_{t,q}$: term frequency (# occurrences) of t in q

Multinomial Distribution

- Multinomial distribution: a generalization of the binomial distribution.
- Then let the random variables X_i: the number of times outcome number i was observed over the n trials.
- $X = (X_1, ..., X_k)$ follows a multinomial distribution with parameters n and p, where $p = (p_1, ..., p_k)$.
- Probability Mass Function

$$f(x_1,\ldots,x_k;n,p_1,\ldots,p_k) = \Pr(X_1 = x_1 \text{ and } \ldots \text{ and } X_k = x_k)$$

$$= \begin{cases} \dfrac{n!}{x_1!\cdots x_k!} p_1^{x1}\cdots p_k^{x_k}, & \text{when } \sum_{i=1}^k x_i = n \\ \\ 0 & \text{otherwise,} \end{cases}$$

Multinomial Distribution

- Example
  - In a recent three-way election for a large country, candidate A received 20% of the votes, candidate B received 30% of the votes, and candidate C received 50% of the votes. If six voters are selected randomly, what is the probability that there will be exactly one supporter for candidate A, two supporters for candidate B and three supporters for candidate C in the sample?

$$Pr(A=1,B=2,C=3)=\frac{6!}{1!2!3!}(0.2)^1(0.3)^2(0.5)^3=0.125$$

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; $tf_{t,d}$: # occurrences of t in d)

Maximum Likelihood Estimation

- Suppose one wishes to determine just how biased an unfair coin is. Call the probability of tossing a HEAD *p*.
- Suppose the coin is tossed 80 times: i.e., the sample might be something like *x1 = H, x2 = T, ..., x80 = T*, and the count of the number of HEADS "*H*" is observed.
- The probability of tossing TAILS is *1 – p*. Suppose the outcome is 49 HEADS and 31 TAILS, and suppose there are three coins: one which gives HEADS with probability p = 1/3, one which gives HEADS with probability p = 1/2 and another which gives HEADS with probability p = 2/3.
- Using maximum likelihood estimation the coin that has the largest likelihood can be found, given the data that were observed.

Maximum Likelihood Estimation

- By using the probability mass function of the binomial distribution with sample size equal to 80, number successes equal to 49 but different values of p (the "probability of success"), the likelihood function (defined below) takes one of three values:

$$\Pr(\mathrm{H}=49 \mid p=1/3) = \binom{80}{49}(1/3)^{49}(1-1/3)^{31} \approx 0.000,$$

$$\Pr(\mathrm{H}=49 \mid p=1/2) = \binom{80}{49}(1/2)^{49}(1-1/2)^{31} \approx 0.012,$$

$$\Pr(\mathrm{H}=49 \mid p=2/3) = \binom{80}{49}(2/3)^{49}(1-2/3)^{31} \approx 0.054.$$

- The likelihood is maximized when *p = 2/3*, and so this is the maximum likelihood estimate for *p*.

Maximum Likelihood Estimation

- Now suppose that there was only one coin but its p could have been any value $0 \le p \le 1$. The likelihood function to be maximized is

$$L(p) = f_D(\mathrm{H} = 49 \mid p) = \binom{80}{49} p^{49}(1-p)^{31},$$

- One way to maximize this function is by differentiating with respect to p and setting to zero:

$$\begin{aligned} 0 &= \frac{\partial}{\partial p}\left(\binom{80}{49} p^{49}(1-p)^{31}\right) \\ &\propto 49p^{48}(1-p)^{31} - 31p^{49}(1-p)^{30} \\ &= p^{48}(1-p)^{30}\left[49(1-p) - 31p\right] \\ &= p^{48}(1-p)^{30}\left[49 - 80p\right] \end{aligned}$$

- Thus the maximum likelihood estimator for p is 49/80.

Parameter estimation

- As always, we have a problem with zeros.
- A single t with $P(t|Md) = 0$ will make $P(q|M_d) = \prod P(t|M_d)$ zero.
- We would give a single term “veto power”.
  - For example, for query [Michael Jackson top hits] a document about “top songs” (but not using the word “hits”) would have $P(t|M_d) = 0$. – That’s bad.
- We need to smooth the estimates to avoid zeros.

Smoothing

- Key intuition: A non-occurring term
- 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.
- We will use $\hat{P}(t|M_c)$ to "smooth" $P(t|d)$ away from zero.

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

Mixture model

- $P(t|d) = \lambda P(t|Md) + (1 - \lambda)P(t|Mc)$
- 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: Compute ranking

- 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 revenue
- 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: $d_1 > d_2$

Language Model for IR (2)

Text Generation with Unigram LM

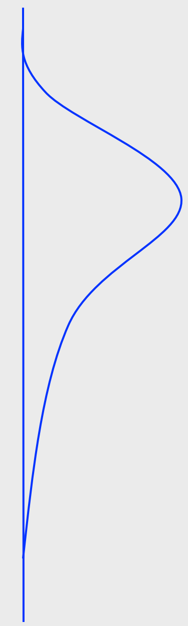
(Unigram) Language Model θ
p(w| θ)

Sampling

Document

Topic 1:
Text mining

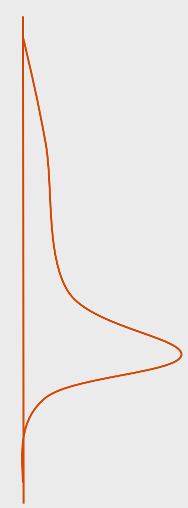
…
text 0.2
mining 0.1
association 0.01
clustering 0.02
…
food 0.00001
…



Text mining paper

Topic 2:
Health

…
food 0.25
nutrition 0.1
healthy 0.05
diet 0.02
…



Food nutrition paper

Estimation of Unigram LM

(Unigram) Language Model θ

p(w| θ)=?

Estimation

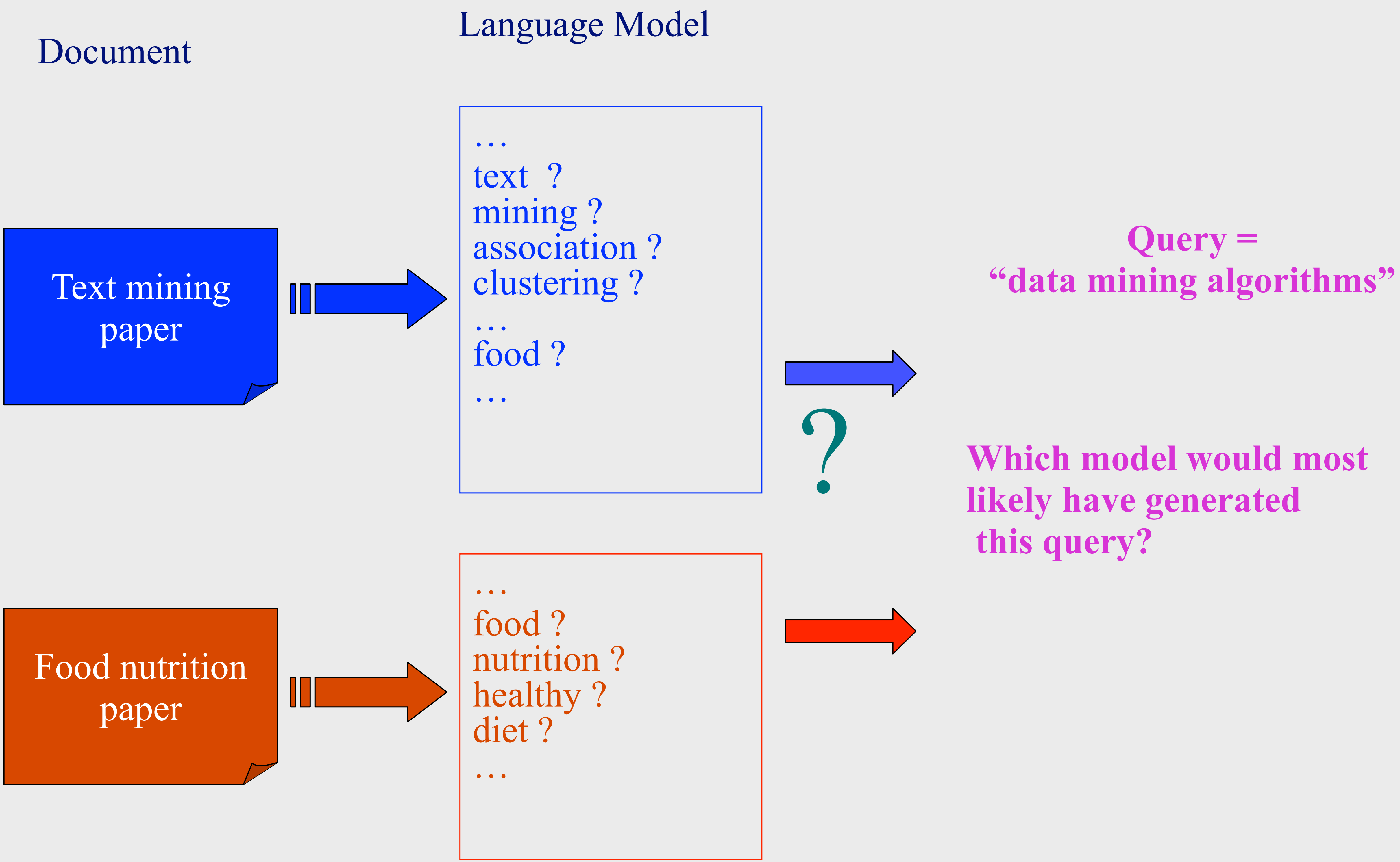
Document

	…
10/100	text ?
5/100	mining ?
3/100	association ?
3/100	database ?
	…
1/100	query ?
	…

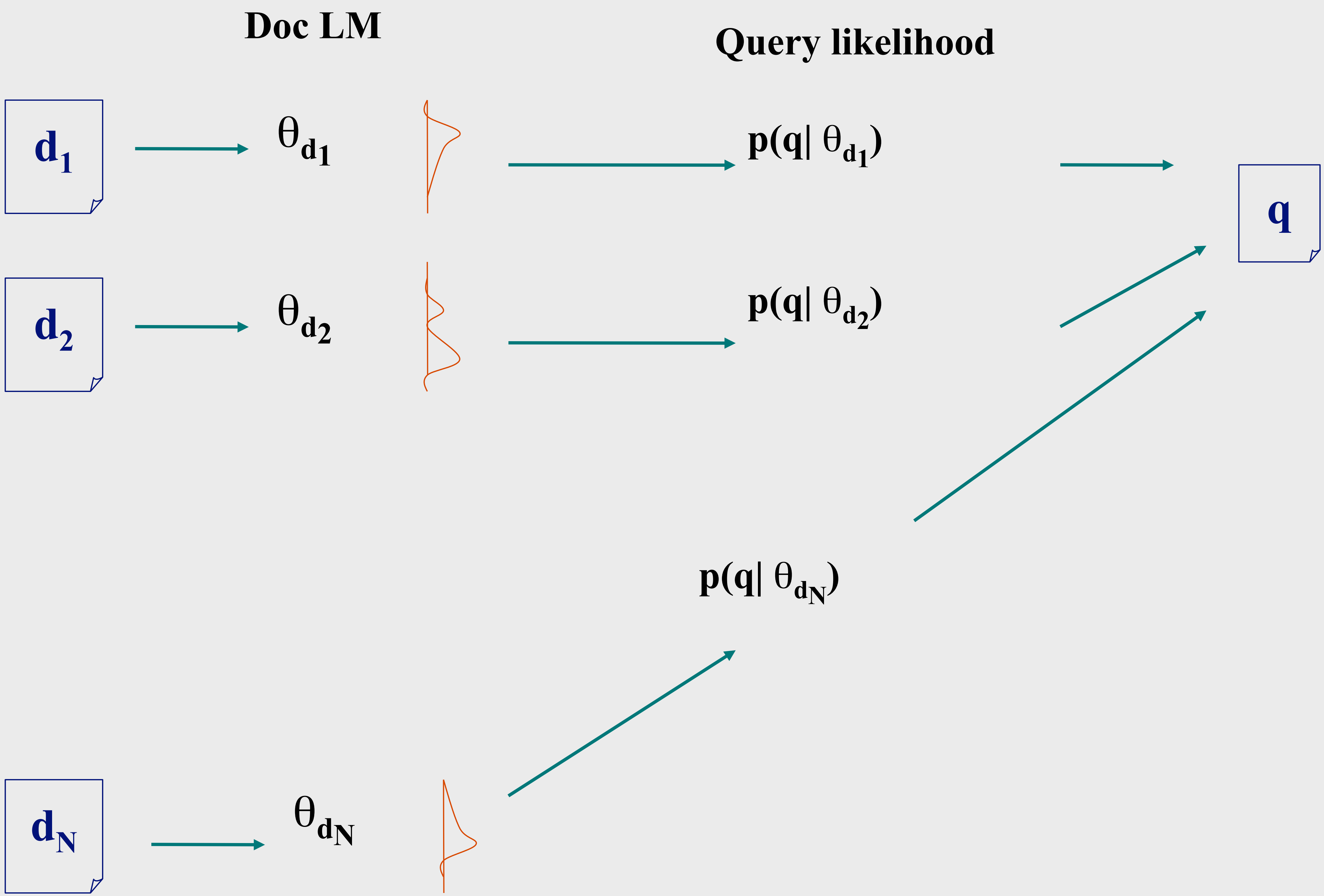
text 10
mining 5
association 3
database 3
algorithm 2
…
query 1
efficient 1

A "text mining paper"
(total #words=100)

Language Models for Retrieval
(Ponte & Croft 98)



Ranking Docs by Query Likelihood



Retrieval as Language Model Estimation

$$\log p(q|d) = \sum_i \log \boxed{p(w_i|d)}$$

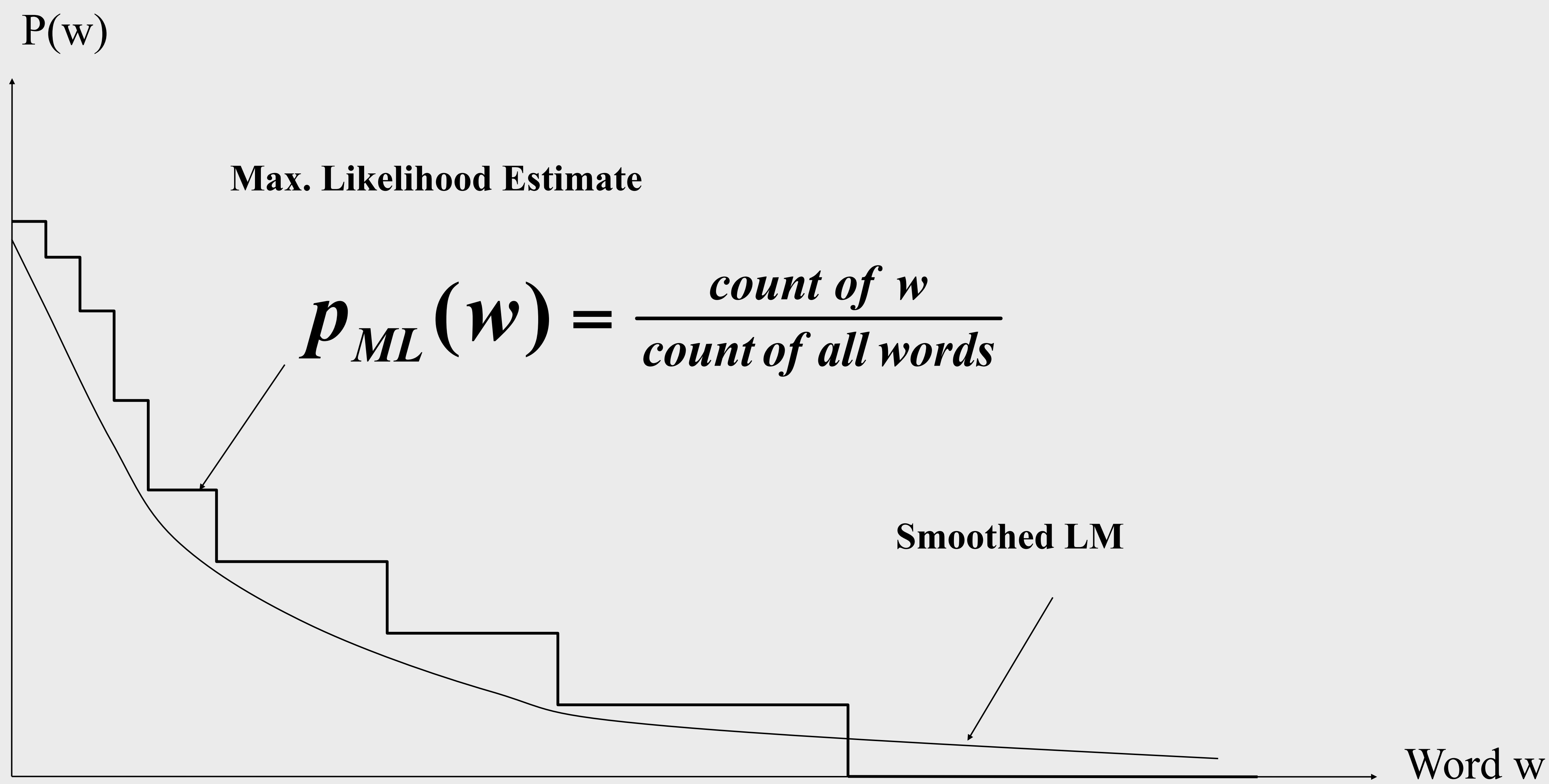
Document language model

- Document ranking based on query likelihood
- Retrieval problem $\approx$ Estimation of $p(w_i \mid d)$
- Smoothing is an important issue, and that distinguishes different approaches

How to Estimate $p(w|d)$?

- Simplest solution: Maximum Likelihood Estimator
  - $P(w|d)$ = relative frequency of word w in d
  - What if a word doesn't appear in the text? $P(w|d)=0$
- If we want to assign non-zero probabilities to such words, we'll have to discount the probabilities of observed words
- This is what "smoothing" is about …

Language Model Smoothing (Illustration)



A General Smoothing Scheme

- All smoothing methods try to
  - Discount the probability of words seen in a doc
  - Re-allocate the extra probability so that unseen words will have a non-zero probability
- Most use a reference model (collection language model) to discriminate unseen words

Discounted ML estimate

$$p(w \mid d) = \begin{cases} p_{seen}(w \mid d) & \textit{if } w \textit{ is seen in } d \\ \alpha_d \, p(w \mid C) & \textit{otherwise} \end{cases}$$

Collection language model

Derivation of the Query Likelihood Retrieval Formula

Discounted ML estimate

$$p(w|d) = \begin{cases} p_{Seen}(w|d) & if \;\; w \;\; is \;\; seen \; in \; d \\ \alpha_d \, p(w|C) & otherwise \end{cases}$$

Reference language model

$$\alpha_d = \frac{1 - \sum_{\text{w is seen}} p_{Seen}(w|d)}{\sum_{\text{w is unseen}} p(w|C)}$$

Derivation of the Query Likelihood Retrieval Formula

- Retrieval Formula using the general smoothing scheme

$$
\begin{aligned}
\log p(q|d) &= \sum_{w\in V, c(w,q)>0} c(w,q)\log p(w|d) \\
&= \sum_{w\in V, c(w,q)>0, c(w,d)>0} c(w,q)\log p_{seen}(w|d) + \boxed{\sum_{w\in V, c(w,q)>0, c(w,d)=0} c(w,q)\log \alpha_d p(w|C)} \\
&= \sum_{w\in V, c(w,q)>0, c(w,d)>0} c(w,q)\log p_{seen}(w|d) + \sum_{w\in V, c(w,q)>0} c(w,q)\log \alpha_d p(w|C) - \\
&\qquad \sum_{w\in V, c(w,q)>0, c(w,d)>0} c(w,q)\log \alpha_d p(w|C) \\
&= \sum_{w\in V, c(w,q)>0, c(w,d)>0} c(w,q)\log \frac{p_{seen}(w|d)}{\alpha_d p(w|C)} + |q|\log \alpha_d + \sum_{w\in V, c(w,q)>0} c(w,q)\log p(w|C)
\end{aligned}
$$

Key rewriting step

Similar rewriting are very common when using LMs for IR

Smoothing & TF-IDF Weighting

- Plug in the general smoothing scheme to the query likelihood retrieval formula, we obtain

TF weighting

Doc length normalization
(long doc is expected to have a smaller α_d)

$$\sum_{w \in V, c(w,q)>0, c(w,d)>0} c(w,q) \log \frac{p_{seen}(w|d)}{\alpha_d p(w|C)} + |q| \log \alpha_d + \sum_{w \in V, c(w,q)>0} c(w,q) \log p(w|C)$$

IDF weighting

Ignore for ranking

- Smoothing with *p(w|C)* ≈ TF-IDF + length norm.

Three Smoothing Methods
(Zhai & Lafferty 01)

- Simplified Jelinek-Mercer: Shrink uniformly toward *p(w|C)*

$$p(w|d) = (1-\lambda)p_{ml}(w|d) + \lambda p(w|C)$$

- Dirichlet prior (Bayesian): Assume pseudo counts *μ p(w|C)*

$$p(w|d) = \frac{c(w,d) + \mu p(w|C)}{|d| + \mu} = \frac{|d|}{|d|+\mu} p_{ml}(w|d) + \frac{\mu}{|d|+\mu} p(w|C)$$

- Absolute discounting: Subtract a constant *δ*

$$p(w|d) = \frac{\max(c(w,d) - \delta, 0) + \delta |d|_u p(w|C)}{|d|}$$

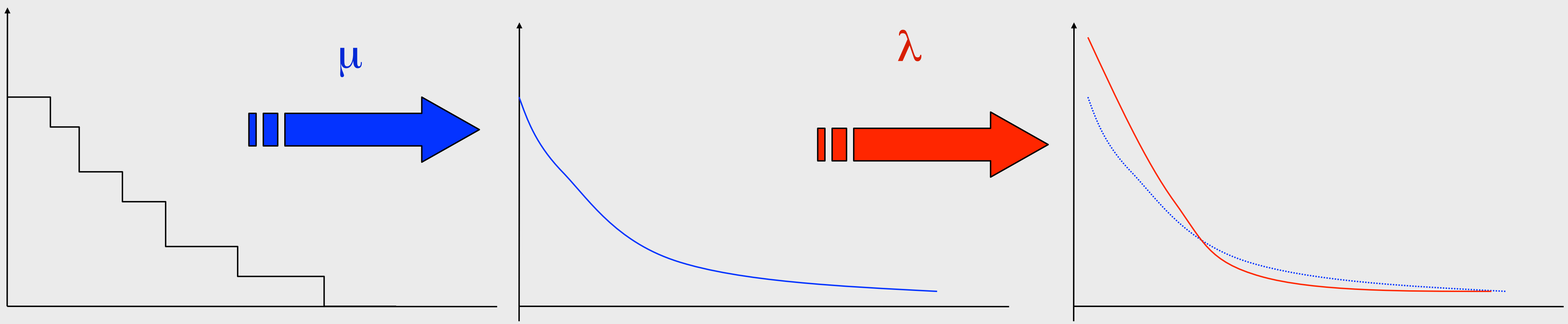
Two-stage Smoothing

Stage-1

-Explain unseen words
-Dirichlet prior(Bayesian)

Stage-2

-Explain noise in query
-2-component mixture



$$P(w|d) = (1-\lambda)\frac{c(w,d) + \mu p(w|C)}{|d| + \mu} + \lambda p(w|U)$$

User background model

λ and μ can be automatically set through statistical estimation

Discussions

Vector space (tf-idf) vs. LM

Rec.	precision			significant?
	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	*
0.4	0.2093	0.2572	+22.9	*
0.6	0.1024	0.1405	+37.1	*
0.8	0.0160	0.0432	+169.6	*
1.0	0.0028	0.0050	+76.9	
11-point average	0.1868	0.2233	+19.6	*

- 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.

LMs vs. Vector Space Model (1)

- LMs have some things in common with vector space models.
  - Term Frequency (TF) is directed in the models.
    - 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 Inverse Document Frequency (IDF).
    - Terms rare in the general collection, but common in some documents will have a greater influence on the ranking.

LMs vs. Vector Space Model (2)

- 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.

Language models for IR: Assumptions

- 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. Not true!
    - 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

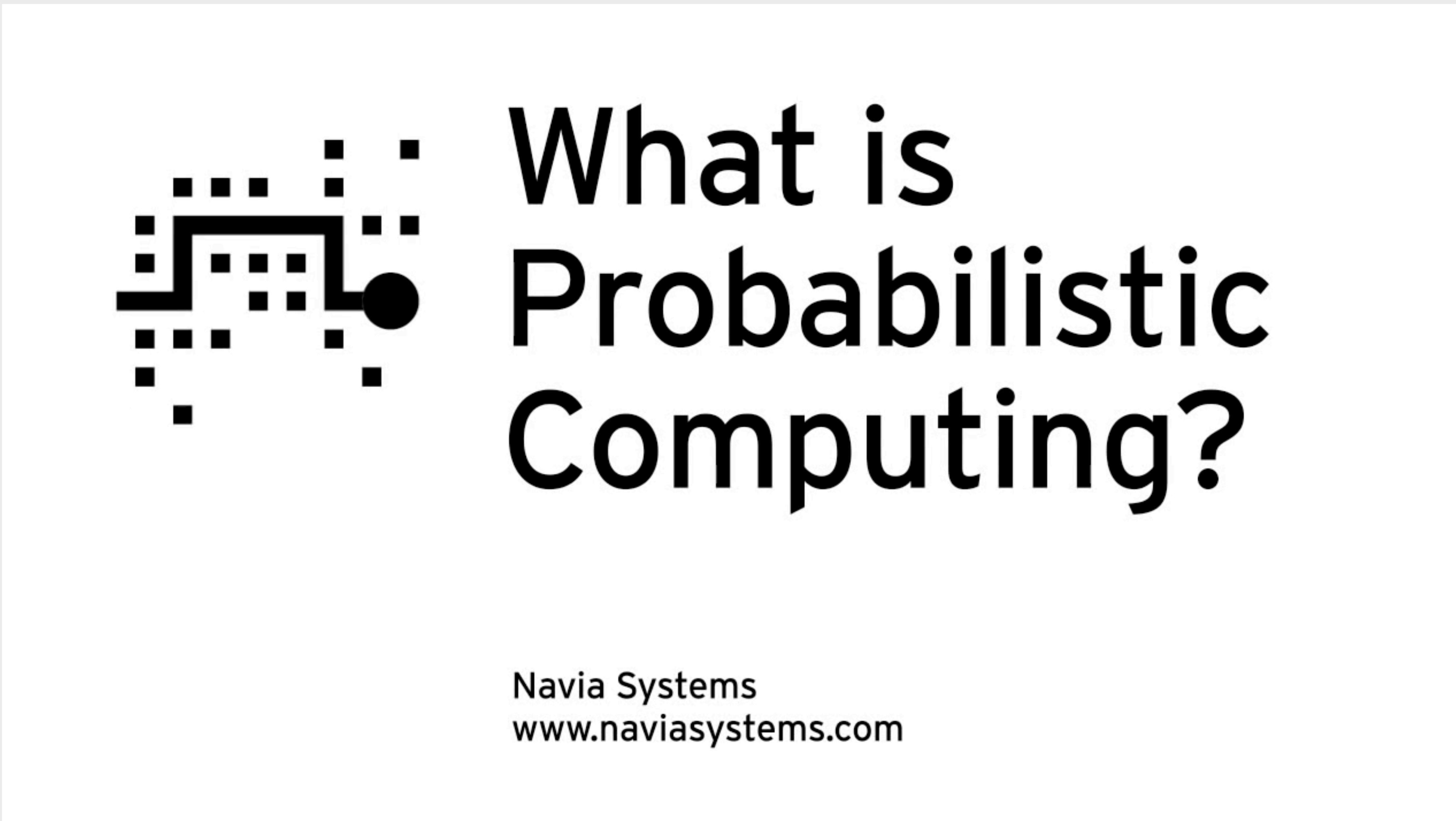
LMs vs. Naive Bayes

- Different smoothing methods
  - Mixture Model vs. Add-One
- We classify the query in LMs; we classify documents in text classification.
  - Each document is a class in LMs vs. classes are human-defined in text classification
  - The formal model is the same: multinomial model.

Resources

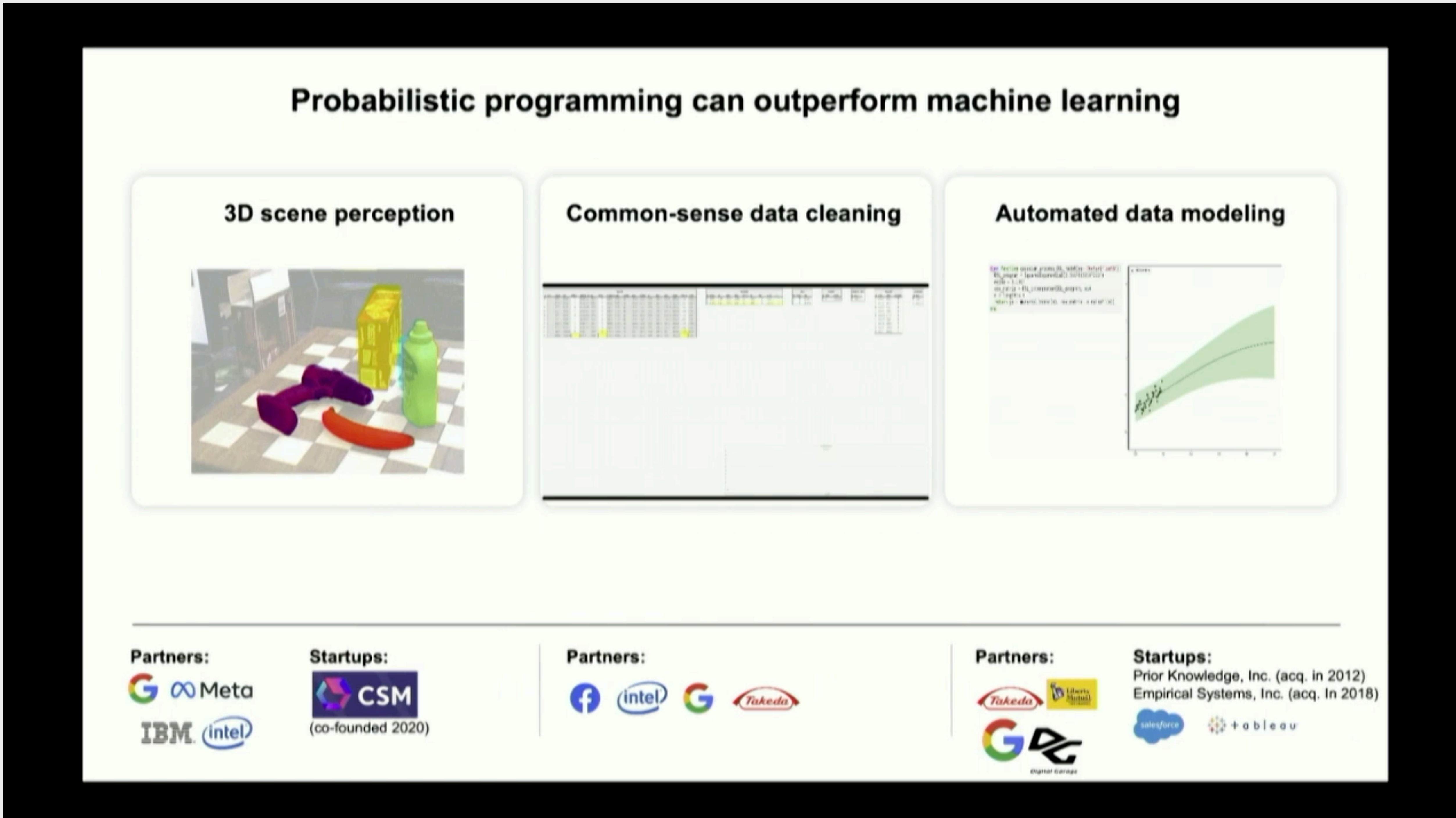
- Chapter 12 of IIR
- Resources
    - Ponte and Croft’s 1998 SIGIR paper (one of the first on LMs in IR)
    - Zhai and Lafferty’s 2001 SIGIR paper (the most important related paper in IR)
    - Lemur Toolkit (good support for LMs in IR)

Probabilistic Computing



http://www.youtube.com/watch?v=huIP__zhDTM

Probabilistic Programming



https://www.youtube.com/watch?v=8j2S7BRRWus