11-442 / 11-642:
Search Engines

Best-Match Retrieval: Statistical Language Models

Jamie Callan
Carnegie Mellon University
callan@cs.cmu.edu

Introduction

The last lecture introduced two best match retrieval models

- The vector space model (VSM), Okapi BM25

This lecture introduces statistical language models

- A newer form of probabilistic retrieval model
- Inspired by work in speech recognition and machine translation

Very different underlying theory

- But, implemented using many of the same values used by the vector space and Okapi BM25 retrieval models
  - Different explanations, but similar effects

Outline

Statistical language models

- Introduction to language models
- Query likelihood
- Kullback-Leibler (KL) Divergence
- Indri

3

Statistical Language Models

A statistical language model uses a probability distribution to determine the probability of terms or term sequences

Language model

Common types of models:

- Unigrams: $p(t_i \mid \theta)$ — p (“search” | θ)
- Bigrams: $p(t_i \mid t_{i-1}, \theta)$ — p (“engines” | “search”, θ)
- Trigrams: $p(t_i \mid t_{i-2}, t_{i-1}, \theta)$ — p (“class” | “search”, “engines”, θ)

Bigrams, trigrams, etc., haven’t helped much for IR (so far)

- But they are a key idea in speech recognition

4

Statistical Language Models

Word histogram

Term	tf_d
camera	17
image	13
picture	11
up	8
movie	8
like	7
mode	7
software	7
red	6
digital	5
eye	5
shutter	5
sony	5

Unigram language model

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

Term	$P(t\|\theta_d)$
camera	0.09551
image	0.07303
picture	0.06180
up	0.04494
movie	0.04494
like	0.03933
mode	0.03933
software	0.03933
red	0.03371
digital	0.02809
eye	0.02809
shutter	0.02809
sony	0.02809

5

A Language Model can be Created From Any Language Sample

Examples

- A document collection
- A document
- Also sentence, paragraph, chapter, …
- A query

This is similar to the vector space

- A vector can be created for any sample of text

Term	$P(t\|\theta_d)$
camera	0.09551
image	0.07303
picture	0.06180
up	0.04494
movie	0.04494
like	0.03933
mode	0.03933
software	0.03933
red	0.03371
digital	0.02809
eye	0.02809
shutter	0.02809
sony	0.02809

6

Terminology

d: A document

θ_d: The language model for document d

q: A query

θ_q: The language model for query q

d and θ_d are not the same thing

- The document and the model of the document are different
- However, to simplify notation, we will often treat them as the same
  - i.e., $p(d \mid q)$ instead of $p(d \mid \theta_q)$ or $p(\theta_d \mid \theta_q)$

Retrieval Model Based Upon Statistical Language Models

A document d defines a probability distribution θ_d over index terms

- E.g., the probability of generating/observing an index term

t	p (t)
Apple	0.0204
Apple	0.0001
Campaign	0.0034
Obama	0.0102
:	:

A query q also defines a probability distribution θ_q

- A sparse distribution (more on this later)

How is a language model used to rank document d?

- Rank d by $p(d \mid \theta_q)$ — "query likelihood"
- Rank d by the similarity of θ_d and θ_q — "KL divergence"

Outline

Statistical language models

- Introduction to language models
- Query likelihood
- Kullback-Leibler (KL) Divergence
- Indri

Rank by P(d|q): The Query-Likelihood Approach

Task: Rank documents by p(d |q)

- Given a query q, what is the probability of document d?
- **Problems**
  - – q is a very sparse language model (few terms)
  - – q contains very little frequency information

Term	$P(t\|\theta_Q)$
digital	0.5
camera	0.5

- **Solution**

$$p(d \mid q) = \frac{p(q \mid d)\, p(d)}{p(q)} \quad \textbf{Bayes rule}$$

$$\propto p(q \mid d)\, p(d) \quad \textbf{Drop document-independent term}$$

Key issues

- How are p(q|d) and p(d) estimated?

Query-Likelihood Approach: Estimating *p(d)*

It is simple and convenient to assume that p(d) is uniform

- Later we will consider non-uniform p(d)

So…

$$p(d \mid q) \propto p(q \mid d)\,p(d)$$
$$\propto p(q \mid d)$$

Drop the constant term (for now)

What is p(q|d)? Simple Unigram Approach

Assume that a query is composed of <u>independent terms</u>

- Unjustified, but convenient
- Doesn't hurt effectiveness of other models (e.g., vector space)

Then …

$$p(q \mid d) = \prod_{q_i \in q} p(q_i \mid d)$$

This should look a little familiar (although the notation differs)

- The score of (q, d) is based on the scores of (q_i, d)
- Similar to what we saw with BM25 and the vector space
- How is p(q_i,|d) calculated?

Estimating $p(q_i|d)$

Maximum likelihood estimation (MLE) is a simple approach

$$p_{MLE}(q_i \mid d) = \frac{tf_{q_i,d}}{length(d)}$$

Is this a good estimate?

- Estimates are based on small samples (a single document)
  - So, perhaps not very accurate
- $p_{MLE}(q_i|d) = 0$ if q_i isn't in document d

$$p(q \mid d) = \prod_{q_i \in q} p_{MLE}(q_i \mid d) = 0$$

Boolean AND

  - q_i could be a good description of document d, even if it does not occur in document d

Estimating $p(q_i|d)$: Smoothing

Smoothing is used to solve two problems in the language modeling framework

- Imprecise probability estimates from MLE
- Probability estimates for unobserved terms
  - E.g., query terms that don't occur in the document

But … there are many smoothing methods … which to use?

Estimating $p(q_i|d)$: Jelinek-Mercer ("Mixture Model") Smoothing

Linear interpolation with a reference language model

$$p(q_i \mid d) = (1-\lambda) p_{MLE}(q_i \mid d) + \lambda p_{MLE}(q_i \mid C)$$

Smoothing decreases as $\lambda \rightarrow 0$

What value of λ is best?

- Small λ (little smoothing) is best for short queries
- Larger λ (more smoothing) is better for long queries
- We will see later in the lecture why this is true

Estimating $p(q_i|d)$: Bayesian Smoothing With Dirichlet Priors

Method #2: Bayesian smoothing using Dirichlet priors

$$p_{MLE}(q_i \mid d) = \frac{tf_{q_i,d} + \mu \, p_{MLE}(q_i \mid C)}{length(d) + \mu}$$

$$p_{MLE}(q_i \mid C) = \frac{ctf_{q_i}}{|length_{tokens}(C)|}$$

What value of μ is best?

- μ in [1,000-10,000] seems best
  - Some people think μ is approximately related to average document length … others disagree

Estimating $p(q_i|d)$: Smoothing

We know that tf.idf weights are effective

…how do they relate to statistical language models

tf is easy to see

$$p_{MLE}(q_i \mid d) = \frac{tf_{q_i,d}}{length(d)}$$

idf is a little harder to see

- Jelenick-Mercer smoothing provides an idf-like effect
- Next slide…

17

No Smoothing Example:

$$p(q \mid d) = \prod_{q_i \in q} p_{MLE}(q_i \mid d)$$

Two query terms: p (apple | C) = 0.01, p (ipod | C) = 0.001

"frequent" **"rare"**

Document d_1

doclen =50, tf_{apple} = 2, tf_{ipod} = 3

(2/50) ×
(3/50) =

0.04 × 0.06 =

$p(q|d_1)$ = 0.0024

Document d_2

doclen =50, tf_{apple} = 3, tf_{ipod} = 2

(3/50) ×
(2/50) =

0.06 × 0.04 =

$p(q|d_2)$ = 0.0024

The model <u>does not</u> distinguish between frequent and rare terms

18

Jelinek-Mercer Smoothing Example:

$$p(q \mid d) = \prod_{q_i \in q} (1-\lambda) p_{MLE}(q_i \mid d) + \lambda\, p_{MLE}(q_i \mid C)$$

Two query terms: p (apple | C) = 0.01, p (ipod | C) = 0.001
λ = 0.4 **"frequent"** **"rare"**

Document d_1

doclen =50, tf_{apple} = 2, tf_{ipod} = 3

(0.6 × (2/50) + 0.4 × 0.01) ×
(0.6 × (3/50) + 0.4 × 0.001) =

(0.6 × 0.04 + 0.004) ×
(0.6 × 0.06 + 0.0004) =

(0.024+0.004)×(0.036+0.0004)=

$p(q|d_1)$ = 0.001019

Document d_2

doclen =50, tf_{apple} = 3, tf_{ipod} = 2

(0.6 × (3/50) + 0.4 × 0.01) ×
(0.6 × (2/50) + 0.4 × 0.001) =

(0.6 × 0.06 + 0.004) ×
(0.6 × 0.04 + 0.0004) =

(0.036+0.004)×(0.024+0.0004)=

$p(q|d_2)$ = 0.000976

The model does distinguish between frequent and rare terms

19

Why Does Smoothing Work?

$$p(q \mid d) = \prod_{q_i \in q} (1-\lambda) p_{MLE}(q_i \mid d) + \lambda\, p_{MLE}(q_i \mid C)$$

doclen =50, tf_{apple} = 2, tf_{ipod} = 2

p (apple | d) = 0.6 × (2/50) + 0.4 × 0.010 = 0.0280
p (ipod | d) = 0.6 × (2/50) + 0.4 × 0.001 = 0.0244

What is the effect of matching one additional instance of a term?

p_δ (apple | d) = 0.6 × (1/50) = 0.012
p_δ (ipod | d) = 0.6 × (1/50) = 0.012

– The unsmoothed effect of each match is the same

The incremental value of matching a term is multiplied by the p (q | d) of other query terms

– tf_{apple} = 3, tf_{ipod} = 2: p (q | d) = (0.028 + 0.012) × 0.0244 = 0.000976
– tf_{apple} = 2, tf_{ipod} = 3: p (q | d) = 0.028 × (0.0244 + 0.012) = 0.001019

20

Estimating $p(q_i|d)$: The idf-like Effect of Jelenick-Mercer Smoothing

$$
\begin{aligned}
p(q \mid d) &= \prod_{q_i \in q} p(q_i \mid d) \\
&= \prod_{q_i \in q} \left((1-\lambda) p_{MLE}(q_i \mid d) + \lambda p_{MLE}(q_i \mid C)\right) && \textbf{J-M smoothing} \\
&= \prod_{q_i \in q} \left((1-\lambda) p_{MLE}(q_i \mid d) + \lambda p_{MLE}(q_i \mid C)\right) \frac{\lambda p_{MLE}(q_i \mid C)}{\lambda p_{MLE}(q_i \mid C)} && \textbf{Multiply by 1} \\
&= \prod_{q_i \in q} \left(\left(\frac{(1-\lambda) p_{MLE}(q_i \mid d)}{\lambda p_{MLE}(q_i \mid C)} + 1 \right) \lambda p_{MLE}(q_i \mid C) \right) && \textbf{Recombine} \\
&= \prod_{q_i \in q} \left(\frac{(1-\lambda) p_{MLE}(q_i \mid d)}{\lambda p_{MLE}(q_i \mid C)} + 1 \right) \prod_{q_i \in q} \lambda p_{MLE}(q_i \mid C) && \textbf{Recombine} \\
&\propto \prod_{q_i \in q} \left(\frac{(1-\lambda) p_{MLE}(q_i \mid d)}{\lambda p_{MLE}(q_i \mid C)} + 1 \right) && \textbf{Drop constant}
\end{aligned}
$$

"tf effect"

"idf effect"

21

Estimating $p(q_i|d)$: Jelinek-Mercer ("Mixture Model") Smoothing

HW2 requires you to think about the effects of parameters

$$p(q_i \mid d) = (1-\lambda) p_{MLE}(q_i \mid d) + \lambda p_{MLE}(q_i \mid C)$$

What happens when λ approaches 0?

- There is no smoothing (no "idf like" effect)

$$p(q_i \mid d) = (1-\lambda) p_{MLE}(q_i \mid d)$$

22

Estimating $p(q_i|d)$: Jelinek-Mercer ("Mixture Model") Smoothing

HW2 requires you to think about the effects of parameters

$$p(q_i \mid d) = (1-\lambda) p_{MLE}(q_i \mid d) + \lambda p_{MLE}(q_i \mid C)$$

How important is smoothing to <u>queries</u> of different lengths?

- Short queries
  - Few query terms, so (usually) <u>every</u> query term must match
  - "idf weighting" is less important, so small λ is best
- Long queries
  - Many query terms, so (usually) <u>most</u> query terms must match
  - "idf weighting" is more important
  - Give priority to the less common terms, so larger λ is best

Estimating $p(q_i|d)$: Bayesian Smoothing With Dirichlet Priors

HW2 requires you to think about the effects of parameters

$$p(q_i \mid d) = \frac{tf_{q_i,d} + \mu \, p_{MLE}(q_i \mid C)}{length(d) + \mu}$$

What happens when μ approaches 0?

- There is no smoothing (maximum likelihood only)

$$p(q_i \mid d) = \frac{tf_{q_i,d}}{length(d)}$$

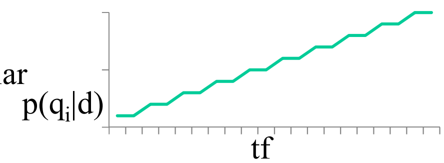
Estimating $p(q_i|d)$: Bayesian Smoothing With Dirichlet Priors

HW2 requires you to think about the effects of parameters

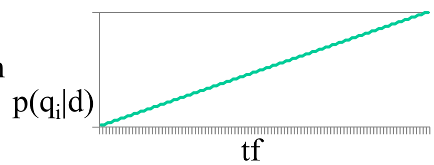
$$p(q_i \mid d) = \frac{tf_{q_i,d} + \mu \, p_{MLE}(q_i \mid C)}{length(d) + \mu}$$

How important is smoothing to <u>documents</u> of different lengths?

- Short documents
  - Probabilities are more granular
  - Larger μ is more important



- Long documents
  - Probabilities are more smooth
  - Larger μ is less important



25

Estimating $p(q_i|d)$: Two-Stage Smoothing

Mixture modeling and Bayesian smoothing with Dirichlet priors are both effective and common … which should you use?

- **They do different things**
  - **Mixture model:** Compensate for differences in word importance
    - » "idf effects"
  - **Dirichlet prior:** Improves the estimate of the document model
    - » "small sample", "unseen words"
- **Two-stage smoothing gives the best effects of both methods**

$$p(q_i \mid d) = (1-\lambda)\frac{tf_{q_i,d} + \mu \, p_{MLE}(q_i \mid C)}{length(d) + \mu} + \lambda \, p_{MLE}(q_i \mid C)$$

26

Query Likelihood With Two-Stage Smoothing: Putting it All Together

$$p(q \mid d) = \prod_{q_i \in q} p(q_i \mid d)$$

$$= \prod_{q_i \in q} \left((1-\lambda) \frac{tf_{q_i,d} + \mu \, p_{MLE}(q_i \mid C)}{length(d) + \mu} + \lambda \, p_{MLE}(q_i \mid C) \right)$$

$$= \prod_{q_i \in q} \left((1-\lambda) \frac{tf_{q_i,d} + \mu \frac{ctf(q_i)}{length(c)}}{length(d) + \mu} + \lambda \frac{ctf(q_i)}{length(c)} \right)$$

$tf_{q,d}$: Term frequency of term q in document d
ctf (q): Term frequency of term q in the entire collection
length (d): Total number of word occurrences in document d
length (c): Total number of word occurrences in collection c

Outline

Statistical language models
- Introduction to language models
- Query likelihood
- Kullback-Leibler (KL) Divergence
- Indri

Retrieval Model Based Upon Statistical Language Models

How is a language model used to rank document d?

- Rank d by $p(d \mid \theta_q)$ "query likelihood"
- Rank d by $p(q \mid \theta_d)$
- Rank d by the similarity of θ_d and θ_q "KL divergence"

In some cases, some of these approaches turn out to be equivalent…

KL Divergence: Measuring the Similarity of Language Models

Kullback-Leibler divergence measures the relative entropy between two probability mass functions

$$KL(p \,\|\, q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}$$

Language models are probability mass functions

- The probability of observing a term in a sample of text
- Why not use KL divergence to measure the similarity of q and d?

Measuring the Similarity of Statistical Language Models

Ranking by (negative) KL-divergence

$$
\begin{aligned}
-KL(q \,\|\, d) &= -\sum_{w \in V} p(w \mid q) \log \frac{p(w \mid q)}{p(w \mid d)} \\
&= -\sum_{q_i \in q} p(q_i \mid q) \log \frac{p(q_i \mid q)}{p(q_i \mid d)} && \textbf{P(w|q) = 0 when w} \notin \textbf{q} \\
&= -\left(\sum_{q_i \in q} p(q_i \mid q) \log p(q_i \mid q) - \sum_{q_i \in q} p(q_i \mid q) \log p(q_i \mid d) \right) \\
&\propto \sum_{q_i \in q} p(q_i \mid q) \log p(q_i \mid d) && \textbf{Drop constant term} \\
&\propto \sum_{q_i \in q} \frac{1}{|q|} \log p(q_i \mid d) && \textbf{Uniform probabilities for query terms} \\
&\propto \sum_{q_i \in q} \log p(q_i \mid d) && \textbf{Drop constant term}
\end{aligned}
$$

31

Two Different Paths to a Common Destination

Query likelihood ranks by $p(q \mid d) = \prod_{q_i \in q} p(q_i \mid d)$

KL diverge ranks by $\sum_{q_i \in q} \log p(q_i \mid d)$

These are rank equivalent

$$
\begin{aligned}
p(q \mid d) &= \prod_{q_i \in q} p(q_i \mid d) \\
&\propto \sum_{q_i \in q} \log p(q_i \mid d)
\end{aligned}
$$

So…the query likelihood and KL divergence approaches are equivalent (when document priors are uniform)

32

Outline

Statistical language models

- Introduction to language models
- Query likelihood
- Kullback-Leibler (KL) Divergence
- Indri

Inference Nets + Language Models: Indri

The Indri retrieval model combines statistical language models with Bayesian inference networks

- A probabilistic retrieval model
- Structured queries
- Documents with multiple representations
- Documents with structure (a later lecture)

The theory is sophisticated, but the underlying ideas are familiar

Inference Nets + Language Models: Indri

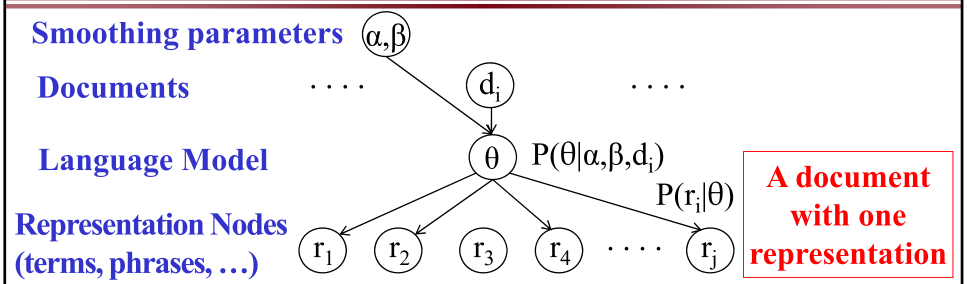
Documents . . . . d_i . . . .

Inference Nets + Language Models: Indri

Documents . . . . d_i . . . .

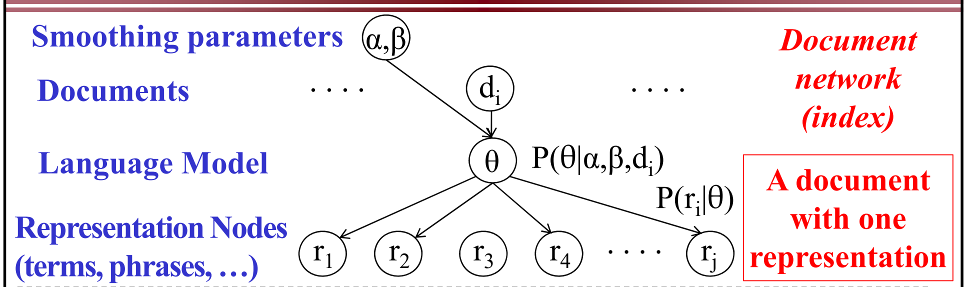
Representation Nodes (terms, phrases, …) r_1 r_2 r_3 r_4 . . . . r_j

Inference Nets + Language Models: Indri



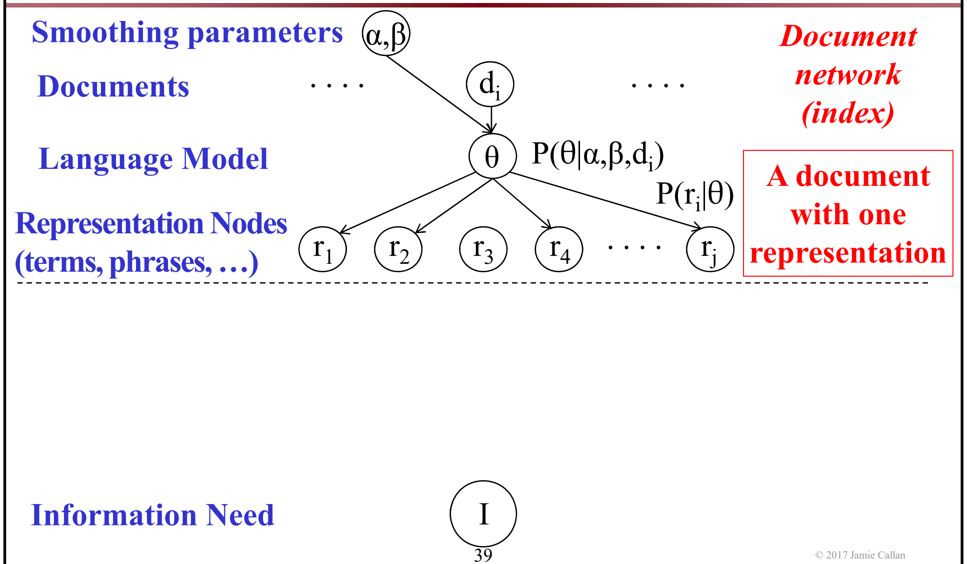
37

Inference Nets + Language Models: Indri



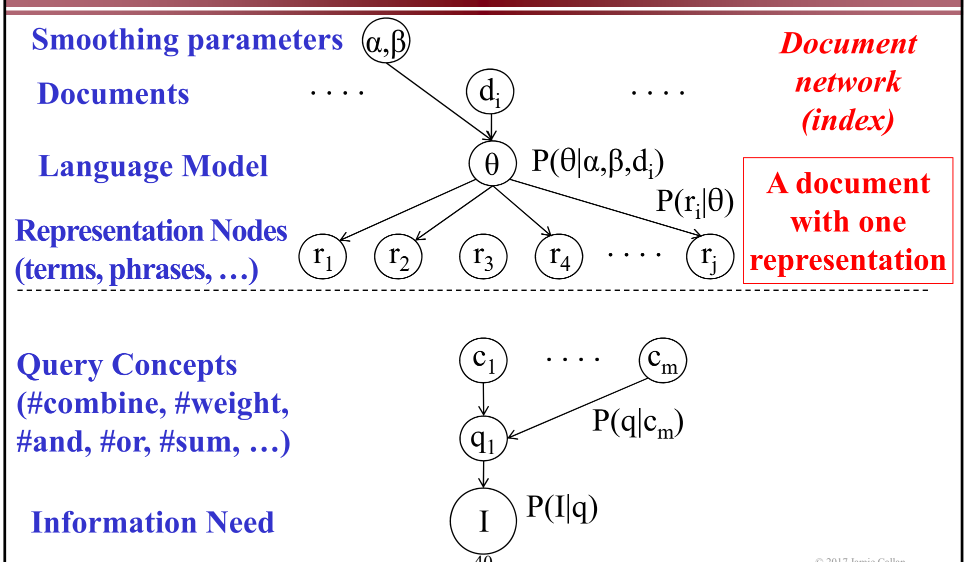
38

Inference Nets + Language Models: Indri



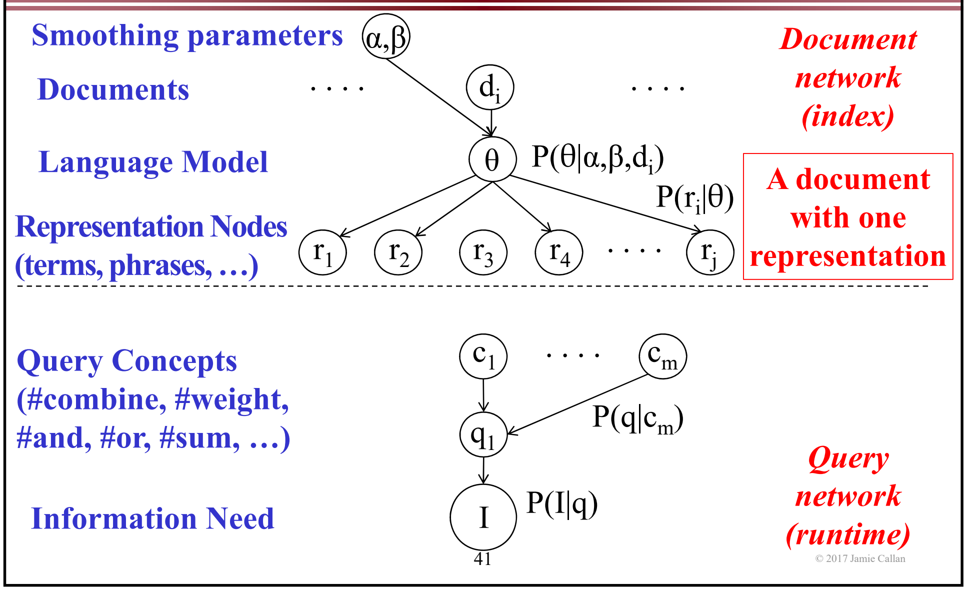
39

Inference Nets + Language Models: Indri



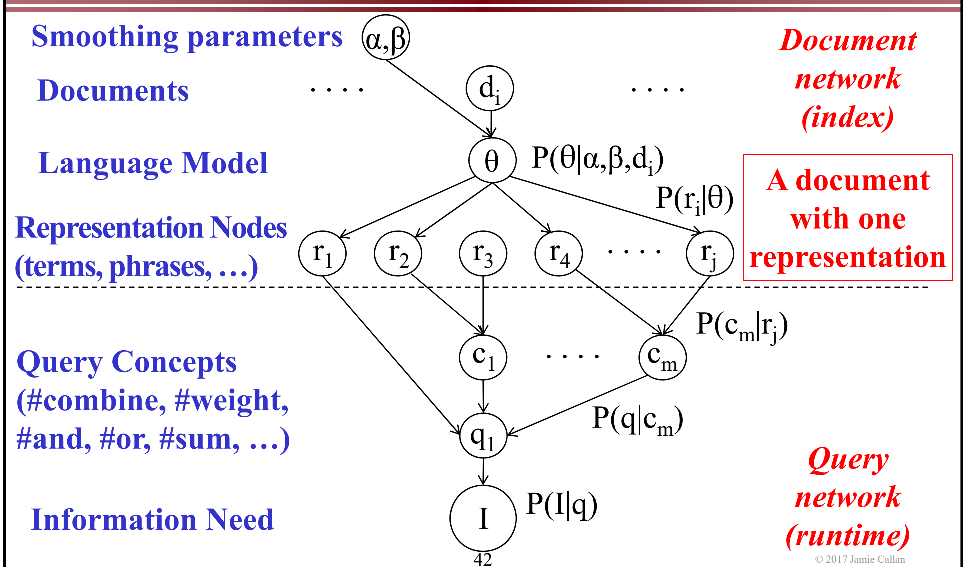
40

Inference Nets + Language Models: Indri



41

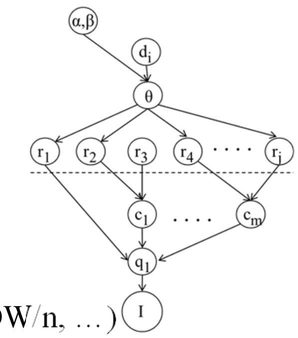
Inference Nets + Language Models: Indri



42

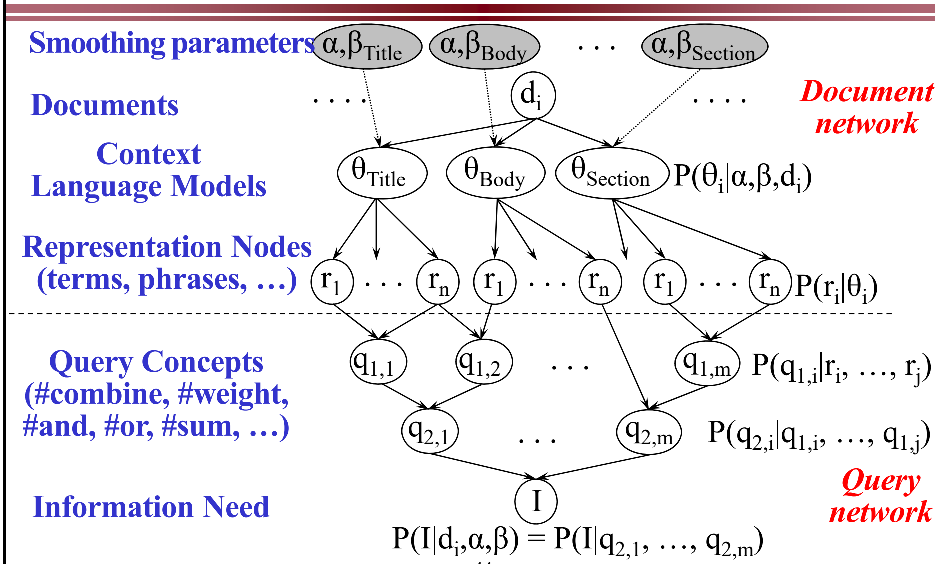
Inference Nets + Language Models: Indri

This isn't as complicated as it looks



- Document + smoothing parameters (α,β) → language model (θ)
- The language model vocabulary is defined by representation nodes (r_i)
  - Terms stored in the index
  - Query operators that <u>create index terms</u>
    - » QryIop (#SYN, #NEAR/n, #WINDOW/n, …)
- Information needs are represented by queries
- Queries are composed of query operators that <u>combine evidence</u>
    - » QrySop (#AND, #OR, #SUM, #WSUM, …)

Inference Nets + Language Models: Indri



Indri Term Weights

Indri can use any language modeling method to estimate $p(r_i|\theta)$

- This weight is equivalent to $p(q_i|d)$ in most language models
- Dirichlet smoothing is the most common choice

$$p(q_i \mid d) = \frac{tf_{q_i,d} + \mu\, p_{MLE}(q_i \mid C)}{length(d) + \mu}$$

- Two-stage smoothing is also used

$$p(q_i \mid d) = (1-\lambda)\frac{tf_{q_i,d} + \mu\, p_{MLE}(q_i \mid C)}{length(d) + \mu} + \lambda\, p_{MLE}(q_i \mid C)$$

45

Indri Query Operators

Operators that map inverted lists to an inverted list are easy

- E.g., InvertedList+ → InvertedList
- E.g., #NEAR/n, #WINDOW/n, #SYN, …
- To the language model these look just like ordinary terms

Operators that combine scores are somewhat less clear

- E.g., ScoreList+ → ScoreList
- E.g., #AND, #OR, …
- How should each query operator combine evidence?

46

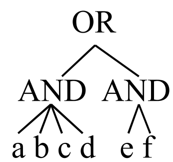
Indri Query Operators

AND is the default query operator for most language modeling systems

- Typically implemented as the product of the argument weights

$$p_{and}(q \mid d) = \prod_{q_i \in q} p(q_i \mid d)$$

- This is pleasing theoretically, but it has a problem
  - Typically #AND (a b c d) < #AND (a b)



Indri's AND operator uses the geometric mean

$$p_{and}(q \mid d) = \prod_{q_i \in q} p(q_i \mid d)^{\boxed{\frac{1}{|q|}}}$$

Indri Query Operators

We may want to give different weight to different evidence

#wand (
0.7 #and (time traveler wife)
0.2 #and (#near/1 (time traveler) #near/1 (traveler wife))
0.1 #and (#window/8 (time traveler) #window/8 (traveler wife)))

Indri's WAND operator (also called WEIGHT) gives this control

- WAND: weighted AND

$$p_{wand}(q \mid d) = \prod_{q_i \in q} p(q_i \mid d)^{\frac{w_i}{\sum w_i}}$$

Indri Query Operators

Indri provides a NOT operator

$$p_{not}(q \mid d) = 1 - p(q \mid d)$$

- NOT operators aren't used a lot in probabilistic systems

Indri provides an OR operator

$$p_{or}(q \mid d) = 1 - \prod_{q_i \in q} (1 - p(q_i \mid d))$$

Indri Query Operators

The AND operator assumes that its arguments are estimates of independent probabilities

- E.g., #AND (buy ipod)

The WSUM operator assumes that its arguments are different ways of estimating the same probability

- E.g., the probability that this document is about "apple")
  #WSUM (0.3 apple.title 0.1 apple.url 0.6 apple.body)
- WSUM takes a weighted average of the estimates
  – It should be called WAVG – the name is a historical artifact

$$p_{wsum}(q \mid d) = \sum_{q_i \in q} \frac{w_i}{\sum w_i} p(q_i \mid d)$$

Indri Query Operators

$$\text{AND, COMBINE}: \quad p_{and}(q \mid d) = \prod_{q_i \in q} p(q_i \mid d)^{\frac{1}{|q|}}$$

$$\text{WAND, WEIGHT}: \quad p_{wand}(q \mid d) = \prod_{q_i \in q} p(q_i \mid d)^{\frac{w_i}{w}}, \quad w = \sum w_i$$

$$\text{OR}: \quad p_{or}(q \mid d) = 1 - \prod_{q_i \in q} \left(1 - p(q_i \mid d)\right)$$

$$\text{WSUM}: \quad p_{wsum}(q \mid d) = \sum_{q_i \in q} \frac{w_i}{w} p(q_i \mid d), \quad w = \sum w_i$$

$$\text{NOT}: \quad p_{not}(q \mid d) = 1 - p(q \mid d)$$

51

Indri Implementation

Mostly the Indri query operators are easy to implement

- See the preceding slide for the score calculations
- **Calculate scores only for documents that contain a query term**
  - Use inverted or score lists – similar to HW1
- Use document length, ctf, and corpus length for smoothing
  - Lookup from the index – see the HW2 web page

But, one aspect is a little tricky to get right…

52

Indri Implementation

Query: #or (a #and (b c)) **Document:** a

Query terms b and c do not appear in this document
… what is the score of the #AND operator?

- $tf_{a,d} = 1$ $tf_{b,d} = 0$ $tf_{c,d} = 0$
- Do the usual Indri score calculation
  – So, only smoothing scores for b and c

This is simple conceptually, but how is it implemented?

- You don't want to calculate #AND scores for every document
  … just the documents that have at least one query term

53

Indri Implementation

Query: #or (a #and (b c)) **Document:** a

Add a new method to all QrySop operators

double getDefaultScore (RetrievalModel r, long docid)

When any QrySop operator calculates scores

If the i^{th} query argument contains document d
then call the i^{th} query argument's getScore method
else call the i^{th} query argument's getDefaultScore method

54

Indri Implementation

Query: #or (a #and (b c)) **Document:** a

QrySopScore.getDefaultScore (RetrievalModel r, long docid)

- The standard Indri SCORE calculation done with tf=0

  If r == RetrievalModel.Indri

$$p_{scoreDefault}(t \mid \text{docid}) = (1-\lambda)\frac{\boxed{0} + \mu\, p_{MLE}(t \mid C)}{length(\text{docid}) + \mu} + \lambda\, p_{MLE}(t \mid C)$$

This is the only difference.
Do the usual calculation, but with tf=0.

Indri Implementation

Query: #or (a #and (b c)) **Document:** a

QrySopAnd.getDefaultScore (RetrievalModel r, long docid)

- The standard Indri AND calculation done on the default score of each argument

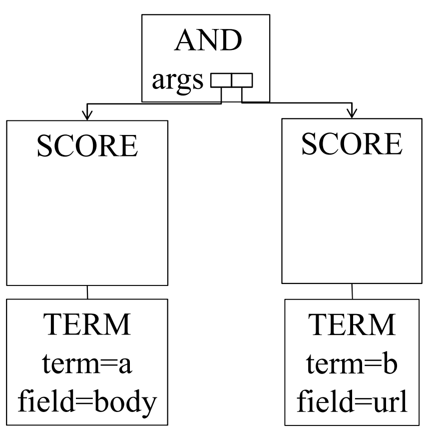
  If r == RetrievalModel.Indri

$$p_{andDefault}(q \mid d) = \prod_{q_i \in q} \boxed{p_{q_i default}}(q_i \mid d)^{\frac{1}{|q|}}$$

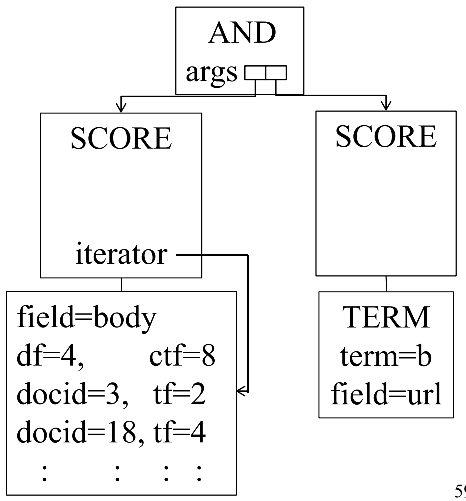
This is the only difference.
Call the i[th] query argument's getDefaultScore method.

QryEval Example

The initial query: #AND (a.body b.url)



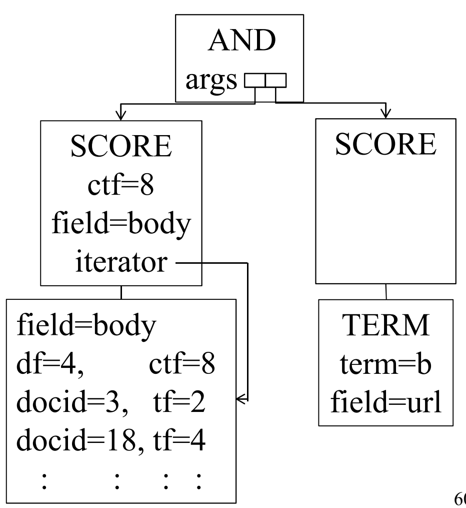
Query Initialization



AND operator initialization

AND initializes its first arg.
SCORE initializes its arg.
The result is an inverted list.

Query Initialization

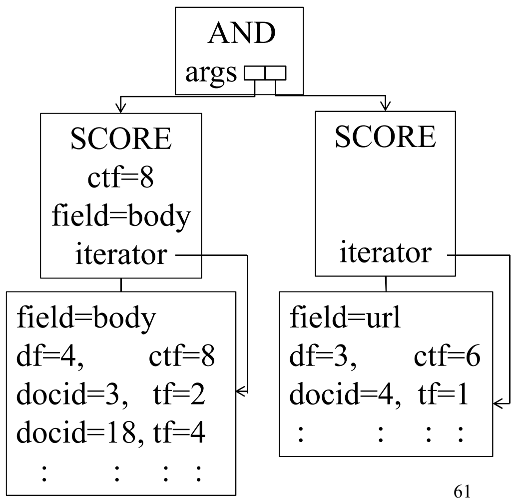


AND operator initialization

AND initializes its first arg.
SCORE initializes its arg.
The result is an inverted list.

The SCORE operator caches information that it will need later.

Query Initialization

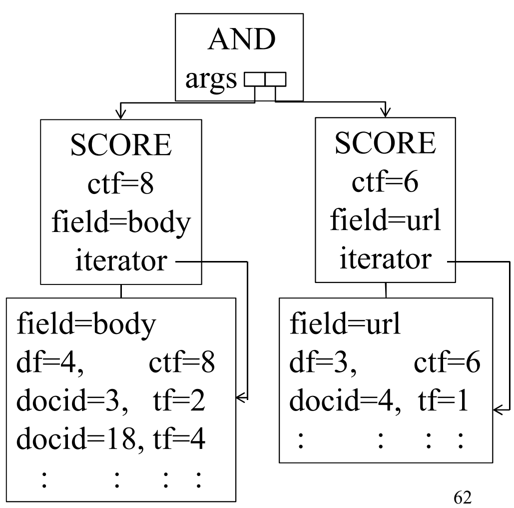


AND operator initialization

AND initializes its second arg.
SCORE initializes its arg.
The result is an inverted list.

61

Query Initialization



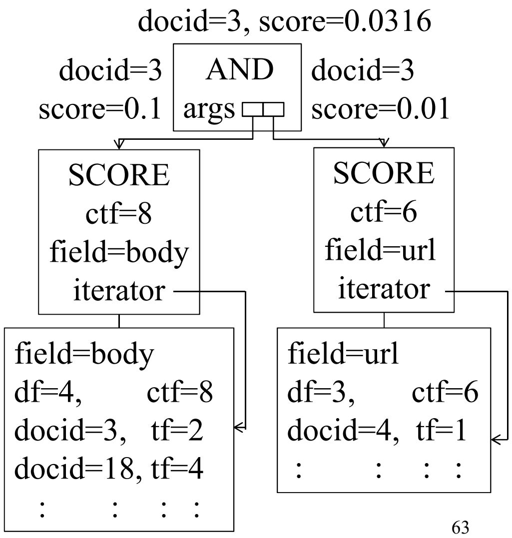
AND operator initialization

AND initializes its second arg.
SCORE initializes its arg.
The result is an inverted list.

The SCORE operator caches information that it will need later.

62

Call to docIteratorHasMatch & getScore (First Time)



AND Operator Evaluation

Min document is 3.

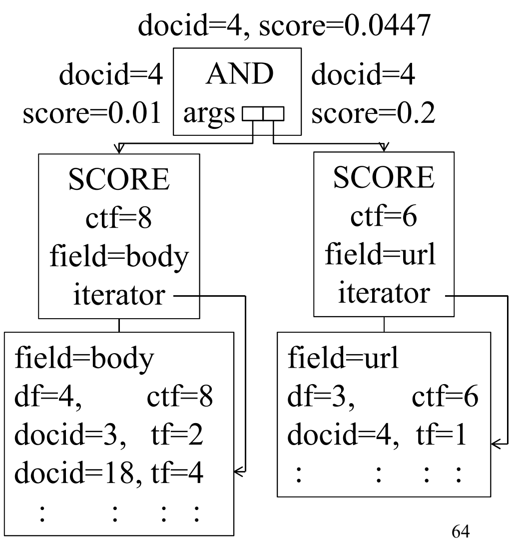
args[0] matches, so call args[0].getScore (). Suppose the result is 0.1.

args[1] does not match, so call args[1].getDefaultScore(3). Suppose the result is 0.01.

$\text{Score}_{\text{AND}}(3) = (0.1^{0.5} \times 0.01^{0.5})$

63

Call to docIteratorHasMatch & getScore (Second Time)



AND Operator Evaluation

Min document is 4.

args[0] does not match, so call args[0].getDefaultScore (4). Suppose the result is 0.01.

args[1] matches, so call args[1].getScore(). Suppose the result is 0.2.

$\text{Score}_{\text{AND}}(4) = (0.01^{0.5} \times 0.2^{0.5})$

64

Default Belief Scores Are Only a Small Complication

When evaluating a query argument

- If it matches the current document
  - Ask the query argument to calculate the document score for the current document
  - Else ask the query argument to calculate a default score for the current document
    - » E.g., a SCORE operator (the example given)
    - » E.g., an OR operator (similar logic)

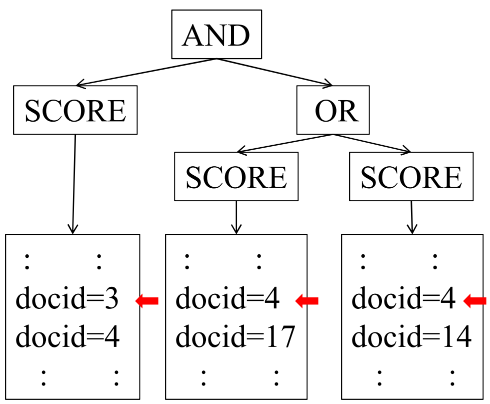
Default Belief Scores Are Only a Small Complication

Which types of query operators calculate default scores?

- If an operator calculates scores

  … it also calculates default scores
- QrySop operators calculate default scores
- QryIop operators do not calculate default scores

Call to docIteratorHasMatch & getScore (First Time)



AND Operator Evaluation

Min document is 3.

args[0] matches, so call args[0].getScore ().
Suppose the result is 0.3.

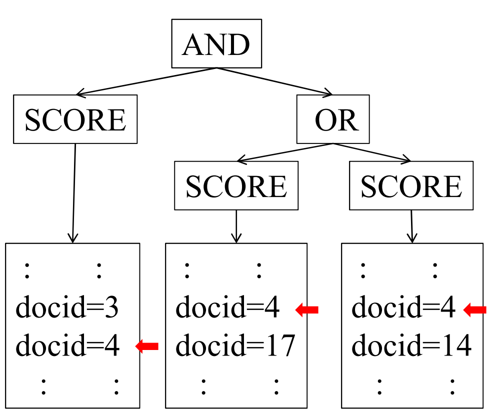
args[1] does not match, so call args[1].getDefaultScore(3).

OR calls getDefaultScore(3) for all of its args and computes a score. Suppose it is 0.01.

$\text{Score}_{\text{AND}}(3) = (0.3^{0.5} \times 0.01^{0.5})$

67

Call to docIteratorHasMatch & getScore (Second Time)



AND Operator Evaluation

Min document is 4.

args[0] matches, so call args[0].getScore ().
Suppose the result is 0.3.

args[1] matches, so call args[1].getScore().
Suppose the score is 0.2.

$\text{Score}_{\text{AND}}(4) = (0.3^{0.5} \times 0.2^{0.5})$

68

Using Default Belief Scores Properly Requires Two Components

Add a new method to all QrySop operators

double getDefaultScore (RetrievalModel r, long docid)

– QrySopScore.getDefaultScore calculates a score for a term

– QrySop <other>. getDefaultScore combines scores for *n* terms

When any QrySop operator calculates scores

If the i^{th} query argument contains document d

then read its score from the i^{th} score list

else call the i^{th} query argument's getDefaultScore method

It may sound complicated now, but actually it is very easy

Outline

Statistical language models

- Introduction to language models
- Query likelihood
- Kullback-Leibler (KL) Divergence
- Indri

Retrieval Models Summary

We have discussed the following retrieval models

- Unranked Boolean
- Ranked Boolean, using tf scoring
- Vector Space, using lnc.ltc scoring
- Okapi BM25
- Language Models
  - Query likelihood, with two-stage smoothing
  - KL Divergence and Jensen-Shannon Divergence
  - Indri

That's a lot of material … what's important?

Retrieval Models Summary

Important differences among the models that you should know

- Boolean vs. ranked retrieval
- The kinds of statistics used by most ranking functions
  - The theories are different, but the stats are mostly the same
- How well the different models support query operators
  - Inverted-list operators vs. score operators
  - Strict Boolean vs. probabilistic Boolean
- Which models you could use (or not use) for different types of tasks

Retrieval Models Summary

All of these retrieval models are used widely

- There must be a reason why … you should know what it is