Lemur Toolkit Tutorial

Introductions

- Paul Ogilvie
- Trevor Strohman

Installation

- Linux, OS/X:
  - Extract software/lemur-4.3.2.tar.gz
  - ./configure --prefix=/install/path
  - ./make
  - ./make install
- Windows
  - Run software/lemur-4.3.2-install.exe
  - Documentation in windoc/index.html

Overview

- Background in Language Modeling in Information Retrieval
- Basic application usage
  - Building an index
  - Running queries
  - Evaluating results
- Indri query language
- Coffee break

Overview (part 2)

- Indexing your own data
- Using ParsedDocument
- Indexing document fields
- Using dumpindex
- Using the Indri and classic Lemur APIs
- Getting help

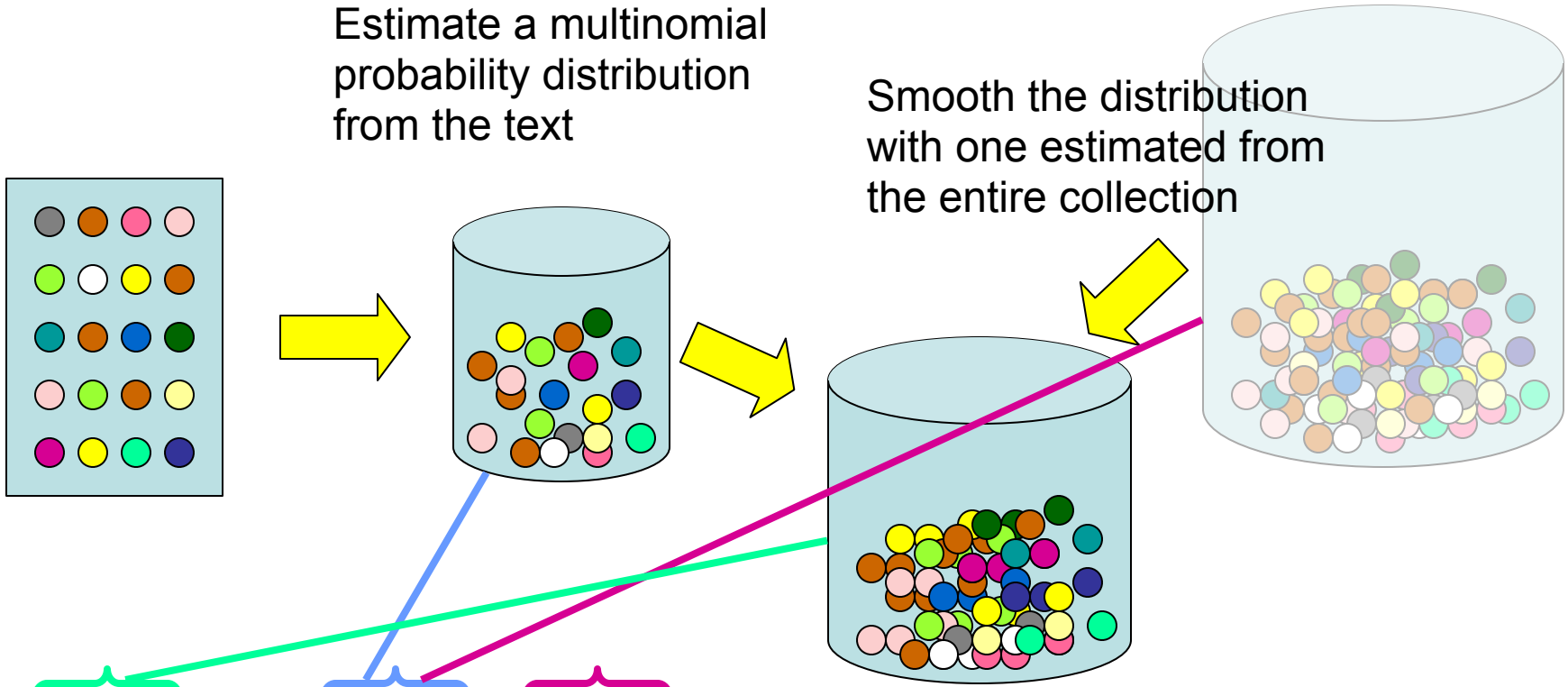
Overview

- Background
  - The Toolkit
  - Language Modeling in Information Retrieval
- Basic application usage
  - Building an index
  - Running queries
  - Evaluating results
- Indri query language
- Coffee break

Language Modeling for IR

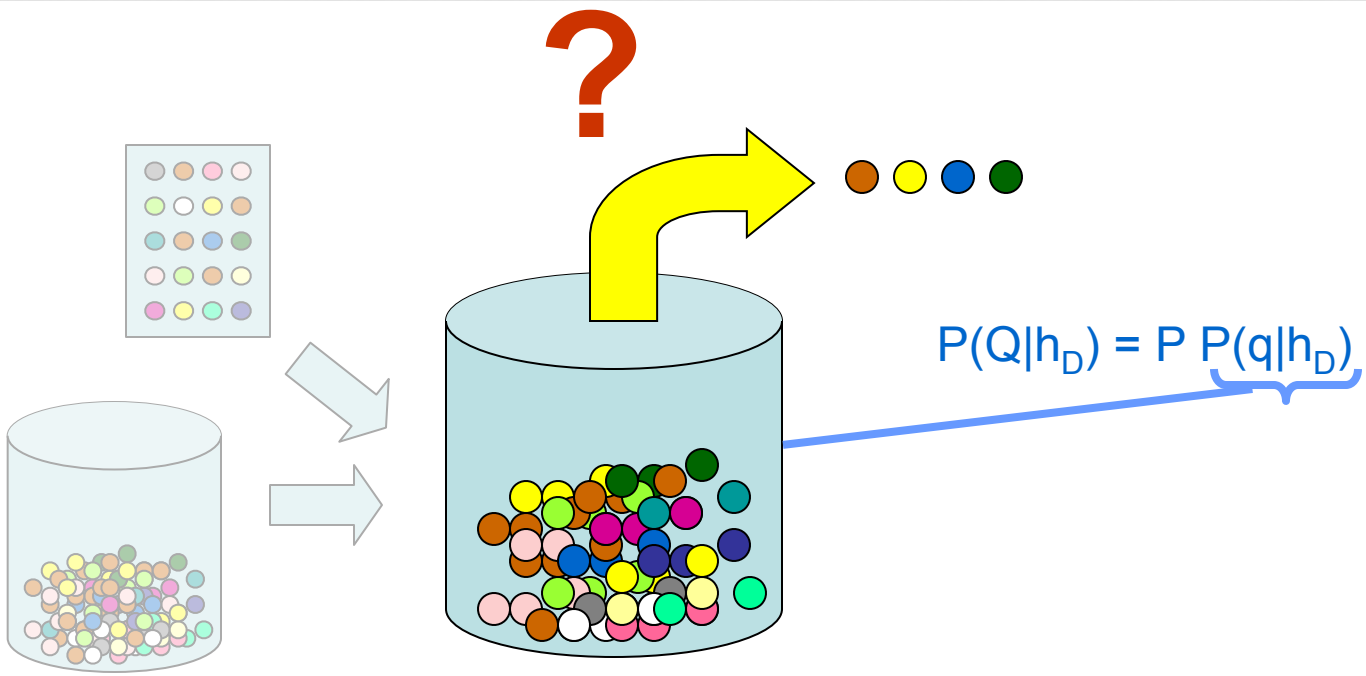
Estimate a multinomial probability distribution from the text

Smooth the distribution with one estimated from the entire collection



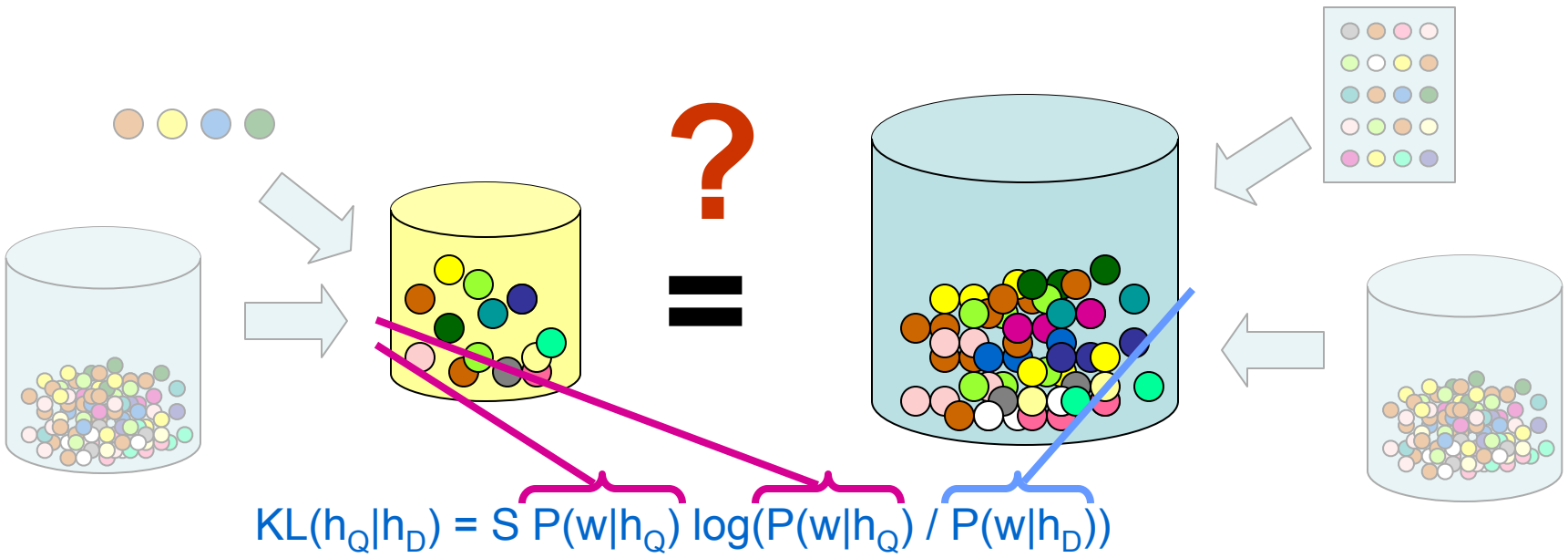
$$P(w|h_D) = (1-k)\,P(w|D) + k\,P(w|C)$$

Query Likelihood



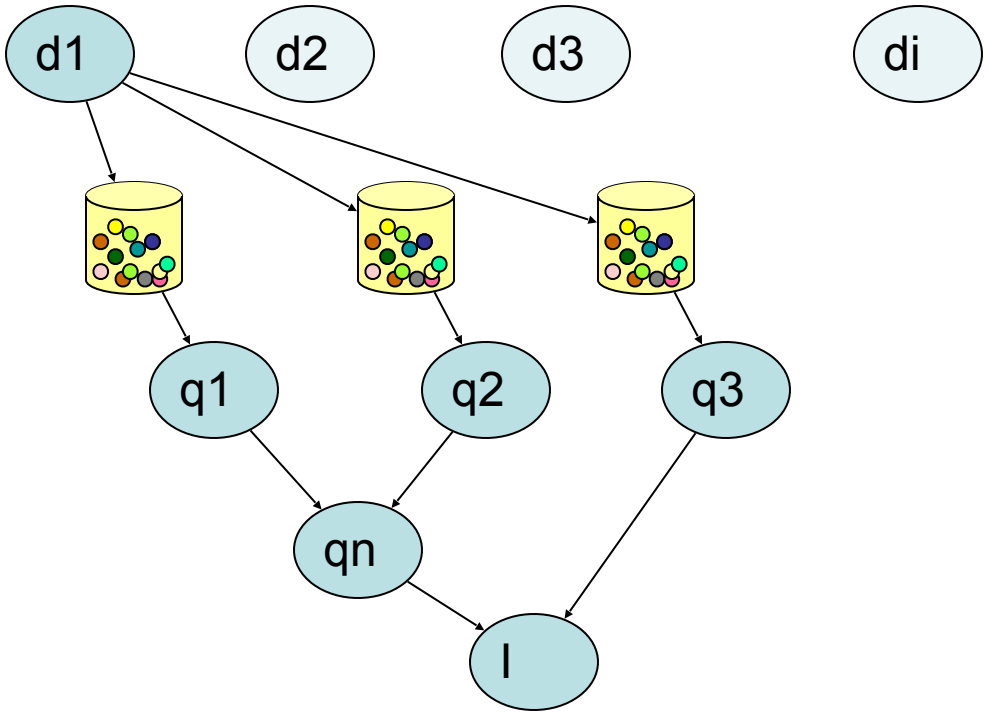
- Estimate probability that document generated the query terms

Kullback-Leibler Divergence



- Estimate models for document and query and compare

Inference Networks



- Language models used to estimate beliefs of representation nodes

Summary of Ranking

- Techniques use simple multinomial probability distributions to model vocabulary usage
- The distributions are smoothed with a collection model to prevent zero probabilities
  - This has an idf-like effect on ranking
- Documents are ranked through generative or distribution similarity measures
- Inference networks allow structured queries – beliefs estimated are related to generative probabilities

Other Techniques

- (Pseudo-) Relevance Feedback
  - Relevance Models [Lavrenko 2001]
  - Markov Chains [Lafferty and Zhai 2001]
- *n*-Grams [Song and Croft 1999]
- Term Dependencies [Gao et al 2004, Metzler and Croft 2005]

Overview

- Background
  - The Toolkit
  - Language Modeling in Information Retrieval
- Basic application usage
  - Building an index
  - Running queries
  - Evaluating results
- Indri query language
- Coffee break

Indexing

- Document Preparation
- Indexing Parameters
- Time and Space Requirements

Two Index Formats

KeyFile

- Term Positions
- Metadata
- Offline Incremental
- InQuery Query Language

Indri

- Term Positions
- Metadata
- Fields / Annotations
- Online Incremental
- InQuery and Indri Query Languages

Indexing – Document Preparation

Document Formats:

The Lemur Toolkit can inherently deal with several different document format types without any modification:

- TREC Text
- TREC Web
- Plain Text
- Microsoft Word(*)
- Microsoft PowerPoint(*)
- HTML
- XML
- PDF
- Mbox

(*) Note: Microsoft Word and Microsoft PowerPoint can only be indexed on a Windows-based machine, and Office must be installed.

Indexing – Document Preparation

- If your documents are not in a format that the Lemur Toolkit can inherently process:

1. If necessary, extract the text from the document.
2. Wrap the plaintext in TREC-style wrappers:

```
<DOC>
<DOCNO>document_id</DOCNO>
<TEXT>
  Index this document text.
</TEXT>
</DOC>
```

– or –

For more advanced users, write your own parser to extend the Lemur Toolkit.

Indexing - Parameters

- Basic usage to build index:
  - IndriBuildIndex <parameter_file>
- Parameter file includes options for
  - Where to find your data files
  - Where to place the index
  - How much memory to use
  - Stopword, stemming, fields
  - Many other parameters.

Indexing – Parameters

- Standard parameter file specification an XML document:

```
<parameters>
  <option></option>
  <option></option>
    …
  <option></option>
</parameters>
```

Indexing – Parameters

- **<corpus>** - where to find your source files and what type to expect
  - **<path>:** (required) the path to the source files (absolute or relative)
  - **<class>:** (optional) the document type to expect. If omitted, IndriBuildIndex will attempt to guess at the filetype based on the file's extension.

```
<parameters>
   <corpus>
         <path>/path/to/source/files</path>
         <class>trectext</class>
   </corpus>
</parameters>
```

Indexing - Parameters

- The **<index>** parameter tells IndriBuildIndex where to create or incrementally add to the index
  - If index does not exist, it will create a new one
  - If index already exists, it will append new documents into the index.

```
<parameters>
    <index>/path/to/the/index</index>
</parameters>
```

Indexing - Parameters

- **<memory>** - used to define a "soft-limit" of the amount of memory the indexer should use before flushing its buffers to disk.
  - Use K for kilobytes, M for megabytes, and G for gigabytes.

```
<parameters>
    <memory>256M</memory>
</parameters>
```

Indexing - Parameters

- Stopwords can be defined within a **<stopper>** block with individual stopwords within enclosed in **<word>** tags.

```
<parameters>
        <stopper>
                <word>first_word</word>
                <word>next_word</word>
                        …
                <word>final_word</word>
        </stopper>
</parameters>
```

Indexing – Parameters

- Term stemming can be used while indexing as well via the **<stemmer>** tag.
  - Specify the stemmer type via the **<name>** tag within.
  - Stemmers included with the Lemur Toolkit include the Krovetz Stemmer and the Porter Stemmer.

```
<parameters>
   <stemmer>
        <name>krovetz</name>
   </stemmer>
</parameters>
```

Indexing anchor text

- Run `harvestlinks` application on your data before indexing
- `<inlink>path-to-links</inlink>` as a parameter to IndriBuildIndex to index

Retrieval

- Parameters
- Query Formatting
- Interpreting Results

Retrieval - Parameters

- Basic usage for retrieval:
  - `IndriRunQuery/RetEval <parameter_file>`

- Parameter file includes options for
  - Where to find the index
  - The query or queries
  - How much memory to use
  - Formatting options
  - Many other parameters.

Retrieval - Parameters

- Just as with indexing:
  - A well-formed XML document with options, wrapped by <parameters> tags:

```
<parameters>
  <options></options>
  <options></options>
    …
  <options></options>
</parameters>
```

Retrieval - Parameters

- The **<index>** parameter tells `IndriRunQuery`/`RetEval` where to find the repository.

```
<parameters>
   <index>/path/to/the/index</index>
</parameters>
```

Retrieval - Parameters

- The <query> parameter specifies a query
  - plain text or using the Indri query language

```
<parameters>
    <query>
     <number>1</number>
     <text>this is the first query</text>
   </query>
    <query>
     <number>2</number>
     <text>another query to run</text>
   </query>
</parameters>
```

Retrieval - Parameters

- A free-text query will be interpreted as using the #combine operator
  - "this is a query" will be equivalent to "#combine(this is a query)"
  - More on the Indri query language operators in the next section

Retrieval – Query Formatting

- TREC-style topics are *not* directly able to be processed via `IndriRunQuery`/`RetEval`.
  - Format the queries accordingly:
    - Format by hand
    - Write a script to extract the fields

Retrieval - Parameters

- As with indexing, the **<memory>** parameter can be used to define a "soft-limit" of the amount of memory the retrieval system uses.
  - Use K for kilobytes, M for megabytes, and G for gigabytes.

```
<parameters>
    <memory>256M</memory>
</parameters>
```

Retrieval - Parameters

- As with indexing, stopwords can be defined within a **<stopper>** block with individual stopwords within enclosed in **<word>** tags.

```
<parameters>
        <stopper>
                <word>first_word</word>
                <word>next_word</word>
                        …
                <word>final_word</word>
        </stopper>
</parameters>
```

Retrieval – Parameters

- To specify a maximum number of results to return, use the **<count>** tag:

```
<parameters>
    <count>50</count>
</parameters>
```

Retrieval - Parameters

- Result formatting options:
  - `IndriRunQuery`/`RetEval` has built in formatting specifications for TREC and INEX retrieval tasks

Retrieval – Parameters

- TREC – Formatting directives:
  - **<runID>**: a string specifying the id for a query run, used in TREC scorable output.
  - **<trecFormat>**: `true` to produce TREC scorable output, otherwise use `false` (default).

```
<parameters>
   <runID>runName</runID>
   <trecFormat>true</trecFormat>
</parameters>
```

Outputting INEX Result Format

- Must be wrapped in **<inex>** tags
  - **<participant-id>**: specifies the participant-id attribute used in submissions.
  - **<task>**: specifies the task attribute (default CO.Thorough).
  - **<query>**: specifies the query attribute (default automatic).
  - **<topic-part>**: specifies the topic-part attribute (default T).
  - **<description>**: specifies the contents of the description tag.

```
<parameters>
        <inex>
                <participant-id>LEMUR001</participant-id>
        </inex>
</parameters>
```

Retrieval – Interpreting Results

- The default output from `IndriRunQuery` will return a list of results, 1 result per line, with 4 columns:
  - **`<score>`**: the score of the returned document. An Indri query will always return a negative value for a result.
  - **`<docID>`**: the document ID
  - **`<extent_begin>`**: the starting token number of the extent that was retrieved
  - **`<extent_end>`**: the ending token number of the extent that was retrieved

Retrieval – Interpreting Results

- When executing `IndriRunQuery` with the default formatting options, the output will look something like:

```
<score> <DocID> <extent_begin> <extent_end>
-4.83646 AP890101-0001 0 485
-7.06236 AP890101-0015 0 385
```

Retrieval - Evaluation

- To use trec_eval:
  - format `IndriRunQuery` results with appropriate trec_eval formatting directives in the parameter file:
    - <runID>runName</runID>
    - <trecFormat>true</trecFormat>
- Resulting output will be in standard TREC format ready for evaluation:

```
<queryID> Q0 <DocID> <rank> <score> <runID>

150 Q0 AP890101-0001 1 -4.83646 runName
150 Q0 AP890101-0015 2 -7.06236 runName
```

Smoothing

- <**rule**>method:linear,collectionLambda:0.4,documentLambda:0.2</**rule**>
- <**rule**>method:dirichlet,mu:1000</**rule**>
- <**rule**>method:twostage,mu:1500,lambda:0.4</**rule**>

Use RetEval for TF.IDF

- First run `ParseToFile` to convert doc formatted queries into queries

```
<parameters>
  <docFormat>format</docFormat>
  <outputFile>filename</outputFile>
  <stemmer>stemmername</stemmer>
  <stopwords>stopwordfile</stopwords>
</parameters>
```

- `ParseToFile paramfile queryfile`
- http://www.lemurproject.org/lemur/parsing.html#parsetofile

Use RetEval for TF.IDF

- Then run `RetEval`

```
<parameters>
  <index>index</index>
  <retModel>0</retModel>  // 0 for TF-IDF, 1 for Okapi,
                          // 2 for KL-divergence,
                          // 5 for cosine similarity
  <textQuery>queries.reteval</textQuery>
  <resultCount>1000</resultCount>
  <resultFile>tfidf.res</resultFile>
</parameters>
```

- RetEval paramfile queryfile
- http://www.lemurproject.org/lemur/retrieval.html#RetEval

Overview

- Background
  - The Toolkit
  - Language Modeling in Information Retrieval
- Basic application usage
  - Building an index
  - Running queries
  - Evaluating results
- Indri query language
- Coffee break

Indri Query Language

- terms
- field restriction / evaluation
- numeric
- combining beliefs
- field / passage retrieval
- filters
- document priors

http://www.lemurproject.org/lemur/IndriQueryLanguage.html

Term Operations

name	example	behavior
term	`dog`	occurrences of `dog` (Indri will stem and stop)
"term"	`"dog"`	occurrences of `dog` (Indri will not stem or stop)
ordered window	`#od`*n*`(blue car)`	`blue` *n* words or less before `car`
unordered window	`#ud`*n*`(blue car)`	`blue` within *n* words of `car`
synonym list	`#syn(car automobile)`	occurrences of `car` or `automobile`
weighted synonym	`#wsyn(1.0 car 0.5 automobile)`	like synonym, but only counts occurrences of `automobile` as 0.5 of an occurrence
any operator	`#any:person`	all occurrences of the `person` field

Field Restriction/Evaluation

name	example	behavior
restriction	`dog.title`	counts only occurrences of `dog` in `title` field
	`dog.title,header`	counts occurrences of `dog` in `title` or `header`
evaluation	`dog.(title)`	builds belief *b*`(dog)` using `title` language model
	`dog.(title,header)`	*b*`(dog)` estimated using language model from concatenation of all `title` and `header` fields
`#od1(trevor strohman).person(title)`		builds a model from all `title` text for *b*`(#od1(trevor strohman).person)` - only counts "`trevor strohman`" occurrences in `person` fields

Numeric Operators

name	example	behavior
less	`#less(year 2000)`	occurrences of `year` field < 2000
greater	`#greater(year 2000)`	`year` field > 2000
between	`#between(year 1990 2000)`	1990 < `year` field < 2000
equals	`#equals(year 2000)`	`year` field = 2000

Belief Operations

name	example	behavior
combine	`#combine(dog train)`	0.5 log(*b*`(dog)`) + 0.5 log(*b*`(train)`)
weight, wand	`#weight(1.0 dog 0.5 train)`	0.67 log(*b*`(dog)`) + 0.33 log(*b*`(train)`)
wsum	`#wsum(1.0 dog 0.5 dog.(title))`	log(0.67 *b*`(dog)` + 0.33 *b*`(dog.(title))`)
not	`#not(dog)`	log(1 - *b*`(dog)`)
max	`#max(dog train)`	returns maximum of *b*`(dog)` and *b*`(train)`
or	`#or(dog cat)`	log(1 - (1 - *b*`(dog)`) * (1 - *b*`(cat)`))

Field/Passage Retrieval

name	example	behavior
field retrieval	`#combine[title] ( query )`	return only title fields ranked according to `#combine(query)` - beliefs are estimated on each `title`'s language model -may use any belief node
passage retrieval	`#combine[passage200:100]( query )`	dynamically created passages of length `200` created every `100` words are ranked by `#combine(query)`

More Field/Passage Retrieval

example	behavior
`#combine[section]( bootstrap #combine[./title] ( methodology ))`	Rank sections matching `bootstrap` where the section's `title` also matches `methodology`

- `.//field` for ancestor
- `.\field` for parent

Filter Operations

name	example	behavior
filter require	`#filreq(elvis #combine(blue shoes))`	rank documents that contain `elvis` by `#combine(blue shoes)`
filter reject	`#filrej(shopping #combine(blue shoes))`	rank documents that do not contain `shopping` by `#combine(blue shoes)`

Document Priors

name	example	behavior
prior	`#combine(#prior(RECENT) global warming)`	treated as any belief during ranking – `RECENT` prior could give higher scores to more recent documents

`RECENT` prior built using `makeprior` application

Ad Hoc Retrieval

- Query likelihood
- `#combine( literacy rates africa )`
- Rank by $P(Q|D) = \Pi_q P(q|D)$

Query Expansion

- #weight(0.75 #combine(literacy rates africa)
  0.25 #combine(additional terms))

Known Entity Search

- Mixture of multinomials
- ```
  #combine( #wsum( 0.5 bbc.(title)
                   0.3 bbc.(anchor)
                   0.2 bbc )
            #wsum( 0.5 news.(title)
                   0.3 news.(anchor)
                   0.2 news ) )
  ```
- $P(q|D) = 0.5\ P(q|\text{title}) + 0.3\ P(q|\text{anchor}) + 0.2\ P(q|\text{news})$
  ```

Overview

- Background
  - The Toolkit
  - Language Modeling in Information Retrieval
- Basic application usage
  - Building an index
  - Running queries
  - Evaluating results
- Indri query language
- Coffee break

Overview (part 2)

- Indexing your own data
- Using ParsedDocument
- Indexing document fields
- Using dumpindex
- Using the Indri and classic Lemur APIs
- Getting help

Indexing Your Data

- PDF, Word documents, PowerPoint, HTML
  - Use IndriBuildIndex to index your data directly
- TREC collection
  - Use IndriBuildIndex or BuildIndex
- Large text corpus
  - Many different options

Indexing Text Corpora

- Split data into one XML file per document
  - Pro: Easiest option
  - Pro: Use any language you like (Perl, Python)
  - Con: Not very efficient
- For efficiency, large files are preferred
  - Small files cause internal filesystem fragmentation
  - Small files are harder to open and read efficiently

Indexing: Offset Annotation

- Tag data does not have to be in the file
  - Add extra tag data using an offset annotation file
- Format:

```
docno type id name start length value parent
```

- Example:
  - DOC001 TAG 1 title 10 50 0 0
  - “Add a title tag to DOC001 starting at byte 10 and continuing for 50 bytes”

Indexing Text Corpora

- Format data in TREC format
  - Pro: Almost as easy as individual XML docs
  - Pro: Use any language you like
  - Con: Not great for online applications
    - Direct news feeds
    - Data comes from a database

Indexing Text Corpora

- Write your own parser
  - Pro: Fast
  - Pro: Best flexibility, both in integration and in data interpretation
  - Con: Hardest option
  - Con: Smallest language choice (C++ or Java)

Overview (part 2)

- Indexing your own data
- Using ParsedDocument
- Indexing document fields
- Using dumpindex
- Using the Indri and classic Lemur APIs
- Getting help

ParsedDocument

```
struct ParsedDocument {
    const char* text;
    size_t textLength;

    indri::utility::greedy_vector<char*> terms;
    indri::utility::greedy_vector<indri::parse::TagExtent*> tags;
    indri::utility::greedy_vector<indri::parse::TermExtent> positions;
    indri::utility::greedy_vector<indri::parse::MetadataPair> metadata;
  };
```

ParsedDocument: Text

```
const char* text;
size_t textLength;
```

- A null-terminated string of document text
- Text is compressed and stored in the index for later use (such as snippet generation)

ParsedDocument: Content

```
const char* content;
size_t contentLength;
```

- A string of document text
- This is a substring of text; this is used in case the whole text string is not the core document
  - For instance, maybe the text string includes excess XML markup, but the content section is the primary text

ParsedDocument: Terms

indri::utility::greedy_vector<char*> terms;

document = “My dog has fleas.”
terms = { “My”, “dog”, “has”, “fleas” }

- A list of terms in the document
  - Order matters – word order will be used in term proximity operators
- A greedy_vector is effectively an STL vector with a different memory allocation policy

ParsedDocument: Terms

```
indri::utility::greedy_vector<char*> terms;
```

- Term data will be normalized (downcased, some punctuation removed) later
- Stopping and stemming can be handled within the indexer
- Parser's job is just tokenization

ParsedDocument: Tags

indri::utility::greedy_vector<indri::parse::TagExtent*> tags;

TagExtent:
const char* name;
unsigned int begin;
unsigned int end;
INT64 number;
TagExtent *parent;
greedy_vector<AttributeValuePair> attributes;

ParsedDocument: Tags

name

The name of the tag

begin, end

Word offsets (relative to content) of the beginning and end name of the tag.

My **<animal>**dirty dog**</animal>** has fleas.

name = “animal”, begin = 2, end = 3

ParsedDocument: Tags

number

A numeric component of the tag (optional)

sample document

This document was written in **<year>**2006**</year>**.

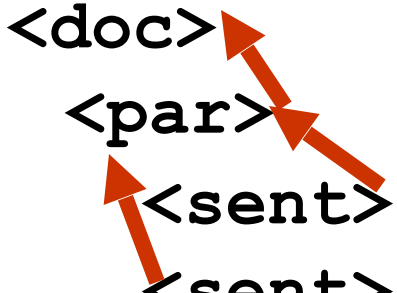
sample query

#between(year 2005 2007)

ParsedDocument: Tags

parent

The logical parent of the tag



```
<doc>
  <par>
    <sent>My dog still has fleas.</sent>
    <sent>My cat does not have fleas.</sent>
  </par>
</doc>
```

ParsedDocument: Tags

attributes

Attributes of the tag

My ****home page****.

Note: Indri cannot index tag attributes. They are used for conflation and extraction purposes only.

ParsedDocument: Tags

attributes

Attributes of the tag

My ****home page****.

Note: Indri cannot index tag attributes. They are used for conflation and extraction purposes only.

ParsedDocument: Metadata

greedy_vector<indri::parse::MetadataPair> metadata

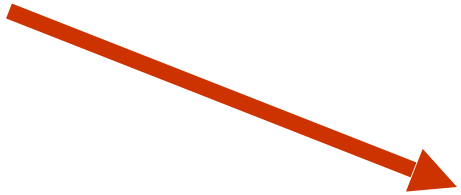
- Metadata is text about a document that should be kept, but not indexed:
  - TREC Document ID (WTX001-B01-00)
  - Document URL
  - Crawl date

Overview (part 2)

- Indexing your own data
- Using ParsedDocument
- Indexing document fields
- Using dumpindex
- Using the Indri and classic Lemur APIs
- Getting help

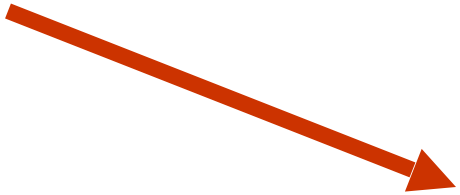
Tag Conflation

<ENAMEX TYPE="ORGANIZATION">



<ORGANIZATION>

<ENAMEX TYPE="PERSON">



<PERSON>

Indexing Fields

- Parameters:
  - **Name**: name of the XML tag, all lowercase
  - **Numeric**: whether this field can be retrieved using the numeric operators, like #between and #less
  - **Forward**: true if this field should be efficiently retrievable given the document number
    - See QueryEnvironment::documentMetadata
  - **Backward**: true if this document should be retrievable given this field data
    - See QueryEnvironment::documentsFromMetadata

Indexing Fields

```
<parameters>
    <field>
        <name>title</name>
        <backward>true</backward>
    <field>
    <field>
        <name>gradelevel</name>
        <numeric>true</name>
    </field>
</parameters>
```

Overview (part 2)

- Indexing your own data
- Using ParsedDocument
- Indexing document fields
- Using dumpindex
- Using the Indri and classic Lemur APIs
- Getting help

dumpindex

- dumpindex is a versatile and useful tool
  - Use it to explore your data
  - Use it to verify the contents of your index
  - Use it to extract information from the index for use outside of Lemur

dumpindex

- Extracting the vocabulary

```
% dumpindex ap89 v
TOTAL 39192948 84678
the 2432559 84413
of 1063804 83389
to 1006760 82505
a 898999 82712
and 877433 82531
in 873291 82984
said 505578 76240
```

word term_count doc_count

dumpindex

- Extracting a single term

```
% dumpindex ap89 tp ogilvie
ogilvie ogilvie 8 39192948
6056 1 1027 954
11982 1 619 377
15775 1 155 66
45513 3 519 216 275
55132 1 668 452
65595 1 514 315
```

term, stem, count, total_count

document, count, positions

dumpindex

- Extracting a document

```
% dumpindex ap89 dt 5
<DOCNO> AP890101-0005 </DOCNO>
<FILEID>AP-NR-01-01-89 0113EST</FILEID>
…
<TEXT>
   The Associated Press reported
  erroneously on Dec. 29 that Sen. James
  Sasser, D-Tenn., wrote a letter to the
  chairman of the Federal Home Loan Back
  Board, M. Danny Wall…
</TEXT>
```

dumpindex

- Extracting a list of expression matches

```
% dumpindex ap89 e "#1(my dog)"
#1(my dog) #1(my dog) 0 0
8270 1 505 507
8270 1 709 711
16291 1 789 791
17596 1 672 674
35425 1 432 434
46265 1 777 779
51954 1 664 666
81574 1 532 534
```

document, weight, begin, end

Overview (part 2)

- Indexing your own data
- Using ParsedDocument
- Indexing document fields
- Using dumpindex
- Using the Indri and classic Lemur APIs
- Getting help

Introducing the API

- Lemur "Classic" API
  - Many objects, highly customizable
  - May want to use this when you want to change how the system works
  - Support for clustering, distributed IR, summarization
- Indri API
  - Two main objects
  - Best for integrating search into larger applications
  - Supports Indri query language, XML retrieval, "live" incremental indexing, and parallel retrieval

Indri: IndexEnvironment

- Most of the time, you will index documents with IndriBuildIndex
- Using this class is necessary if:
  - you build your own parser, or
  - you want to add documents to an index while queries are running
- Can be used from C++ or Java

Indri: IndexEnvironment

- Most important methods:
  - addFile: adds a file of text to the index
  - addString: adds a document (in a text string) to the index
  - addParsedDocument: adds a ParsedDocument structure to the index
  - setIndexedFields: tells the indexer which fields to store in the index

Indri: QueryEnvironment

- The core of the Indri API
- Includes methods for:
  - Opening indexes and connecting to query servers
  - Running queries
  - Collecting collection statistics
  - Retrieving document text
- Can be used from C++, Java, PHP or C#

QueryEnvrionment: Opening

- Opening methods:
  - addIndex: opens an index from the local disk
  - addServer: opens a connection to an Indri daemon (IndriDaemon or indrid)
- Indri treats all open indexes as a single collection
- Query results will be identical to those you'd get by storing all documents in a single index

QueryEnvironment: Running

- Running queries:
  - runQuery: runs an Indri query, returns a ranked list of results (can add a document set in order to restrict evaluation to a few documents)
  - runAnnotatedQuery: returns a ranked list of results and a list of all document locations where the query matched something

QueryEnvironment: Retrieving

- Retrieving document text:
  - documents: returns the full text of a set of documents
  - documentMetadata: returns portions of the document (e.g. just document titles)
  - documentsFromMetadata: returns documents that contain a certain bit of metadata (e.g. a URL)
  - expressionList: an inverted list for a particular Indri query language expression

Lemur "Classic" API

- Primarily useful for retrieval operations
- Most indexing work in the toolkit has moved to the Indri API
- Indri indexes can be used with Lemur "Classic" retrieval applications
- Extensive documentation and tutorials on the website (more are coming)

Lemur Index Browsing

- The Lemur API gives access to the index data (e.g. inverted lists, collection statistics)
- IndexManager::openIndex
  - Returns a pointer to an index object
  - Detects what kind of index you wish to open, and returns the appropriate kind of index class
- docInfoList (inverted list), termInfoList (document vector), termCount, documentCount

Lemur Index Browsing

Index::term

term(char* s) : convert term string to a number

term(int id) : convert term number to a string

Index::document

document(char* s) : convert doc string to a number

document(int id) : convert doc number to a string

Lemur Index Browsing

Index::termCount

termCount() : Total number of terms indexed

termCount(int id) : Total number of occurrences of term number *id*.

Index::documentCount

docCount() : Number of documents indexed

docCount(int id) : Number of documents that contain term number *id*.

Lemur Index Browsing

Index::docLength(int docID)

The length, in number of terms, of document number *docID*.

Index::docLengthAvg

Average indexed document length

Index::termCountUnique

Size of the index vocabulary

Lemur Index Browsing

Index::docLength(int docID)

The length, in number of terms, of document number *docID*.

Index::docLengthAvg

Average indexed document length

Index::termCountUnique

Size of the index vocabulary

Lemur: DocInfoList

Index::docInfoList(int termID)

Returns an iterator to the inverted list for *termID*. The list contains all documents that contain *termID,* including the positions where *termID* occurs.

Lemur: TermInfoList

Index::termInfoList(int docID)

Returns an iterator to the direct list for *docID*.

The list contains term numbers for every term contained in document *docID*, and the number of times each word occurs.

(use termInfoListSeq to get word positions)

Lemur Retrieval

Class Name	Description
TFIDFRetMethod	BM25
SimpleKLRetMethod	KL-Divergence
InQueryRetMethod	Simplified InQuery
CosSimRetMethod	Cosine
CORIRetMethod	CORI
OkapiRetMethod	Okapi
IndriRetMethod	Indri (wraps QueryEnvironment)

Lemur Retrieval

- RetMethodManager::runQuery
  - query: text of the query
  - index: pointer to a Lemur index
  - modeltype: “cos”, “kl”, “indri”, etc.
  - stopfile: filename of your stopword list
  - stemtype: stemmer
  - datadir: not currently used
  - func: only used for Arabic stemmer

Lemur: Other tasks

- Clustering: ClusterDB
- Distributed IR: DistMergeMethod
- Language models: UnigramLM, DirichletUnigramLM, etc.

Getting Help

- http://www.lemurproject.org
  - Central website, tutorials, documentation, news
- http://www.lemurproject.org/phorum
  - Discussion board, developers read and respond to questions
- http://ciir.cs.umass.edu/~strohman/indri
  - My own page of Indri tips
- README file in the code distribution

Concluding: In Review

- Paul
  - About the toolkit
  - About Language Modeling, IR methods
  - Indexing a TREC collection
  - Running TREC queries
  - Interpreting query results

Concluding: In Review

- Trevor
  - Indexing your own data
  - Using ParsedDocument
  - Indexing document fields
  - Using dumpindex
  - Using the Indri and classic Lemur APIs
  - Getting help

Lemur

Questions

Ask us questions!

What is the best way to do *x*?

When do we get coffee?

How do I get started with my particular task?

Does the toolkit have the *x* feature?

How can I modify the toolkit to do *x*?

CIIR

lti