Lemur
(Draft last modified 12/08/2010)

1. **Module name:** Lemur

2. **Scope**
   This module addresses the basic concepts of the Lemur platform that is specifically designed to facilitate research in Language Modeling and information retrieval.

3. **Learning Objectives**
   By the end of work on this module, students will be able to:
   a. Describe the basic workings of the Lemur Toolkit
   b. Use effectively the Indri search engine, a part of the Lemur Toolkit
   c. Do Language Modeling with the Lemur Toolkit
   d. Cluster documents using the Lemur Toolkit

4. **5S Characteristics of the module**
   a. Space: The concept of space for this module includes physical. The data collections and the Lemur application are stored in a physical space on servers (IBM cloud computers). Lemur creates an index which uses a vector space representation.

   b. Stream: Text documents and queries are inputs that lead to result sets as well as documents as the output of an IR system. These all are sequences of characters.

   c. Structure: Data collections are stored in a TREC or TRECWeb format structure.

   d. Scenario: These include where users/administrators and/or the system submit queries, cluster documents, do language modeling, and retrieve information.

   e. Society: End users of the system such as researchers, journalists, librarians, or developers who would like to use Lemur to search relevant data or to cluster documents.

5. **Level of effort required**
   a. Class time: 2.5 hours

   b. Student time outside class:
   Preparation /reading: 2 hours

6. **Relationships with other modules:**
   Related to module 7-a, which is indexing and searching

7. **Prerequisite knowledge required:**
   Basic Linux commands

8. **Introductory remedial instruction**: None

9. **Body of Knowledge**

   a. **Lemur Toolkit Description**

   - **Lemur Toolkit Introduction**

   The Lemur Toolkit has been created for the purpose of Language Modeling and Information Retrieval (IR). This toolkit is used for developing search engines, text analysis tools, browser toolbars, and data resources in the area of IR. Lemur supports the following features:

   - Indexing:
     - English, Chinese, and Arabic text
     - Word stemming
     - Stop words
     - Tokenization
     - Passage and incremental indexing
   - Retrieval:
     - Ad hoc retrieval (TF-IDF, Okapi, and InQuery)
     - Passage and cross-lingual retrieval
     - Language modeling
       - Query model updating
       - Two stage smoothing
     - Relevance feedback
     - Structured query language
     - Wildcard term matching
   - Distributed IR:
     - Query-based sampling
     - Database based ranking (CORI)
     - Results merging
   - Document clustering
   - Summarization
   - Simple text processing

   The programming language used to create Lemur is C++ and it comes along with the source files and a make file. The provided source code can be modified for the purpose of developing new libraries. It is compatible with various operating systems which include UNIX (Linux and Solaris) and Windows XP.

   The latest available version of the Lemur Toolkit is version 4.12.

- **Indri Search Engine**

  The Indri search engine is a subset of the Lemur Toolkit which is available for free for everyone. The query language that is used in Indri allows researchers to index data or structure documents without writing any code. Indri offers flexibility in terms of adaptation to various current applications. It also can be distributed across a cluster of nodes for high performance. The Indri API supports various programming and scripting languages like C++, Java, C#, and PHP. The latest available version of Indri is 2.12.

b. Language Modeling and Information Retrieval Background

Language Modeling is a probabilistic mechanism to generate a text. Language Modeling has been a subject of interest for researchers for the past few decades, especially statistical language modeling. The information retrieval community has proved that the language modeling approach is effective and provides an attractive framework for building IR systems.

- **Language Models for Text Retrieval**

  The very basic techniques used in language modeling have been around for a very long time. For example in the Naïve Bayes text classification method, the unigram language model is estimated for each class, then combined with the class prior for classification. One problem with this approach, however, is that it assumes independence of words, which is unrealistic, and which language modeling overcomes.

  The method of using document language models to assign likelihood scores to queries has come to be known as the language modeling approach.

  Language models can be represented using finite automata, where each word can be represented as a state and the transitions between the different states are depicted. In language modeling each word is dependent on the words which occur before it. Assume a string contains four terms t1, t2, t3, and t4, occurring as "t1t2t3t4". Its probability is determined using the chain rule, as shown using the equation below:

$$P(t_1t_2t_3t_4) = P(t_1)P(t_2|t_1)P(t_3|t_1t_2)P(t_4|t_1t_2t_3)$$

Figure 1: Chain rule equation

(Manning C. D., et al. (2009). Page 240. Chapter 12. Language models for information retrieval. In Introduction to Information Retrieval. Cambridge University Press.)

The query likelihood language model constructs a language model for every document and calculates the probability of the document being generated by the query, i.e., P(q|d). The equation depicting the query likelihood language model is:

$$P(d|q) \propto P(d) \prod_{t \in q}((1-\lambda)P(t|M_c) + \lambda P(t|M_d))$$

Figure 2: Query Likelihood Language Model Equation

(Manning C. D., et al. (2009). Page 245. Chapter 12. Language models for information retrieval. In Introduction to Information Retrieval. Cambridge University Press.)

where M_d is the language model for the document, M_c is the language model for the collection, d is a document, t is a term, q is a query and λ is a constant such that $0< \lambda<1$.

- **KL Divergence**

KL Divergence is a non-symmetric ranking function which measures how bad the probability distribution M_q is at modeling M_d. It captures the term occurrence distribution much better than the multinomial approach.

$$\sum_{w:c(w;d)>0,p(w|\widehat{\theta}_Q)>0} p(w|\widehat{\theta}_Q) \log \frac{p_s(w \mid \mathbf{d})}{\alpha_d p(w \mid C)} + \log \alpha_d$$

Figure 3: Scoring function used by KL Divergence

(ChengXiang Zhai, (2007). Notes on the KL-divergence retrieval formula and Dirichlet prior smoothing.)

Where d is document, w is word, p(w|θQ) is a query model, $|\widehat{\theta}_Q$ is estimated query, p(w|C) is the collection language model, ps(w|d) is the smoothed probability of a word seen in the document, αd is a coefficient controlling.

- **Dirichlet smoothing**

Smoothing methods are defined as avoiding zeros for the query words in the document. Dirichlet smoothing is one of the Bayesian justified smoothing methods; it uses a general smoothing scheme.

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

$$\alpha_d = \frac{\mu}{\mu + |d|}$$

Figure 4: General Smoothing Method

(ChengXiang Zhai, (2007). Notes on the KL-divergence retrieval formula and Dirichlet prior smoothing.)

Where ps(w|d) is the smoothed probability of a word seen in the document, w is word, d is document, C is collection , αd is a coefficient controlling

The Dirichlet prior determines the amount of smoothing based on a document's length and performs better than fixed linear interpolation.

- **Jelinek-Mercer (JM) Smoothing**

JM smoothing is a traditional linear interpolation method. It focuses on mixing unigram maximum likelihood estimators with unigram background models. JM smoothing methods produces a movement of probability mass from seen terms in the documents.

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

Figure 5: Retrieval Score of JM Smoothing Divergence

(Advances in Multilingual and Multimodal Information Retrieval: 8th Workshop)

Where it uses coefficient λ to control the influence, w is the word, d is the given document and C is collection of documents.

- **OKAPI**

OKAPI ranks documents based on their estimated relevance regarding the given query.

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})},$$

Figure 6: Scoring function used by OKAPI

Where $f(qi,D)$ is qi's term frequency in the document D, $|D|$ is the length of the document D in words, and *avgdl* is the average document length in the text collection from which documents are drawn. k_1 and b are free parameters, usually chosen as $k_1 = 2.0$ and $b = 0.75$. IDF(qi) is the IDF (inverse document frequency) weight of the query term qi.

- **Relevance Feedback**

Relevance feedback (RF) involves the user in the retrieval process so as to improve the final result set. In particular, the user gives feedback on the

relevance of documents in an initial set of results. There are different methods such as probabilistic relevance feedback and the Rocchio algorithm relevance feedback.

- **Wildcard queries**
  A wildcard query is a term that contains "*". It is used when the user is uncertain of a query term or the user seeks documents containing variants of a particular term.

- **Galago**

Every search engine supports some kind of query language, but Galago takes the query language concept a bit further than most. In Galago, the query language defines the ranking function. In other systems you might consider changing the ranking code to get a desired result, but in Galago you usually just change the query.

- **Lemur Query Toolbar**

The Lemur Query Log Toolbar is a FireFox Add-On and alternatively, a plugin for IE, that captures user browsing behavior specifically when loading web pages and generating search engine requests. A set of configuration options enable researchers and users to specify toolbar behavior.

- **XML Retrieval**

XML Retrieval, or XML Information Retrieval, supports documents structured with XML (eXtensible Markup Language). As such it is used for estimating the relevance of XML documents.

- **Cross-language Information Retrieval**

Cross-language information retrieval (CLIR) is a subfield of information retrieval dealing with retrieving information written in a language different from the language of the user's query. For example, a user may pose their query in English but retrieve relevant documents written in French.

- **Pseudo Relevance Feedback**

Pseudo relevance feedback, also known as blind relevance feedback, provides a method for automatic local analysis. It automates the manual part of relevance feedback, so that the user gets improved retrieval performance without an extended interaction. The method is to do normal retrieval to find an initial set of most relevant documents, add their terms to the query, and then repeat this process and finally return what hopefully are the most relevant documents.

- **Krovetz Stemmer**

The Krovetz Stemmer was developed by Bob Krovetz, at the University of Massachusetts, in 1993. It is quite a 'light' stemmer, as it makes use of inflectional linguistic morphology.

- **Porter Stemmer**

The Porter stemming algorithm (or 'Porter stemmer') is a process for removing the commoner morphological and inflexional endings from words in English. Its main use is as part of a term normalization process that is usually done when setting up Information Retrieval systems.

- **Using the Lemur Toolkit:**

A. **Introduction**

i. The Lemur Toolkit has been installed, compiled, and is ready to use on the cloud instance. Apart from the basic utilities of Lemur, the clustering plugin also has been installed. Lemur has a wide variety of plugins one can install. You can look up http://sourceforge.net/apps/trac/lemur/wiki/Compiling%20and%20Installing for details about the other available plugins.

B. **Indexing**

The first step before you start searching is to index all the data you have. Explained below are the data formats supported for your data and the different types of parameters you can set before you begin indexing.

i. **Data Format**

The Lemur Toolkit accepts data in two formats -TRECTEXT and TRECWEB format. Following are examples, one for each of these formats.

TRECTEXT Format

<DOC>
<DOCNO>1</DOCNO>
<TEXT>
document content
</TEXT>
</DOC>

TRECWEB Format

<DOC>
<DOCNO>...</DOCNO>
<DOCHDR>
... e.g. URL
and other metadata information

</DOCHDR>
... HTML content
</DOC>

ii. **Parameters**

There are a number of parameters you can set before you begin indexing, like the memory, the type of stemmer to be used, the location of the corpus on your file system, etc. These are depicted in the example below:

<parameters>
<memory>200m</memory>
<index>/path/to/outputIndex-v4.11</index>

<metadata>
<forward>docno</forward>
<backward>docno</backward>
</metadata>

<stemmer>
<name>krovetz</name>
</stemmer>

<corpus>
<path>/path/to/collection1/</path>
<class>trectext</class>
</corpus>
<corpus>
<path>/path/to/collection2/</path>
<class>trecweb</class>
</corpus>

<field><name>title</name></field>
<field><name>date</name><numeric>true</numeric><parserName
>DateFieldAnnotator</parserName></field>

</parameters>

The purposes of the parameters are as follows:
Memory parameter - defines a rough limit for RAM memory consumption by the indexer.

Metadata - the forward tag helps define fast forward retrieval and the backward tag helps define fast backward retrieval.

Stemmer - The Lemur toolkit provides two types of stemmers: Krovtz and Porter.

Field - The columns mentioned in the field tags can be searched using Indri.

Corpus - Indicates the location of the corpus in your filesystem.

iii. **Indexing Command**

After the data is in its correct format and the parameters have been set, indexing can be done using the following command:

lemur-4.11/app/obj/IndriBuildIndex parameter_file

C. **Retrieval**

Similar to setting parameters for indexing, one must set parameters for batches of queries too before retrieval.

i. **Query parameter file**

Given below is an example of how the query can be specified between the text tags.

<parameters>
<query>
<type>indri</type>
<number>751</number>
<text>
#combine(popular scrabble players)
</text>
</query>
<query>
<type>indri</type>
<number>752</number>
<text>
#combine(dam removal environmental impact)
</text>
</query>
</parameters>

Where <number> indicates QueryID, #combine indicates OR operation on the specified query.

ii. **Retrieval Command**

The command for retrieval is:

lemur-4.11/app/obj/IndriRunQuery query_parameter_file -count=1000 -index=/path/to/index -trecFormat=true > result_file

-count is used to specify the number of output files to be displayed as result of the query which is executed by the user.
-index specifies the location of the index in the filesystem
-trecFormat = true specifies that the file is in TREC format

D. **Evaluation**

The Lemur Toolkit also allows you to perform evaluation of your results using the following command:

trec_eval -q QREL_file Retrieval_Results > eval_output

where the QREL_file contains the ideal results and Retrieval_Results contains the results returned by the Lemur Toolkit.

E. **Clustering**

In the beginning of this section it was mentioned that on the cloud instance apart from the basic lemur Toolkit, the clustering plugin has also been installed. Here is a brief description of how to perform clustering using the Lemur Toolkit.

- **Lemur Clustering**
  - **Overview**
    Document clustering is the action of assembling documents in a same group. Lemur supports different types of clustering algorithms such as probabilistic latent semantic analysis (PLSA), agglomerative hierarchical, K-means, and bisecting K-means. With the exception of Probabilistic Latent Semantic Analysis (PLSA), all use cosine similarity in the vector space model as their metric.

    The LEMUR clustering supports two principal APIs, the Cluster API, that defines the clusters themselves, and the ClusterDB API, that defines how clusters are persistently stored.
  - **Applications**
    - **Cluster**
      Performs the basic online clustering task. The parameters accepted by Cluster are:
        - Index -- the index to use. Default is none.

- clusterIndex -- the name of the cluster database index to use. Default is none.
- clusterdb_type -- One of flatfile (simple cluster database) or keyfile (btree based).
- clusterType -- Type of cluster to use, either agglomerative or centroid. Centroid is agglomerative using mean which trades memory use for speed of clustering. Default is centroid.
- simType -- The similarity metric to use. Default is cosine similarity (COS), which is the only implemented method.
- docMode -- The scoring method to use for the agglomerative cluster type. The default is max (maximum). The choices are:
- max -- Maximum score over documents in a cluster.
- mean -- Mean score over documents in a cluster. This is identical to the centroid cluster type.
- avg -- Average score over documents in a cluster.
- min -- Minimum score over documents in a cluster.
- threshold -- Minimum score for adding a document to an existing cluster. Default is 0.25.

- **Offline Cluster**
  Performs the basic offline clustering. The parameters accepted by OfflineCluster are:
  - index -- the index to use. Default is none.
  - clusterType -- Type of cluster to use, either agglomerative or centroid. Centroid is agglomerative using mean which trades memory use for speed of clustering. Default is centroid.
  - simType -- The similarity metric to use. Default is cosine similarity (COS), which is the only implemented method.
  - docMode – The scoring method to be used for the agglomerative cluster type. The default is max (maximum). The choices are:
    - max -- Maximum score over documents in a cluster.
    - mean -- Mean score over documents in a cluster. This is identical to the centroid cluster type.
    - avg -- Average score over documents in a cluster.
    - min -- Minimum score over documents in a cluster.
  - numParts -- Number of partitions to split into. Default is 2.
  - maxIters -- Maximum number of iterations for k-means. Default is 100.
  - bkIters -- Number of k-means iterations for bisecting k-means. Default is 5.

- **PLSA**

Performs Probabilistic Latent Semantic Analysis (PLSA) on a document collection to build three probability tables: P(z), P(d|z), and P(w|z).

The parameters accepted by PLSA are:

- index -- the index to use. Default is none.
- numCats -- the number of latent variables (categories) to use. Default is 20.
- beta -- The value of beta for Tempered EM (TEM). Default is 1.
- betaMin -- The minimum value for beta; TEM iterations stop when beta falls below this value. Default is 0.6.
- eta -- Multiplier to scale beta before beginning a new set of TEM iterations. Must be less than 1. Default is 0.92.
- annealcue -- Minimum allowed difference between likelihood in consecutive iterations. If the difference is less than this, beta is updated. Default is 0.
- numIters -- Maximum number of iterations to perform. Default is 100.
- numRestarts -- Number of times to recompute with different random seeds. Default is 1.
- testPercentage -- Percentage of events (d,w) to hold out for validation.
- doTrain -- whether to construct the probability tables or read them in. Default is true.

- **Clustering API**
  - **Cluster**
    Provides an abstraction over a collection of clusters.
  - **ClusterDB**
    Provides for interactions with persistent collections of cluster objects.
  - **Similarity Method**
    SimilarityMethod is an abstraction over comparing two ClusterRep (vector space representation) cluster objects. To add a new SimilarityMethod, one needs to do the following:

    1. In ClusterParam.hpp add a symbol for the new method in the simTypes enum.
    2. In ClusterParam.hpp add an else if to test the simTypeString for equality with the new method type parameter.
    3. In SimFactory.hpp add an include statement for the new method's header file.
    4. In SimFactory.hpp add a case for the new method symbol to makeSim that makes an instance of the new method.

5. Recompile your Lemur library. (http://www.cs.cmu.edu/~lemur/3.1/cluster.html#application)

- **Instructions:**

a. **Indexing**

Index the data to be clustered as described in the previous sections.

b. **Clustering Parameters**

The parameters for clustering must be specified as follows:

<parameters>
<index>/path/to/your/index/file</index>
<clusterType>centroid</clusterType>
<docMode>max</docMode>
</parameters>

The cluster type tag specifies the type of clustering to be performed.

c. **Command for clustering**

The command for clustering using the Lemur Toolkit is:

Cluster centroid_cluster.OfflineCluster.xml > centroid_cluster_last.log

where the centroid_cluster.OfflineCluster.xml file contains the parameters described in the previous section and centroid_cluster_last.log is the file to which the output is written.

- **Relationship to Chapters from the textbook**

Lemur uses some of the concepts described in the IR textbook Chapters 6, 7 and 12 like:

- Word stemming
- Stop words
- Tokenization
- Ad hoc retrieval (TFIDF, Okapi, and InQuery)
- Language modeling
- Relevance feedback
- Structured query language
- Wildcard term matching
- Query-based sampling
- Document clustering

- Simple Text processing

10. Resources

Required reading for students

- Manning C. D., et al. (2009). *Chapter 12*. Language models for information retrieval. In *Introduction to Information Retrieval.* Cambridge University Press.

- Manning C. D., et al. (2009). *Chapter 9*. Relevance feedback & query expansion. In *Introduction to Information Retrieval.* Cambridge University Press.

- Manning C. D., et al. (2009). *Chapter 11*. Probabilistic information retrieval. In Introduction to Information Retrieval. Cambridge University Press.
- Manning C. D., et al. (2009). *Chapter 16*. Flat clustering. In *Introduction to Information Retrieval.* Cambridge University Press.

- Manning C. D., et al. (2009). *Chapter 17*. Hierarchical clustering. In *Introduction to Information Retrieval.* Cambridge University Press.

Recommended reading for students

- Lemur web page http://www.lemurproject.org/
- Lemur wiki page http://sourceforge.net/apps/trac/lemur/wiki
- Lemur Toolkit Documentation http://www.lemurproject.org/doxygen/lemur/html/index.html

11. Concept map

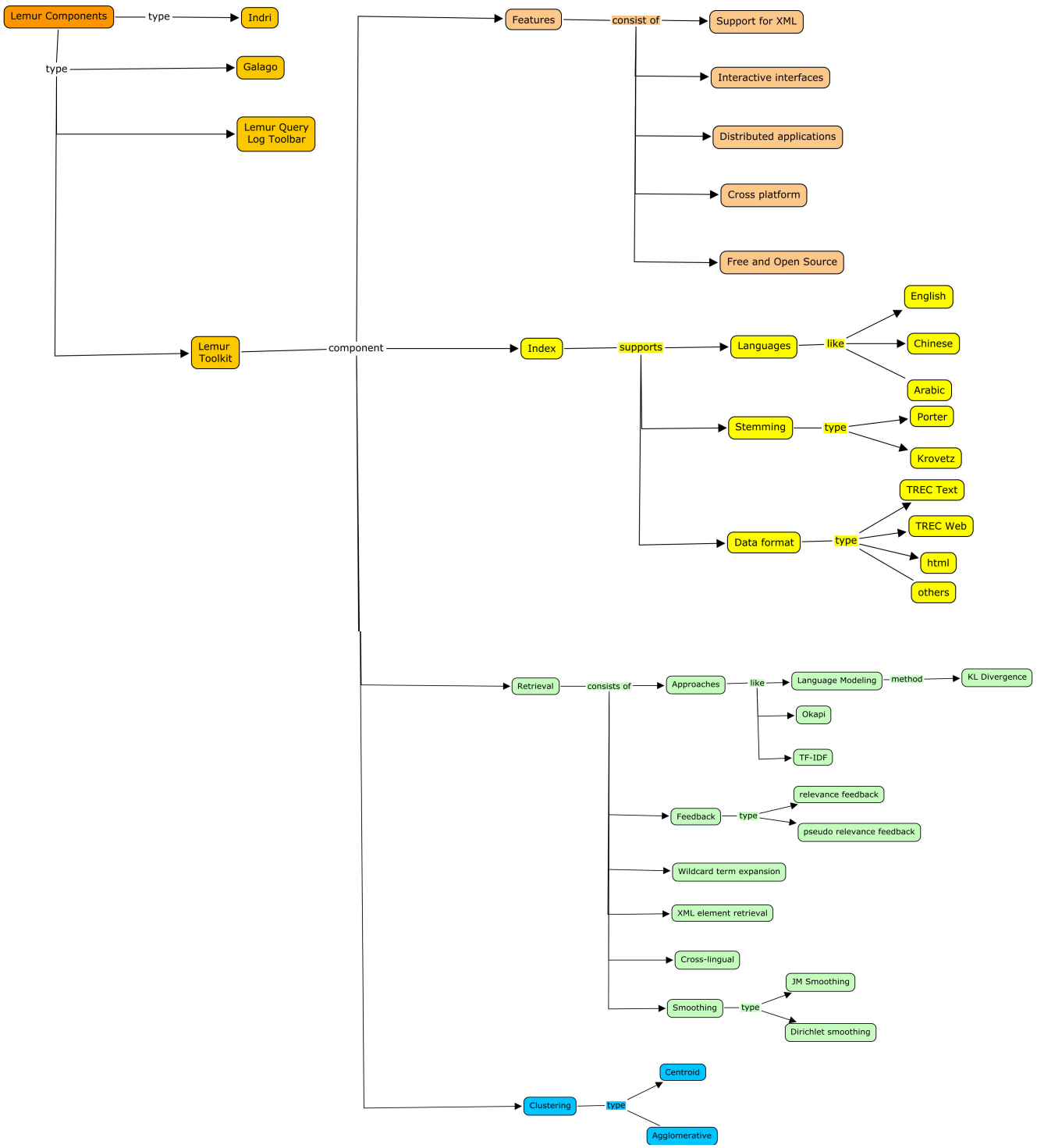

Figure 5: Lemur concept map

12. Exercises/Learning activities

TRECEVAL is a program to evaluate TREC results using the standard, NIST evaluation procedures. The dataset used for solving the exercise questions given below is a part of the TRECEVAL dataset from CD-1. It contains a set of articles

from various technical journals.

Mr. David is a data analyst, whose primary job is to keep track of the content being published by the technical magazines and to report any interesting observations he finds. Mr. Davis is currently using the Lemur Toolkit for his work.

1. Mr. David would like to study the Microsoft vs. Apple trend and see which of these two companies has more coverage in the magazines. But Lemur offers multiple ways to analyze the data present:-

i. Simple TF-IDF
ii. TF-IDF with feedback retrieval
iii. Simple Okapi
iv. Language Modeling with various smoothing methods like JM Smoothing and Dirichlet Smoothing

Your duty is to run each of these methods in Lemur, which will help him in his study.
(**HINT:**
Step 1: Index the data provided in the collections folder.
Step 2: Run each of the methods from i. to iv., with the query term as 'Microsoft'.
Step 3: Repeat Step 2, with query term as 'Apple'.
Step 4: Compare results in Steps 2 and 3.)

2. Mr. David wants to study all the articles published by PC Magazine. Use the Language Modeling feature available in Lemur to help him retrieve all the articles published by PC Magazine.

3. Matzkin is considered one of the most popular technical writers of the current era. Help Mr. David to locate the articles written by Matzkin.

4. Your final task if to help Mr. David to cluster all the articles available from the TREC EVAL data set provided to you. Use the centroid clustering method.

13. Evaluation of learning achievement

At the end of this module, students must understand the working of the Lemur Toolkit. They must be able to formulate queries, execute searches, cluster documents, and evaluate the results.

14. Glossary

- *Collection*: A group of items, often documents.

- *Feature:* Information extracted from an object and used during query processing.

- *Index:* A data structure built for the documents to speed up searching.

- *Information Retrieval:* Part of Computer Science that studies retrieval of information (not data) from a collection of written documents.

- *Metadata:* Attributes of a data or a document.

- *Query:* The expression of the user information need in the input language provided by the information system.

- *Relevance feedback:* An interactive process of obtaining information from the user about the relevance and the non-relevance of retrieved documents.

- *Language Model:* A statistical language model assigns a probability to a sequence of m words by means of a probability distribution.

- *Cluster:* a grouping of a number of similar things

15. Contributors

a. Iccha Sethi
b. Serdar Aslan
c. Dr. Edward Fox

Information Storage and Retrieval CS 5604
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061 USA