# Implement n-gram language model based retrieval and ranking within Galago. Length of n-grams must vary from 1 to N=5. Compare over noisy text.

Milestone: If you are using Galago for your project, **describe the architecture of the Galago system**, its **query language, query processing technique**, and the **retrieval model** used.

Galago is an open-source toolkit for development of information retrieval engine. It is based on query language from Indri and Limur developments. This query language framework provides several advantages such as flexibility, multiple document representations, efficient implementation and formal grounding. Information on Galago and related information is obtained from Galago docs, Galago source wiki, Limur docs and developed and research blogs.

### Query language:

Query languages enable a syntactical representation of a query towards a database or a retrieval system. Galago’s query language is based on Indri and Inquery query languages of the Lemur project. However, unlike other search engines, Galago’s query language extensively embedded into retrieval model allowing for greater flexibility. For example, scores of retrieval of the query ‘hello world’ are combined as #combine( #text:dog() #text:cat() )

The Galago query language is based on sequential operators, in its simple form can capture several ranking operations. Unlike other existing systems, the query language itself directly provides handles to request information such as inverted index count of a term etc. Such information that is usually available in the indexing module. The query is presented as a graphical model, with nodes containing various tags and meta-parameters such as smoothing parameters etc.

Query operations of Galago query language provide information of the following categories:

* Index operators:  #counts and #extents
  + They are processed directly from the indexes.
* Proximity operators:  #inside, #ordered and #unordered
  + Provide proximity information
* Scoring operators:  #feature:dirichlet
  + The runQuery method requires scores provided by these operators.
* Score combination operators: #combine and #weight
  + Combine scores from different scoring operations.
* Other operators
  + It is possible to implement new operators by implementing StructuredIterator interface.
* We now briefly discuss the various operators of Galago, focusing primarily on those which are deemed relevant to this project.
* **#extents**
* This operator provides the context/fields of the term as per the posting list.
* **#count**
* This operator provides the document count of a given term obtained from the posting list
* **#prior**
* Provides the document prior in the posting list. In the case that the prior is zero, minScore is assigned. (minScore = ln( 0.0000000001 ))
* **#lengths**
* Provides document lengths
* **#feature**
* Provides smoothed log probabilities of a document

Every document is parsed while maintaining the arbitrary tags that are present in it. Hence, any term is accompanied with a **context** parameter, which retains the zone of the term. Further, since a single term can contain multiple contexts, these are stored in a nested fashion.

### Traversal:

An important difference between Indri and Galago query languages is traversals. This feature allows for query transformation, often adding additional information and annotation that can be derived from the retrieval model. The queries are converted upon traversal, to the extent where it is directly executed over the Galago indexes. For example,

hello world

#combine(hello world)

#combine(#feature:dirichlet(#counts:part=postings:hello())#feature:dirichlet(#counts:part=postings:world()))

A simple query ‘hello world’ is converted using the Galago query operators into the final notation which is executed.

The traversal interface provided in Galago enables query transformation. Each query is represented as a graph on nodes; the different attributes can be changed to transform the query.

Galago already uses five traversals on every query executed:

* AddCombineTraversal
* WeightConversionTraversal
* IndriWindowCompatibilityTraversal
* TextFieldRewriteTraversal
* ImplicitFeatureCastTraversal

### Retrieval Model:

The indexing strategy of Galago is designed so most modification can be done at query level itself. Since most information from indexing process is available at query time. Most retrieval strategies can be implemented by appropriate query structure. It is generally discouraged to modify the core indexing code, which is a bad strategy due to immense sophistication of the query processing. The processing is optimized using Tupleflow, a variant of MapReduce developed by Google to improve computation performance.

However, the posting list is maintained in the following structure:

**Posting**: Each term of a document is indexed in a triple containing word count, docID and position.

**Stemming**: All terms are first stemmed using Porter stemmer. However, the unstemmed list is also stored in the index; hence both options are available during query processing.

**Extents**: During the index process, the arbitrary tags present in the document are also stored along with docID, start and end position. This feature is useful in zone based weighting strategies etc.

References:

<http://sourceforge.net/apps/trac/lemur/wiki/Indri%20Retrieval%20Model>

# Bibliography

*Galago Query language*. (n.d.). Retrieved 2012, from Galagosearch: http://www.galagosearch.org/retrieval.html