Biobank Search on Db /Ontocat based in Lucene Indexing featuring Query Expansion using Ontologies

A. How to set up , configure & run

(Run molgenis project - molgenis4phenotype or in your own)

1. Download & install molgenis (http://www.molgenis.org/wiki/MolgenisOnWindows,)
2. Download latest version of molgenis4phenotype from http://www.molgenis.org/svn/molgenis_projects/molgenis4phenotype
3. Set Build path (add Lucene/Ontocat/ols/owlapi/wsdl/ jars)
   Make sure the following jar exist in your WEB-INF/lib directory:
   - lucene-core-3.0.2.jar
   - lucene-demos-3.0.2.jar
   - lucene-highlighter-3.0.1.jar
   - lucene-memory-3.0.2.jar
   - ols-client.jar
   - ontoCAT_v0.9.4-SNAPSHOT.jar
   - opencsv-1.8.jar
   - owlapi-bin.jar
   - owlapi-src.jar
   - wsdl4j-1.6.2.jar
   - xpp3_min-1.1.4c.jar

   Next select project properties → java build path→ Libraries→ Add Jars→ select the jars mentioned above.

   The libraries may also exist in the preconfigured Web App Libraries , if so , the current step (3) can be skipped .

4. Select two of preconfigured existing sets of files, AnimalDB , lifelines, or create your own. Currently adjusted for animalDB. Respective configuration includes (consider the corresponding files if you want to create your own) :
   a. animaldb.molgenis.properties :

      db_user = molgenis

      db_password = molgenis

   b. Create db : animaldb_pheno or yours as follows

      mysql> create database animaldb_pheno;

      mysql> grant all privileges on animaldb_pheno. to molgenis@localhost identified by 'molgenis'; flush privileges; **

c. *Plugin files already exist in molgenis4phenotype. For your own project:*

Add in molgenis_ui.xml the menu:

```
<!-- Lucene biobank search plugin  -->
 <menu name="submenu" position="left" label="Indexing...">
    <plugin name="DBIndex" label="DB Index and Search" type="plugins.LuceneIndex.DBIndexPlugin" />
    <plugin name="GenericWizard" type="plugin.genericwizard.GenericWizard" label="Excel upload"/>
    <plugin name="OntoCatIndexPlugin2" label="Index OntoCAT"  type="plugins.LuceneIndex.OntoCatIndexPlugin2" />
</menu>
```

d. AnimalDBGenerate.java : run
e. AnimalDBUpdateDatabase.java: run
f. Files are now created. Replace created files with plugin's files. Also add *LuceneIndexConfiguration.properties* in directory where all .properties files leave (/molgenis4phenotypeWorkspace/molgenis4phenotype)
g. Adjust in *LuceneIndexConfiguration.properties configuration* file: The number of DB fields in which the search "makes sense " (mainly description fields). Also fill in the names of the fields. Also select if you want to use ontologies in query expansion [1] by selecting *useOntologiesInQueryExpansion = "true"*

Building & searching an index on database contents

h. Fill in data (in case of Animal Db project used). Run in server and select from the menu "System Tasks→ Fill in Database ". You can also load latest animal db files from the same page.
i. Db is now ready to be search. From the main menu, select "Indexing → DB Index & Search".
The index is created in folder predefined in variable LUCENE_INDEX_DIRECTORY. If it does not exist, the directory is being created. If the directory has contents, the index is NOT created. (Be sure to remove the contents for the new index).

After the creation of the index is complete you can search by entering a term (or a sentence) in the search box.

[1] In case you use ontologies for query expansion you need to follow the instruction in section B .

If useOntologiesInQueryExpansion is selected (true), the query is being expanded by terms retrieved from the downloaded ontologies that leave in LUCENE_INDEX_DIRECTORY

Building & searching an index on Ontocat contents

http://sourceforge.net/projects/ontocat/

In order to build an index based on Ontocat contents some adjustments must be made:

1. Set the VM arguments for OntoCatIndexPlugin2.java to "–Xms1024M –Xmx1024M"

(Select project→ Run As → Run configurations→ Arguments → Add in VM arguments -Xms1024M -Xmx1024M)

2. Enter a desired term in order to retrieve from online ontology resources and press "Build Ontocat Index".

3. The index is created in folder predefined in variable LUCENE_ONTOINDEX_DIRECTORY. If it does not exist, the directory is being created. If the directory has contents, the index is NOT created. (Be sure to remove the contents for the new index).

4. After the creation of the index (this may take a while depending on the response of the ontology resources that Ontocat is speaking to – EBI Ontology service) is complete you can search by entering a term (or a sentence) in the search box.

B. How to run query expansion enabled search (using ontologies)

1) Download the ontologies from http://bioportal.bioontology.org/

You should download

- (http://rest.bioontology.org/bioportal/ontologies/download/44307?applicationid=4ea81d74-8960-4525-810b-fa1baab576ff)

- Human Disease (http://rest.bioontology.org/bioportal/ontologies/download/44309?applicationid=4ea81d74-8960-4525-810b-fa1baab576ff)

- NCI Thesaurus (http://rest.bioontology.org/bioportal/ontologies/download/42838?applicationid=4ea81d74-8960-4525-810b-fa1baab576ff)

MeSH can be taken from biobank_search\WebContent\WEB-INF

2) Change the directory names:

- In DBIndexPlugin: LUCENE_INDEX_DIRECTORY, INDEX_CONFIGURATION

- In OntoCatIndexPlugin2: LUCENE_ONTOINDEX_DIRECTORY, ONTOLOGIES_DIRECTORY

C. Lucene scoring

Lucene scoring uses a combination of the Vector Space Model (VSM) of Information Retrieval and the Boolean model to determine how relevant a given Document is to a User's query.

In general, the idea behind the VSM is the more times a query term appears in a document relative to the number of times the term appears in all the documents in the collection, the more relevant that document is to the query. It uses the Boolean model to first narrow down the documents that need to be scored based on the use of boolean logic in the Query specification.

Lucene also adds some capabilities and refinements onto this model to support boolean and fuzzy searching, but it essentially remains a VSM based system at the heart. For some valuable references on VSM and IR in general refer to the Lucene Wiki IR references.

(see more in Appendinx B)

The score for a document given a query is the cosine of the angle formed between the query vector and the document vector. The explain() method can be used to show exactly what score calculation is for a given query and a given document. So explanation() 's results (explanation. toString()) is presented to the user.

Appendix

A. Information Retrieval with Query Expansion

General ideas:

1) Query expansion adds additional terms related to initial query terms to the query. They shouldn't be obligatory contained in a document (it would be difficult to find a document in a database, containing every term of ["exercise-induced asthma", "chronic obstructive asthma with acute exacerbation"," exercise-induced asthma (disorder)", "bronchial hypersensitivity", "chronic obstructive asthma", "chronic obstructive asthma with status asthmaticus", "bronchial hyperreactivity", "exercise induced asthma", "cough variant asthma", "intrinsic asthma", "status asthmaticus", "allergic asthma"]), that's why they should be appended by "OR" operator and can be assigned a lower weight. Thus such query expansion usually changes the document ranking and consequently the order of retrieved documents in the output, rather than significantly changes the number of documents retrieved.

2) What terms to add? Obviously, the added terms should be very close to the query term, that's why in information retrieval as expansion terms usually synonyms and children (terms, related to the query term by IS_A relationship) are added. For example, if a user enters a broad query, such as lung disease, query expansion will add documents concerning narrower terms, such as pneumonia (children node),

3) It is very important to have good ontologies at hand. Otherwise the expansion terms may turn out to be very inaccurate. This is the problem with nonscientific terminology: it's practically impossible to construct an accurate ontology, due to the vagueness of words of natural languages. Synonymy is very approximate here and it's difficult to determine where exactly in the ontology tree the term is to be put. Scientific terminology is much better in this respect, because it is much more exact. Of course there is still some inaccuracy, but query expansion can be efficient.

4) Even if query expansion itself doesn't improve the search, the query can be made more precise: if some of the terms are found in the ontologies, they are put in quotation marks, thus avoiding wrong results.

For example, if user doesn't put quotation marks in his query: cystic lung disease, then documents, containing disease will be retrieved:

(1) New diagnosis of heart disease since last study visit

(2) The score on the Unified Parkinson's Disease Rating Scale

During query expansion cystic lung disease will be found in ontologies and the query will become: asthma "cystic lung disease" OR (…expansion terms…). This

query won't find those two irrelevant documents, because of the quotation marks.

Ontologies

What ontologies to use?

It should be decided by the user in accordance with his query. He should be given the list of ontologies to choose:

I would propose to choose among the following ontologies:

- Human Phenotype Ontology

- Human disease

- NCI Thesaurus

- Medical Subject Headings

- International Classification of Diseases (http://bioportal.bioontology.org/visualize/35686)

-Synonyms are graphical variants used in special cases

- Online Mendelian Inheritance in Man (http://bioportal.bioontology.org/visualize/40398)

(The relation "manifestation of" may be useful, though too broad)

- Practically no synonyms

How to search the ontologies?

The search is performed by OntoCAT.

In this project I tried different ways of accessing the ontologies:

(1) Directly on BioPortal

(2) Downloading the ontologies on local computer

(3) Indexing them and searching in the index files

The third variant turned out to be significantly faster, so it is used in the project.

What is done?

The User first can index his database and ontologies and then search for relevant database entries.

The User enters his query in the textbox, he can choose whether to expand the query or not. Choose “Search with query expansion”, the query is expanded with synonyms and children from indexed ontologies (Human disease, Human Phenotype ontology? and NCI Thesaurus). Then Lucene performs the search in the indexed database.

B. Lucene Indexing: scoring

Fields and Documents

In Lucene, the objects we are scoring are Documents. A Document is a collection of Fields. Each Field has semantics about how it is created and stored (i.e. tokenized, untokenized, raw data, compressed, etc.) It is important to note that Lucene scoring works on Fields and then combines the results to return Documents. This is important because two Documents with the exact same content, but one having the content in two Fields and the other in one Field will return different scores for the same query due to length normalization (assumming the DefaultSimilarity on the Fields).

Score Boosting

Lucene allows influencing search results by "boosting" in more than one level:

- Document level boosting - while indexing - by calling document.setBoost() before a document is added to the index.
- Document's Field level boosting - while indexing - by calling field.setBoost() before adding a field to the document (and before adding the document to the index).
- Query level boosting - during search, by setting a boost on a query clause, calling Query.setBoost().

Indexing time boosts are preprocessed for storage efficiency and written to the directory (when writing the document) in a single byte (!) as follows: For each field of a document, all boosts of that field (i.e. all boosts under the same field name in that doc) are multiplied. The result is multiplied by the boost of the document, and also multiplied by a "field length norm" value that represents the length of that field in that doc (so shorter fields are automatically boosted up). The result is decoded as a single byte (with some precision loss of course) and stored in the directory. The similarity object in effect at indexing computes the length-norm of the field.

This composition of 1-byte representation of norms (that is, indexing time multiplication of field boosts & doc boost & field-length-norm) is nicely described in Fieldable.setBoost().

Encoding and decoding of the resulted float norm in a single byte are done by the static methods of the class Similarity: encodeNorm() and decodeNorm(). Due to loss of precision, it is not guaranteed that decode(encode(x)) = x, e.g. decode(encode(0.89)) = 0.75. At scoring (search) time, this norm is brought into the score of document as norm(t, d), as shown by the formula in Similarity.

C. Documentation

public class DBIndexPlugin

the plugin to index and search the database (with or without query expansion):

@param LUCENE_INDEX_DIRECTORY – empty directory to put index files in

public void buildIndexAllTables(Database db) –makes the index

public void SearchAllDBTablesIndex(Database db) –searches the index (in “description” field)

public void ExpandQuery(Database db) –expands the query by calling expand(OntologiesForExpansion)from OntocatQueryExpansion_lucene

public class OntocatQueryExpansion_lucene

public List<String> parseQuery(String query) –parses the query by ignoring the punctuation, splitting the query by ‘ ‘, Boolean operators, reading phrases in quotation marks as a single unit. Calls public List<String> chunk (List<String> words)

public List<String> chunk (List<String> words) – chunks the query (List<String> words) into all possible n-grams (combinations of subsequent query words) (n ranges from 1 to words.size())

public void expand(List<String> ontologiesToUse) – finds expansion terms in ontologiesToUse. For every n-gram of the chunked query searches it in ontologies, if found, adds expansion terms to initial query list

public String output(List<String> parsed) – constructs a new query of the initial query list, adding expansion terms with lower weight, using the same Boolean operators and quotes (if any) as in user query.

public class OntoCatIndexPlugin2

the plugin that indexes and searches the ontologies

@param LUCENE_ONTOINDEX_DIRECTORY - empty directory to put index files in

@param ONTOLOGIES_DIRECTORY – the directory, where the ontologies are stored

@param ontologyNamesMap – the list of ontologies and the correspondence between ontology names and file names containing them

public String SearchIndexOntocat(String query, List<String> ontologyLabels) – searches the query in the ontologies with names ontologyLabels. Returns a string “term:expansion term1; expansion term2;… expansion termN;”

public void buildIndexOntocat() - builds the ontology index. Pairs (term:expansion) are stored for each term of each ontology