Using the MetaMap Java API

Willie Rogers

March 12, 2014

Contents

1 Purpose

MetaMap maps terms occuring in text to UMLS Metathesaurus concepts. As part of this mapping process, MetaMap tokenizes text into sections, sentences, phrases, terms, and words. MetaMap maps the noun phrases of the text to the best matching UMLS concept or set of concepts that best cover each phrase. The MetaMap Java API provides java programs with programmatic access to MetaMap mapping engine.

2 MetaMap API's Underlying Architecture

MetaMap mapping engine is written primarily in SICStus Prolog; to facilitate its use by Java programs, the system uses PrologBeans to provide a loose coupling between the Java API and the mapping engine. See the SICStus Prolog PrologBeans documentation for more information (http://www.sics.se/sicstus/docs/latest4/html/sicstus.html/lib_002dprologbeans.html).

3 Pre-requisites

The full MetaMap download and installation is required to use the MetaMap Java API (see \http://metamap.nlm.nih.gov/#Downloads). Also, Java 1.6 SDK or greater is required. (It should work Java 1.5 but it has not been tested with Java 1.5)

4 Downloading, Extracting and Installing the API distribution

In the directory where you installed the Public Metamap (the directory containing the public_mm directory) extract the javaapi archive:

```
$ bzip2 -dc /home/piro/public_mm_linux_javaapi_{year}.tar.bz2 | tar xvf -
```

If you plan on modifying the sources to the prolog-based MetaMap server (mmserver) you will need to download and extract the source archive (http://metamap.nlm.nih.gov/download/public_mm_src_{year}.tar.bz2) as well:

```
$ bzip2 -dc /home/piro/public_mm_src_{year}.tar.bz2 | tar xvf -
```

You will need to re-run ./bin/install.sh from the public_mm directory to setup the files for javaapi.

```
$ ./bin/install.sh
```

5 Using the MetaMap server

5.1 Starting supporting servers

The MetaMap server (mmserver) must first be running to use the Java API. If the SKR/Medpost Tagger is not already running start it using the following command:

```
$ ./bin/skrmedpostctl start
```

If you wish to the Word Sense Disambiguation (WSD) Server (optional), start it also.

```
$ ./bin/wsdserverctl start
```

5.2 Running the MetaMap server

Then start the MetaMap server:

```
$ ./bin/mmserver{year}
/home/piro/public_mm/bin/SKRrun -L {year} \
     -w /home/piro/public_mm/lexicon /home/piro/public_mm/bin/mmserver{year}.BINARY.Linux -Z {y
Server options: [port(8888),accepted_hosts(['127.0.0.1','130.14.111.76','130.14.110.82'])]
Berkeley DB databases (normal {year} strict model) are open.
Static variants will come from table varsan in /home/piro/public_mm/DB/BDB4/DB.normal.{year}.st
Derivational Variants: Adj/noun ONLY.
Accessing lexicon /home/piro/public_mm/lexicon/data/BDB4/lexiconStatic{year}.
Variant generation mode: static.
```

6 Testing the API

Using another terminal, you can verify that api is running using the program `testapi.sh` which takes a query as an argument:

```
$ ./testapi.sh laboratory culture
options: []
terms: laboratory culture
input text:
 laboratory culture
Utterance:
 Id: 00000000.tx.1
 Utterance text: laboratory culture
 Position: (0, 19)
Phrase:
 text: laboratory culture
 Minimal Commitment Parse: [head([lexmatch([laboratory culture]),inputmatch([laboratory,culture
Candidates:
 Candidate:
  Score: -1000
```

```
 Concept Id: C0430400
 Concept Name: Laboratory culture
 Preferred Name: Laboratory culture
 Matched Words: [laboratory, culture]
 Semantic Types: [lbpr]
 MatchMap: [[[1, 2], [1, 2], 0]]
 MatchMap alt. repr.: [[[phrase start: 1, phrase end: 2], [concept start: 1, concept end: 2],
 is Head?: true
 is Overmatch?: false
 Sources: [MTH, LNC, MDR, NCI, RCD, MEDCIN, CCPSS, SNOMEDCT]
 Positional Info: [(0, 18)]
Candidate:
 Score: -861
 Concept Id: C0010453
 Concept Name: Culture
 Preferred Name: Anthropological Culture
 Matched Words: [culture]
 Semantic Types: [idcn]
 MatchMap: [[[2, 2], [1, 1], 0]]
 MatchMap alt. repr.: [[[phrase start: 2, phrase end: 2], [concept start: 2, concept end: 2],
 is Head?: true
 is Overmatch?: false
 Sources: [MTH, PSY, ICNP, LCH, MSH, NCI, CSP]
 Positional Info: [(11, 7)]
Candidate:
 Score: -861
 Concept Id: C0022877
 Concept Name: Laboratory
 Preferred Name: Laboratory
 Matched Words: [laboratory]
 Semantic Types: [mnob, orgt]
 MatchMap: [[[1, 1], [1, 1], 0]]
 MatchMap alt. repr.: [[[phrase start: 1, phrase end: 1], [concept start: 1, concept end: 1],
 is Head?: true
 is Overmatch?: false
 Sources: [LNC, MSH, MTH, NCI, RCD, SNOMEDCT, OMIM, LCH, ALT]
 Positional Info: [(0, 10)]
Candidate:
 Score: -861
 Concept Id: C0220814
 Concept Name: culture
 Preferred Name: Cultural aspects
 Matched Words: [culture]
 Semantic Types: [ftcn]
 MatchMap: [[[2, 2], [1, 1], 0]]
 MatchMap alt. repr.: [[[phrase start: 2, phrase end: 2], [concept start: 2, concept end: 2],
 is Head?: true
 is Overmatch?: false
```

```
 Sources: [MTH, MSH]
 Positional Info: [(11, 7)]
Candidate:
 Score: -861
 Concept Id: C0430400
 Concept Name: Culture
 Preferred Name: Laboratory culture
 Matched Words: [culture]
 Semantic Types: [lbpr]
 MatchMap: [[[2, 2], [1, 1], 0]]
 MatchMap alt. repr.: [[[phrase start: 2, phrase end: 2], [concept start: 2, concept end: 2],
 is Head?: true
 is Overmatch?: false
 Sources: [MTH, LNC, MDR, NCI, RCD, MEDCIN, CCPSS, SNOMEDCT]
 Positional Info: [(11, 7)]
Candidate:
 Score: -861
 Concept Id: C1706355
 Concept Name: CULTURE
 Preferred Name: Culture Dose Form
 Matched Words: [culture]
 Semantic Types: [bodm]
 MatchMap: [[[2, 2], [1, 1], 0]]
 MatchMap alt. repr.: [[[phrase start: 2, phrase end: 2], [concept start: 2, concept end: 2],
 is Head?: true
 is Overmatch?: false
 Sources: [MTH, NCI]
 Positional Info: [(11, 7)]
Candidate:
 Score: -861
 Concept Id: C2242979
 Concept Name: Culture
 Preferred Name: Culture
 Matched Words: [culture]
 Semantic Types: [lbpr]
 MatchMap: [[[2, 2], [1, 1], 0]]
 MatchMap alt. repr.: [[[phrase start: 2, phrase end: 2], [concept start: 2, concept end: 2],
 is Head?: true
 is Overmatch?: false
 Sources: [MTH, SNOMEDCT, SNM, SNMI]
 Positional Info: [(11, 7)]
Candidate:
 Score: -827
 Concept Id: C1619828
 Concept Name: Laboratories
 Preferred Name: Laboratories
 Matched Words: [laboratories]
 Semantic Types: [inpr]
```

```
  MatchMap: [[[1, 1], [1, 1]]]
  MatchMap alt. repr.: [[[phrase start: 1, phrase end: 1], [concept start: 1, concept end: 1],
  is Head?: true
  is Overmatch?: false
  Sources: [MTH, HL7V3.0]
  Positional Info: [(0, 10)]
Mappings:
 Map Score: -1000
   Score: -1000
   Concept Id: C0430400
   Concept Name: Laboratory culture
   Preferred Name: Laboratory culture
   Matched Words: [laboratory, culture]
   Semantic Types: [lbpr]
   MatchMap: [[[1, 2], [1, 2], 0]]
   MatchMap alt. repr.: [[[phrase start: 1, phrase end: 2], [concept start: 1, concept end: 2],
   is Head?: true
   is Overmatch?: false
   Sources: [MTH, LNC, MDR, NCI, RCD, MEDCIN, CCPSS, SNOMEDCT]
   Positional Info: [(0, 18)]
$
```

The source to java class used `testapi.sh` is in `./src/javaapi/sources/gov/nih/nlm/nls/metamap/MetaMapApiT`

7 Using the API

7.1 Instantiating the API

The following sections expose the source code used to produce the example output shown in previous section

```
MetaMapApi api = new MetaMapApiImpl();
```

If one is running the MetaMap server (mmserver) of a machine other than the one the Java api client is running, one can specify the hostname of the MetaMap server when instantiating the api: If the MetaMap server is running on the host: "resource.example.org" then the instantiation would be as follows:

```
MetaMapApi api = new MetaMapApiImpl("resource.example.org");
```

7.2 Setting MetaMap options

```
    List<String> theOptions = new ArrayList<String>();
    theOptions.append("-y");  // turn on Word Sense Disambiguation
    if (theOptions.size() > 0) {
      api.setOptions(theOptions);
    }
```

MetaMap options available in the api:

```
-@ --WSD <hostname>                    : Which WSD server to use.
-8 --dynamic_variant_generation        : dynamic variant generation
-A --strict_model                      : use strict model
-C --relaxed_model                     : use relaxed model
-D --all_derivational_variants         : all derivational variants
-J --restrict_to_sts <semtypelist>     : restrict to semantic types
-K --ignore_stop_phrases               : ignore stop phrases.
-R --restrict_to_sources <sourcelist>  : restrict to sources
-S --tagger <sourcelist>               : Which tagger to use.
-V --mm_data_version <name>            : version of MetaMap data to use.
-X --truncate_candidates_mappings      : truncate candidates mapping
-Y --prefer_multiple_concepts          : prefer multiple concepts
-Z --mm_data_year <name>               : year of MetaMap data to use.
-a --all_acros_abbrs                   : allow Acronym/Abbreviation variants
-b --compute_all_mappings              : compute/display all mappings
-d --no_derivational_variants          : no derivational variants
-e --exclude_sources <sourcelist>      : exclude semantic types
-g --allow_concept_gaps                : allow concept gaps
-i --ignore_word_order                 : ignore word order
-k --exclude_sts <semtypelist>         : exclude semantic types
-l --allow_large_n                     : allow Large N
-o --allow_overmatches                 : allow overmatches
-r --threshold <integer>               : Threshold for displaying candidates.
-y --word_sense_disambiguation         : use WSD
-z --term_processing                   : use term processing
```

7.3 Performing a query using the api

```
List<Result> resultList = api.processCitationsFromString(terms);
```

7.4 Interrogating the result

7.4.1 Getting Acronyms and Abbreviations

To get a list of all the acronyms and abbreviations occurring in the input text use the instance method `Result.getAcronymsAbbrevs`:

```
Result result = resultList.get(0);
List<AcronymsAbbrevs> aaList = result.getAcronymsAbbrevs();
if (aaList.size() > 0) {
  System.out.println("Acronyms and Abbreviations:");
  for (AcronymsAbbrevs e: aaList) {
    System.out.println("Acronym: " + e.getAcronym());
    System.out.println("Expansion: " + e.getExpansion());
    System.out.println("Count list: " + e.getCountList());
    System.out.println("CUI list: " + e.getCUIList());
```

```
  }
} else {
  System.out.println(" None.");
}
```

7.4.2 Getting Negations

To get a list of all the negated concepts in the input text use the instance method `Result.getNegations`:

```
List<Negation> negList = result.getNegations();
if (negList.size() > 0) {
  System.out.println("Negations:");
  for (Negation e: negList) {
    System.out.println("type: " + e.getType());
    System.out.print("Trigger: " + e.getTrigger() + ": [");
    for (Position pos: e.getTriggerPositionList()) {
      System.out.print(pos  + ",");
    }
    System.out.println("]");
    System.out.print("ConceptPairs: [");
    for (ConceptPair pair: e.getConceptPairList()) {
      System.out.print(pair + ",");
    }
    System.out.println("]");
    System.out.print("ConceptPositionList: [");
    for (Position pos: e.getConceptPositionList()) {
      System.out.print(pos + ",");
    }
    System.out.println("]");
  }
} else {
System.out.println(" None.");
}
```

7.4.3 Getting Utterances and Associated Phrases, Candidates, and Mappings

The instance method `Result.getUtteranceList()` produces a list of the utterances present in the result:

```
for (Utterance utterance: result.getUtteranceList()) {
System.out.println("Utterance:");
System.out.println(" Id: " + utterance.getId());
System.out.println(" Utterance text: " + utterance.getString());
System.out.println(" Position: " + utterance.getPosition());
```

To get the list of phrases, candidates, and mappings associated with an utterance use the instance method `Utterance.getPCMList`:

```
for (PCM pcm: utterance.getPCMList()) {
```

Each phrase, and the list of candidates and mappings associated with the phrase are encapsulated within a PCM instance. Use `PCM.getPhrase` to get the phrase instance residing within the PCM instance:

```
System.out.println("Phrase:");
System.out.println(" text: " + pcm.getPhrase().getPhraseText());
```

Similarly, get the candidate list using `PCM.getCandidateList()`:

```
System.out.println("Candidates:");
for (Ev ev: pcm.getCandidateList()) {
  System.out.println(" Candidate:");
  System.out.println("  Score: " + ev.getScore());
  System.out.println("  Concept Id: " + ev.getConceptId());
  System.out.println("  Concept Name: " + ev.getConceptName());
  System.out.println("  Preferred Name: " + ev.getPreferredName());
  System.out.println("  Matched Words: " + ev.getMatchedWords());
  System.out.println("  Semantic Types: " + ev.getSemanticTypes());
  System.out.println("  MatchMap: " + ev.getMatchMap());
  System.out.println("  MatchMap alt. repr.: " + ev.getMatchMapList());
  System.out.println("  is Head?: " + ev.isHead());
  System.out.println("  is Overmatch?: " + ev.isOvermatch());
  System.out.println("  Sources: " + ev.getSources());
  System.out.println("  Positional Info: " + ev.getPositionalInfo());
}
```

One can get the mappings list from the PCM instance using `PCM.getMappingList`:

```
System.out.println("Mappings:");
for (Mapping map: pcm.getMappingList()) {
  System.out.println(" Map Score: " + map.getScore());
  for (Ev mapEv: map.getEvList()) {
    System.out.println("   Score: " + mapEv.getScore());
    System.out.println("   Concept Id: " + mapEv.getConceptId());
    System.out.println("   Concept Name: " + mapEv.getConceptName());
    System.out.println("   Preferred Name: " + mapEv.getPreferredName());
    System.out.println("   Matched Words: " + mapEv.getMatchedWords());
    System.out.println("   Semantic Types: " + mapEv.getSemanticTypes());
    System.out.println("   MatchMap: " + mapEv.getMatchMap());
    System.out.println("   MatchMap alt. repr.: " + mapEv.getMatchMapList());
    System.out.println("   is Head?: " + mapEv.isHead());
    System.out.println("   is Overmatch?: " + mapEv.isOvermatch());
    System.out.println("   Sources: " + mapEv.getSources());
    System.out.println("   Positional Info: " + mapEv.getPositionalInfo());
  }
}
```

```
    }
}
```

Refer to the API javadoc for more information on the available methods for each interface.

A complete example of this code is in `src/javaapi/sources/gov/nih/nlm/nls/metamap/MetaMapApiTest.java`.

7.5 Getting Raw MetaMap Machine Output

A copy of the raw MetaMap machine output can be obtained by using the instance method `Result.getMachineOutput`:

```
List<Result> resultList = api.processCitationsFromString(terms);
Result result = resultList.get(0);
String machineOutput = result.getMachineOutput();
```

8 Advanced Configuration

8.1 Accepting the client connections from remote hosts

To allow the MetaMap server to accept connections from clients on a host other than localhost (127.0.0.1, the default) modify the environment variable ACCEPTED**HOSTS the script public**mm/bin/SKRrun.11 to contain all of the client hosts you wish to have access to the MetaMap server.

For example, change the entry:

```
ACCEPTED_HOSTS="['127.0.0.1']"
export ACCEPTED_HOSTS
```

to:

```
ACCEPTED_HOSTS="['127.0.0.1','192.168.111.27','192.168.111.61','192.168.111.76']"
export ACCEPTED_HOSTS
```

all of the entries must be ip-addresses, hostnames will not work.

8.2 Running the MetaMap server on an alternate port

To run the MetaMap server on a port other than the default port (8066) modify the environment variable "MMSERVER_PORT" in the script public_mm/bin/SKRrun.{year}.

E.G. to change the port to 8888:

```
MMSERVER_PORT=8888
export MMSERVER_PORT
```

8.3 Specifying alternate MetaMap Server hosts and ports in the API

The MetaMap Java API now includes the methods `setHost` and `setPort` to specify the host and port locations of the MetaMap server. The source to the class gov.nih.nlm.nls.metamap.MetaMapApiTest (public_mm/src/javaapi/sources/gov/nih/nlm/nls/metamap/ MetaMapApiTest.java) contains an example of the use of these methods.

9 Notes for Maven Users

To install the MetaMap api and its supporting libraries use the following commands:
First install PrologBeans:

```
$ mvn install:install-file \
  -Dfile=<parent>/public_mm/src/javaapi/dist/prologbeans.jar \
  -DgroupId=se.sics -DartifactId=prologbeans -Dversion=4.2.1 \
  -Dpackaging=jar
```

Then install MetaMap API:

```
$ cd <parent>/public_mm/src/javaapi
$ mvn install
```

10 What is missing from the API

The server sorely needs support for dealing with internal exceptions gracefully when processing options and input text.

11 Possible enhancements

Possible enchancments include providing a UIMA module that uses an underlying MetaMapApi instance, providing a factory method for instantiating MetaMapApi instances, and providing instantiation of instances through JNDI. A mechanism for querying the Prolog Server to determine which options are available and how to use them.

12 Differences compared to MMTx API

The MetaMap Java API only provides access to components available through machine output. That includes The Final mappings, the Candidates, Phrases, Utterances, Negated Concepts, and Acronyms and Abbreviations. Access to structures such as lexical elements and tokenization that were available in the eariler MMTx API are not currently available in the MetaMap API.

13 For more information

SICStus Prolog http://www.sics.se/sicstus/

PrologBeans Docs http://www.sics.se/sicstus/docs/latest4/html/sicstus.html

PrologBeans Java API Docs http://www.sics.se/sicstus/docs/3.12.9/html/prologbeans/

MetaMap http://metamap.nlm.nih.gov/