**MetaMap wrapper project**

*Written by Jonathan Shifman  
February 2018*

**General Description**

**Background**

MetaMap is a tool developed at the National Library of Medicine to analyze biomedical texts, to locate UMLS Metathesaurus concepts and map those concepts to standardized names. The UMLS Metathesaurus itself is a large biomedical thesaurus that contains the data representing the mappings from different names with similar meaning to the same concept.

The output generated by MetaMap includes extensive information about each concept it was able to locate in a given text.

Let’s look at an example. Consider the following phrase:

*‘Classification systems aim to separate hypertension similar to that seen outside pregnancy’*

MetaMap is able to map numerous Metathesaurus concepts in the given sentence. To name a few of the main located concepts: ‘**systems’**, ‘**hypertension’**, ‘**pregnancy’**.  
We will later go over in details about the output from analyzing the sentence.

MetaMap’s services are offered using a variety of different APIs. The quickest way to use it is interactively – you can submit your medical text in an online form and an output will be generated. The method we will be using is running MetaMap locally on the server we are working on. To be precise, we will be using a wrapper (called pymetamap) of the Java API that Meta Map provides.

Main project

**This project**

MetaMap

pymetamap

UMLS

Metathesaurus

This project is intended to serve as a utility for a project that aims to analyze biomedical texts, gather information about the nature of their content and be able to detect and link articles that are connected in terms of topic.

As such, this project was designed to offer an easy-to-use API that can be configured to extract the needed information from MetaMap and be inserted into the flow of the main project.

**Basic Use Case Example**

Let’s go back to the sentence that was given as an example earlier:

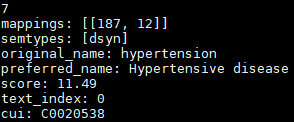
*‘Classification systems aim to separate hypertension similar to that seen outside pregnancy’*

As mentioned, MetaMap was able to locate concepts in this sentence. For each such concept, this is the main info that is being generated and can be accessed by the user of this project:

* **cui** – Concept unique identifier
* **preferred\_name** – The preferred name of the UMLS concept, i.e. the standardized concept that the original concept has been mapped to.
* **score** – Indexing score (the maximum being 1000). The higher the score – the greater the relevance of the concept.
* **semtypes** – Comma separated list of the semantic types the concept belongs to.
* **pos\_info** – Positional info for the concept, i.e. the index where it started and its length. This value can contain multiple entries, in case the concept doesn’t appear in succession. This will be discussed in more detail later.
* **tree\_codes** – Semicolon separated list of MeSH treecode associated with the concept.
* **original\_name** – The original name, i.e. the phrase that appeared in the analyzed sentence.
* **text\_index** – The index of the text this concept belongs to (out of the ones that have been passed to the analyze\_texts method).

Usually only certain parts of this info will be useful. The project makes it possible to configure the desired elements that should be returned as an output.

Here’s an example for the returned output regarding one of the mapped concepts:



As mentioned, The values that are included in the output can be configured in the config file.

**API and key features**

The main functional method offered by the MetaMap Wrapper class is analyze\_texts:

|  |  |
| --- | --- |
| **analyze\_texts**(self, texts) | **texts** – a list (or any collection) of texts to be processed by the wrapper. The text\_index value for each located concept will correspond to the index of the sentence in the given list.  **Return value** – A dictionary containing all concepts that were located in the texts, along with all the required info based on the contents of the config file. |

The method returns a list of concept dictionaries – each representing a single located concept.

Each concept dictionary contains mappings between names of the requested output fields and their corresponding values.

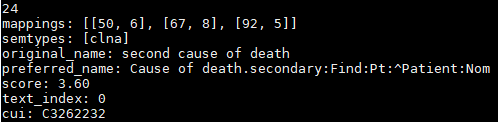
Note that the analyze\_texts method supports input that consists of multiple sentences (or multiple blocks of text in general). Among the other available information, it is specified for each concept which text block it was originally located in.

**Non-successive concepts**

Furthermore, MetaMap has the ability to map concepts that are not necessarily written in succession within the given sentence. For example:

*'Hypertensive disorders of pregnancy represent the second commonest cause of direct maternal death and complicate an estimated 5-10 of pregnancies. '*

This particular concept (‘second cause of death’) is made of three separate parts, each appearing at a different location within the original sentence. Those concepts are returned by the wrapper as well, with appropriate indices specifying the location of all parts of the concept in the original sentence:



Note that the value for ‘mappings’ contains 3 different locations.

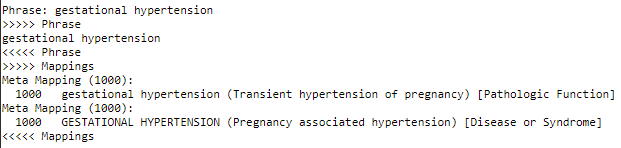
**Semantic category**

As part of its output, MetaMap offers the categorization of every concept to one (or more) of many possible semantic types. For example, one of the possible types is ‘Disease or Syndrome’, which is pretty self explanatory.

This feature makes it easier to draw conclusion about the general topic of a given article.

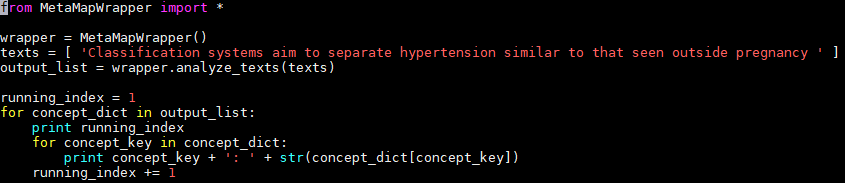
**Multiple mappings for a single concept**

Another feature of MetaMap is the ability to produce multiple mappings from a single concept to different semantic types (possibly containing each other). For example, for the concept ‘gestational hypertension’:



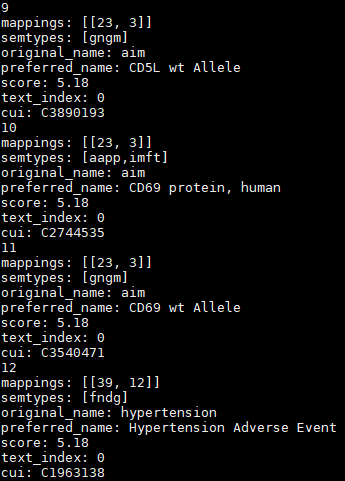
**Usage Instructions and Code Samples**

We will now demonstrate a basic example of the wrapper’s usage in code. Following is a class that passes a single sentence to the analyze\_texts method:



This class initializes an instance of MetaMapWrapper, passes the sentence as input, and prints the returned values of all concept dictionaries in output\_list.

Part of the generated output:



The example is included in the project repository in the UsageExample.py file.

**Set-up**

To use the wrapper, you will need to perform the following steps:

Download & Install MetaMap

Download the latest version of MetaMap from the [downloads page](https://metamap.nlm.nih.gov/MainDownload.shtml).  
Make sure to download the Linux release, since the project is currently not compatible with windows.

Follow the installation instructions to install MetaMap. Be sure to start the Tagger server as described.

Install pymetamap

You will need to install the pymetamap module, which can be found in [this GitHub repository](https://github.com/AnthonyMRios/pymetamap).

Download the repository and run:

*python setup.py install*

Set up the wrapper

Run *python setup.py* from the wrapper directory to install the necessary modules.

Configure the wrapper to use the MetaMap binary

In the config.ini file, set the meta\_map\_path variable to point to the binary of Meta Map you installed.  
For example: /home/public\_mm/bin/metamap16

Set the relevant\_field\_names variable to contain the fields you want included in each concept dictionary. The possible field values are listed in the Basic Use Case Example section.

After having performed these steps, you can start using the wrapper.

**Sources & Further Information**

[Project repository](https://github.com/TechnionTDK/meta-map-wrapper)

[pymetamap repository](https://github.com/AnthonyMRios/pymetamap)

[MetaMap website](https://metamap.nlm.nih.gov/)

[Interactive MetaMap API](https://ii.nlm.nih.gov/Interactive/UTS_Required/metamap.shtml)

[MetaMap Java release downloads page](https://metamap.nlm.nih.gov/JavaApi.shtml)

[MetaMap output documentation](https://metamap.nlm.nih.gov/Docs/MMI_Output.pdf)

[UMLS website](https://www.nlm.nih.gov/research/umls/)