# Cabinet

*Release 0.1.0*

Nick Anthony

Apr 18, 2023

# CONTENTS:

This package is for a bunch of code that we frequently replicate here at IPOP and to provide programmatic access to tools developed such as the REST API, and MetaMap without needed to install a bunch of dependencies.

This site is for documenting the functions themselves and how to use them. It will be technical and detailed, not a tutorial. For a more general overview of the package, see the README.

For more information, see the GitHub README.

CHAPTER

ONE

# CABINET PACKAGE

## 1.1 Subpackages

### 1.1.1 cabinet.cleaning_drawer package

#### Submodules

#### cabinet.cleaning_drawer.normalize module

This module contains code for typical data normalization tasks.

We generally enforce 'type-saftey' by using pydantic's validate_arguments decorator and other pydantic types.

`cabinet.cleaning_drawer.normalize.`**`categorize_age`**(*age: PositiveInt*) → str

Categorize an age into a string.

**Parameters**
**age** (*PositiveInt*) – Age to categorize. Must be a positive integer.

**Returns**
Categorized age.

**Return type**
str

#### Examples

```
>>> categorize_age(10)
'<18'
>>> categorize_age(18)
'18-25'
>>> # a general example using pandas
>>> df['age'].apply(categorize_age)
pandas.Series(['<18', '18-25', '26-35', '36-45', '46-55', '56-65', '65+'])
```

#### Module contents

This drawer exists for cleaning functions.

It’s a bit of a catch-all for functions that don’t fit in any other drawer.

### 1.1.2 cabinet.umls_drawer package

#### Submodules

#### cabinet.umls_drawer.knowledge_base module

This module contains the knowledge base class.

Currently it simply loads the concept conversion map and SNOMED tree from disk and provides a few functions to access them.

In the future, we want to support tree traversal specifically. This will allow us to get paths to the root, paths to any given terminal node, and paths to common ancestors/children. We can do this in a recursive way, but the exact implementation is still undecided for how to best handle the interface.

We also want to support “prettifying” these paths using string formatting such as ‘/’ or ‘->’. This will allow us to get paths like ‘A/B/C’ or ‘A->B->C’ instead of [‘A’, ‘B’, ‘C’].

`class cabinet.umls_drawer.knowledge_base.`**`Knowledge`**

Bases: `BaseModel`

A class to hold the knowledge base.

This class loads the data from disk for you and provides a few functions to access the concept map and SNOMED tree.

**`convert`**`(`*cui: str*`)` → str | None

Convert a CUI to a SNOMED code.

Will return None if the CUI is not in the map. This will occur if it is the root SNOMED concept (138875005) as it has no parents.

**Parameters**
**`cui`** (*`str`*) – The CUI to convert.

**Returns**
The SNOMED code, or None if the CUI is not in the map. None will occur if it is the root SNOMED concept as it has no parents.

**Return type**
str | None

**`tree_get_children`**`(`*sctid: str*`)` → set[str] | None

Get the children of a SNOMED code.

**Parameters**
**`sctid`** (*`str`*) – The SNOMED code to get the children of.

**Returns**
The children of the SNOMED code, or None if the SNOMED code has no children.

**Return type**
set[str] | None

**tree_get_parents**(*sctid: str*) → set[str] | None

Get the parents of a SNOMED code.

**Parameters**
`sctid` (*str*) – The SNOMED code to get the parents of.

**Returns**
The parents of the SNOMED code, or None if the SNOMED code is not in the tree.

**Return type**
set[str] | None

`cabinet.umls_drawer.knowledge_base.`**load_cui_map**`()` → dict[str, str]

`cabinet.umls_drawer.knowledge_base.`**load_snomed_tree**`()` → dict[str, set[str]]

### cabinet.umls_drawer.metamap_ner module

This module is for working with MetaMap.

MetaMap is a tool for extracting structured information from biomedical text provided by the National Library of Medicine (NLM). MetaMap is a text processing engine is a natural language processing (NLP) system that uses a set of rules and heuristics to identify and extract concepts from unstructured text.

You can download MetaMap by purchasing a NLM License (or accessing via your institution) and downloading the binary from [here](https://lhncbc.nlm.nih.gov/ii/tools/MetaMap/run-locally/MainDownload.html).

For more information, see the [MetaMap documentation](https://lhncbc.nlm.nih.gov/ii/tools/MetaMap.html).

`class cabinet.umls_drawer.metamap_ner.`**MMOutputType**(*value*, *names=None*, *, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `Enum`

Enum for MetaMap output types.

**JSON = 'json'**

//lhncbc.nlm.nih.gov/ii/tools/MetaMap/Docs/JSON.pdf) for more information.

**Type**
Json output, see [here](https

**MMI = 'mmi'**

//lhncbc.nlm.nih.gov/ii/tools/MetaMap/Docs/MMI_Output.pdf) for more information.

**Type**
Fielded MMI output format, see [here](https

`class cabinet.umls_drawer.metamap_ner.`**MetaMap**(*, *metamap_location: Path*)

Bases: `BaseModel`

Class for running MetaMap on a text string.

**Parameters**
`metamap_location` (*str*) – The path to the MetaMap installation. This should be the path to the *public_mm* directory.

### Example

`` `python from cabinet.umls_drawer import MetaMap mm = MetaMap(metamap_location="/Users/username/metamap/public_mm") ` ``

**`initialize`**`()` → None

Initialize MetaMap.

This function must be run to start the MetaMap servers. It will check if the servers are already running and if not, it will start them.

**`metamap_location: Path`**

**`run`**(*text: str*, *output_type: Literal['mmi', 'json'] = 'mmi'*) → list[str] | str | None

Run MetaMap on a text string.

This will return None if no results were found, otherwise the return type will match the output_type argument. Future work will include returning a dataclass for each result, but for now:

MMOutputType.JSON -> str (the json data itself) MMOutputType.MMI -> list[str] (each line of the MMI output)

**Parameters**

- **`text`** (*`str`*) – The text to run MetaMap on.
- **`output_type`** (*`Literal["mmi", "json"], optional`*) – The output type. Defaults to “mmi”.

**Returns**

The output of MetaMap. The type of the output depends on the *output_type* argument.

**Return type**

list[str] | str | None

### Example

`` `python >>> from cabinet.umls_drawer import MetaMap, MMOutputType >>> mm = MetaMap(metamap_location="/Users/username/metamap/public_mm") >>> mm.initialize() >>> results = mm.run(text="I have a headache.", output_type=MMOutputType.MMI) >>> print(results) ` ``

**`run_many`**(*texts: list[str]*, *output_type: Literal['mmi', 'json'] = 'mmi'*) → Iterator[list[str] | str | None]

Runs MetaMap on multiple strings.

Calls thread_map from tqdm.contrib.concurrent to run MetaMap on multiple strings.

Returns an iterator that must be consumed.

**Parameters**

- **`texts`** (*`list[str]`*) – The texts to run MetaMap on.
- **`output_type`** (*`Literal["mmi", "json"], optional`*) – The output type. Defaults to “mmi”.

**Returns**

An iterator that must be consumed. The type of the output depends on the *output_type* argument.

**Return type**
Iterator[list[str] | str | None]

### Example

```python >>> from cabinet.umls_drawer import MetaMap >>> mm = MetaMap(metamap_location="/Users/username/metamap/public_mm") >>> mm.initialize() >>> results = mm.run_many(texts=["I have a headache.", "I have a fever."]) >>> for result in results:

print(result)

```

## cabinet.umls_drawer.scispacy_ner module

This module contains functions for interacting with the scispacy NER model via our API.

The private functions utilize async/await syntax and are used by the public functions which are synchronous. The public functions are the ones that you should use in your code unless you are confident that you know what you are doing.

The core type of this module is *NEROutput* which is a pydantic model that represents the output from the scispacy NER model. All public functions return either an instance of this class or an iterator of instances of this class attached to an index (tuple[int, NEROutput]) for the index of the text that was submitted… this helps with link to original data.

*web_socket_ner*, specifically, returns an iterator and thus needs to be consumed to be used: `` `python >>> from cabinet.umls_drawer.scispacy import web_socket_ner >>> for text_index, ner_output in web_socket_ner(texts=["cocaine", "heroin", "cociane"]): ... print(text_index, ner_output) 0 cui='12' concept_name='test' concept_definition='test22' entity='{"text":"cocaine"}' score=1.0 1 cui='12' concept_name='test' concept_definition='test22' entity='{"text":"heroin"}' score=1.0 2 cui='12' concept_name='test' concept_definition='test22' entity='{"text":"cociane"}' score=1.0 ` ``

**class** `cabinet.umls_drawer.scispacy_ner.`**`NEROutput`**(***, *cui: str*, *concept_name: str*, *concept_definition: str*, *entity: str*, *score: float*)

Bases: `BaseModel`

Output from the ([scispacy](https://github.com/allenai/scispacy/tree/4f9ba0931d216ddfb9a8f01334d76cfb662738ae)) NER model.

This class is keyword only so you must pass in the arguments as: *cui="C0004096", concept_name="Acetaminophen", …*

**Args/Attributes:**
cui (str): The UMLS CUI. concept_name (str): The UMLS concept name. concept_definition (str): The UMLS concept definition. entity (str): The entity that matched a UMLS concept from the source text. score (float): The score of the match.

### Examples

An example of manually creating this class: `` `python >>> from cabinet.umls_drawer import NEROutput >>> NEROutput( ... cui="C0004096", ... concept_name="Acetaminophen", ... concept_definition="A nonsteroidal anti-inflammatory drug that is used as an analgesic and antipyretic. It is also used in the treatment of rheumatoid arthritis and osteoarthritis.", ... entity="acetaminophen", ... score=0.96, ... ) ` ``

However, much more likely is that you get this class as a return type from one of the various functions in this module that make calls to our API. ```python >>> from cabinet.umls_drawer import post_ner_single >>> post_ner_single(text="cocaine") [

0 cui='12' concept_name='test' concept_definition='test22' entity='{"text":"cocaine"}' score=1.0

### ]

**concept_definition: str**
The UMLS concept definition.

**concept_name: str**
The UMLS concept name.

**cui: str**
The UMLS CUI.

**entity: str**
The entity that matched a UMLS concept from the source text.

**score: float**
The score of the match.

`cabinet.umls_drawer.scispacy_ner.`**`post_ner_many`**(*texts: list[str]*, *with_progress: bool = True*) → list[tuple[int, *cabinet.umls_drawer.scispacy_ner.NEROutput*]]

Submit multiple text blobs to the scispacy NER model and return the results.

**Parameters**

- **texts** (*list[str]*) – The texts to submit to the NER model.
- **with_progress** (*bool, optional*) – Whether or not to show a progress bar. Defaults to True.

**Returns**
The results from the NER model.

**Return type**
Iterator[tuple[int, *NEROutput*]]

**Raises**
**Exception** – If the response status is not 200.

### Example

```python >>> from cabinet.umls_drawer import post_ner_many >>> post_ner_many(texts=[“acetaminophen”, “ibuprofen”]) [

**(0, NEROutput(**
cui=”C0004096”, concept_name=”Acetaminophen”, concept_definition=”A nonsteroidal anti-inflammatory drug that is used as an analgesic and antipyretic. It is also used in the treatment of rheumatoid arthritis and osteoarthritis.”, entity=”acetaminophen”, score=0.96,

)), (1, NEROutput(

cui=”C0004096”, concept_name=”Ibuprofen”, concept_definition=”A nonsteroidal anti-inflammatory drug that is used as an analgesic and antipyretic. It is also used in the treatment of rheumatoid arthritis and osteoarthritis.”, entity=”ibuprofen”, score=0.96,

))

]

`cabinet.umls_drawer.scispacy_ner.`**`post_ner_single`**`(`*text: str*`)` → list[*cabinet.umls_drawer.scispacy_ner.NEROutput*]

Submit a single text blob to the scispacy NER model and return the results.

**Parameters**
**`text`** (*`str`*) – The text to submit to the NER model.

**Returns**
The results from the NER model.

**Return type**
list[*NEROutput*]

**Raises**
**`Exception`** – If the response status is not 200.

### Example

```python >>> from cabinet.umls_drawer import post_ner_single >>> post_ner_single(text=”acetaminophen”) [

**NEROutput(**
cui=”C0004096”, concept_name=”Acetaminophen”, concept_definition=”A nonsteroidal anti-inflammatory drug that is used as an analgesic and antipyretic. It is also used in the treatment of rheumatoid arthritis and osteoarthritis.”, entity=”acetaminophen”, score=0.96,

)

]

`cabinet.umls_drawer.scispacy_ner.`**`websocket_ner`**`(`*texts: list[str]*, *with_progress: bool = True*`)` → AsyncIterator[tuple[int, *cabinet.umls_drawer.scispacy_ner.NEROutput*]]

Connect to the scispacy NER model websocket and submit texts.

*IMPORTANT*: This function requires using the *async for* syntax and thus may not work in all scenarios or environments. It exists for **very** large datasets where the overhead of the HTTP request/response cycle is too much.

**Parameters**

- **`texts`** (*`list[str]`*) – The texts to submit to the NER model.

- **with_progress** (*bool, optional*) – Whether or not to show a progress bar. Defaults to True.

**Yields**

*AsyncIterator[tuple[int, NEROutput]]* – The results from the NER model.

**Raises**

**Exception** – If the response status is not 200.

#### Example

```python >>> from cabinet.umls_drawer import websocket_ner >>> async for i, result in websocket_ner(texts=["acetaminophen", "ibuprofen"]): … print(i, result) 0 cui='12' concept_name='test' concept_definition='test22' entity='{"text":"cocaine"}' score=1.0 1 cui='12' concept_name='test' concept_definition='test22' entity='{"text":"heroin"}' score=1.0 2 cui='12' concept_name='test' concept_definition='test22' entity='{"text":"cociane"}' score=1.0

### Module contents

This drawer is for UMLS related activities.

The *scispacy_ner* module gives you access to the scispacy biomedical NER model via our API.

The *metamap_ner* module interacts with the MetaMap NLP tool to extract structured information from biomedical text and requires you to have MetaMap installed locally.

The *knowledge_base* module allows you to interact with the UMLS knowledge base data at a high level and mostly focuses on SNOMED CT concepts. Further work on this module may take advantage of the *entire* UMLS, but require a locally downloaded copy due to licensing restrictions.

In general, we recommend using the *scispacy_ner* module for NER tasks and the *knowledge_base* module for knowledge base related tasks unless you specifically need the power of MetaMap.

The *post_ner* methods exposed here utilize the API to perform NER on your text.

## 1.2 Submodules

## 1.3 cabinet.utils module

Utility functions and classes for the cabinet package.

Mostly used to initialize the environment and console.

## 1.4 Module contents

Our cabinet of tools.

**Highlights:**

- Local MetaMap operations
- SciSpacy NER via API
- SNOMED tree traversal
```

- UMLS CUI to SNOMED CUI
- Common data normalization tasks

In general, we try to expose the high-level functionality of these tools at the top level of their corresponding "drawers" (e.g. *cabinet.umls_drawer*).

This way you can import a drawer and use its functionality specifically.

For example: ``` `python from cabinet import umls_drawer umls_drawer.post_ner_single("I have a headache.") ` ```

If you want more granular control/exposure, check out the underscore methods inside the drawers although this is not recommended practice.

CHAPTER

# TWO

# CABINET

CHAPTER

THREE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

# INDEX