Installation instructions and general notes about the application and files

The app was developed and tested on a Windows 10 laptop with 6gb of RAM. The following things were needed to make it run:

1. XAMPP – to run application on a local server with PHP.
    a. Setting was changed in php.ini to allow for php files to run for more than 30 seconds. This was needed for the indexer.php to have enough time to index json docs into Elasticsearch. Indexing 477 docs took 1-2 minutes.

    ```
    max_execution_time=10000
    ```

    b. Place all files from "Search app htdocs files" folder into the one you created for the app in xampp (e.g. c:\xampp\htdocs\searchapp).
2. Composer – to install Elasticsearch PHP client. Install Composer using instructions here. Once Composer is installed, to install Elasticsearch PHP client, open XAMPP htdocs directory where you placed this search app's composer.json and composer.lock files, and run:

    ```
    composer install
    ```

3. Python 3.6 – to install leveldb for QuickUMLS. To get leveldb pre-built binary for Windows and for instructions on where to put it go here.
4. Anaconda – to install Simstring as per instructions from here, and to run all python scripts related to this app that use QuickUMLS including QuickUMLS installation.
    a. An environment was created in anaconda as per supplied condaumlsenvironment.yml file. Leveldb python binary added in there.
    b. Due to leveldb for Windows requiring Python 3.6, the Anaconda environment (or the whole Anaconda version) needs to have Python 3.6. Place leveldb binary into the Anaconda environment python folder as per instructions in point 3.
5. QuickUMLS – modified version as supplied, to be based around SNOMED CT codes instead of UMLS cuis.
    a. Requires UMLS installation first.
    b. Initialise and run QuickUMLS (from Anaconda environment if in Windows) with supplied modified files. If you already have QuickUMLS installed, copy files from "QuickUMLS just the modified files" folder into your QuickUMLS folder, overwriting the files, and then rerun QuickUMLS initialisation. In case they are needed, I put all QuickUMLS files used on my laptop into "QuickUMLS all files" folder (remember to install Simstring if not already). Initialisation creates the database of SNOMED CT concepts IDs with concept names, which is used for subsequent concept extractions. I did the following to install QuickUMLS in Windows after doing items 3 and 4 and following instructions here up to "How to

get the System Initialized” step 2 (not including). In other words, got Simstring installed in step 4 and then did the following:

(1) modified toolbox.py to "import simstring" instead of "from simstring..." since it was not possible to do --inplace install of Simstring in Anaconda.

(2) open Conda console

(3) run the following to activate the dedicated Anaconda environment (e.g. “umls”):

```
activate umls
```

(4) navigate within the console to where QuickUMLS files are and Simstring is installed

(5) run the following (change location of UMLS META folder and desired QuickUMLS data folder if needed) to initialise QuickUMLS:

```
python install.py -L -U c:\2018AB\META c:\quickumls-data
```

c. semtypes.py includes the desired semantic types to be used by QuickUMLS during concept extraction. The server.py is changed to use this file instead of constants.py.

d. Once QuickUMLS is initialised, the QuickUMLS server is started using this command (settings can be changed as required). The server needs to be running for concept extraction requested by the search app or for text/document annotation:

```
python server.py c:\quickumls-data -o length -w 15 -t 0.8
```

I assume that items 3 and 4 are not required (except Python) if this app is run on Linux or Mac, as this is mainly due to leveldb and Simstring installation issues in Windows. In which case, point 5.b.(1) may need reversing.

QuickUMLS initialisation will be centred around SNOMED CT concept IDs instead of UMLS cuis. Some of the lines in MRCONSO.RRF file are marked with O for obsolete (or suppressed). From my understanding it refers to the concept linked to that particular SNOMED CT code (or possibly to UMLS cui). Some codes are inactive on such lines (if checked in Ontoserver), but others are not, in which case it is just the term that’s marked as obsolete when used with that code. I modified install.py to check if an obsolete line’s SNOMED CT code exists in MRCONSO.RRF file on a line that’s marked with N (non-obsolete). If exists, then all concept terms with that code will be stored. If not, then they won’t be. During initialisation two files are created, concepts_added.txt and concepts_rejected.txt containing such concepts. I added them into Misc folder if you wish to examine them. I also tried excluding all obsolete lines, regardless if that code has non-obsolete ones. That stopped recognising DVT as a concept, so I reverted back.

When running the QuickUMLS server for use with the search app, it’s probably best to change the window (-w) setting to 12 – 15. This would ensure that the really long fully qualified concept names given by Ontoserver are successfully converted to concepts. For annotating plain-text EHRs a smaller window would probably be sufficient, as it’s unlikely for fully qualified names to be in plain text. Window setting of 10 was used during file annotation for thesis.

6. Place extract.py, extractor.py, and display_names.json files (from the “Python scripts for extraction” folder) into the QuickUMLS folder (where quickumls.py is).
    a. extractor.py uses functions from extract.py to run concept extraction for the search app.
    b. extract.py contains functions to extract concepts from text and to create JSON files with marked-up text and concept categories as per Thesis.pdf based on plain-text files. It has different helper functions for that to happen. It needs server.py to be running to work as per 5.d.
        i. The annotation commands for text, file and directory are similar to The Health Search tutorial. An option to include concept descendants is added for text.
            1. python extract.py “some text”
            2. python extract.py -f somefile.txt
            3. python extract.py -d somedir
            4. python extract.py -c “some text to get concepts with descendants”
        ii. extract.py uses semtypes.py file during json file creation. That file has a map that indicates which category a semantic type belongs to. If different categories or semantic types are needed, this file will need modifying.
    c. When using extract.py to annotate a set of text files and create the corresponding JSON files for indexing, a display_names.json file is created (or updated if one exists) that contains the preferred name of each extracted concept as obtained using Ontoserver API. These names are shown in the charts on the search app. They are not the fully qualified names, as those would occupy even more space in the charts. A copy of display_names.json file is then placed into the search app’s htdocs folder to be used for the charts. So if a new set of documents is created for indexing, then the file in htdocs folder needs to be updated. The current file is for documents in EHRs/annotated_json_0.8 folder.
    d. File annotation also creates multicategory.txt. This file records concept IDs that are classified under two or more categories within one file. It’s not used for anything else.
7. Elasticsearch version 7.0.0.
    a. Need to install a Mapper Annotated Text plugin. This allows proper highlighting of embedded concepts. Blog about it is here. To install per instructions here, from inside Elasticsearch_installation directory run (with sudo for Linux?):

        ```
        bin/elasticsearch-plugin install mapper-annotated-text
        ```

    b. Need to change setting to allow for huge number of internal Boolean clauses during searching. Added the following line to elasticsearch.yml file inside C:\ProgramData\Elastic\Elasticsearch\config:

        ```
        indices.query.bool.max_clause_count: 50000
        ```

8. If you want to run the search app using existing annotated docs that were created for the demo, they are located in the EHRs/annotated_json_0.8 folder. These were done using -w 10 and -t 0.8 settings of QuickUMLS. I found that when using threshold of 0.7, “illness” was recognised as “stillness” as well as illness, and “right” was recognised as “bright”, and both then appear in the symptoms chart. So I went with 0.8.
    a. Copy that folder to your C drive or inside the search app folder and change $dir variable inside init.php (see next point).
9. Change init.php file inside the xampp/htdocs/searchapp-folder to configure the following:
    a. $index_name – how you want to name your index
    b. $number_of_results_to_show – how many results to show on the home page
    c. $python – location of the python interpreter to run the extractor.py script (your Ananconda environment’s python if in Windows)
    d. $extractor_location – location of the extractor.py file
    e. $dir – location of annotated json files to index (like the ones in point 8).
10. Start Elasticsearch
11. Start QuickUMLS server if not already.
12. Start XAMPP Apacher local server.
13. Run indexer.php to create an index and index prepared json files. Simply open it in your browser (e.g. localhost/searchapp/indexer.php).
14. After indexing is completed, go to index.php in the browser. The search interface should be showing if all goes well.

Miscellaneous:

- Charts use Google Charts. They don’t work without internet.
- The JSON files used for indexing have original_text section. This section is not actually used in the interface. The idea was to use it with free-text search option, but as soon as you use charts as filters, the use of marked_up_text is required for highlighting. So it is kind of obsolete.
- The JSON files used for indexing in the category sections have count for each concept to indicate how many times that concept appeared in the document. This is not used anywhere.
- QuickUMLS removes negation words when analysing ngrams (e.g. not, none, no). So negated concepts are not handled. SNOMED CT has some negated concepts (e.g. no history of depression, no family history of stroke). Switching that off in QuickUMLS should pick up such concepts in plain text.