NLP Project Workflow Proposal

- Parinishtha

AIM: The primary goal of this project is to develop a search engine using NER technology that will find relevant papers from a corpus.

- This will help to narrow down to only the publications/abstracts that contain the keyword searched, rather just having to sift through all.

1. **Create a corpus of scientific abstracts** using the NCBI E-utilities API

   We can use NCBI E-utilities esearch and efetch to download abstracts from PubMed, even do a batch download of all abstracts corresponding to a keyword search. I found a link explaining it with an example:

   https://erilu.github.io/pubmed-abstract-compiler/

   This can be done to get a corpus of abstracts from PubMed. These abstracts can even be stored in "cached data" folder which will make it easier to access data. Also, attention needs to be paid to keep a check on the numbers of requests being made to the server; can be done by introducing a sleep time to limit the number of queries per second. Biological publications can be filtered by limiting the queries to biological terms.

   - **Esearch** provides a list of UIDs matching a text query.
   - **Efetch** returns formatted data records for a list of input UIDs.

   - retmax specifies how many abstracts can be returned using the search (max 10,000 records).
   - db=pubmed specifies that the pubmed database will be searched.
   - retstart parameter can also be used with retmax.
   - retmode parameter lets set the output to xml/json (for Efetch→ xml).
   - Date parameters can also be set using mindate/maxdate or datetype.

2. **Apply NER technology** to tag words within the abstracts with specific keywords

   - BERT based pre-trained model approaches can be used to tag words within abstracts. We can use the spacy en_core_web_sm pretrained model.
   - Hugging face transformer library or bioBERT can be looked into.
   - BioBERT is a domain-specific language representation model pre-trained on large-scale biomedical corpora. So, it can be suitable for our tasks.
   - The data from the first step will then be pre-processed to be useable by the model(s).
   - The training and validation of data should be done over certain no. of epochs, as supported by our PCs (as it could take a very long time to train too, so being mindful of that).

- One or two more pre-trained for models specific for biological data can be tried out and the performance metrics can be compared at the end, choosing the one that works better on our test data. (choose smaller sub-sets to try this out, save time)

3. **Store the abstracts and their tagged metadata** for faster searching

    - Save all above information in cached file or DB, preferably DBs for easier look-up and access due to frequent querying.
    - Data can be returned in form of dictionaries like we did for our assignments.
    - DB columns:
        - PMID of abstract,
        - hyperlink,
        - date of publication,
        - full abstract text,
        - tags and tagged keywords
    - The functions for querying can be changed for have an option for retrieving data corresponding to a certain data range by specifying the start and end date. (the corpus data can be sorted while downloading as per the date of publication/Entrez entry)
    - Use of sqlalchemy to communicate with database.

4. **Create a frontend** which allows one to search the corpus using a keyword or phrase

    - Similar to our assignments, a GUI can be made using Jinja or JavaScript.
    - It should contain a text field to enter the keywords for which the user wants to retrieve the data; a button to submit, a button to download the data, an option to filter data according to date.
    - The DB columns stated above will be returned.

- Unit testing
    - Idea: Tests can be written to check existence of cached data files and/or DB, certain return datatypes, export file formats.

- Packaging
    - Should contain:
        1. Setup.py file
        2. README.md file
        3. Code and API folder
        4. Unit test files folder

- CLI
    - Idea: Get entries in form of dictionary for a query, to run the API.

- Containerization
    - Creating a docker file to bundle the backend and frontend together.