Advanced Programming 2021/2022

Project Specification

Andrea Giovanni Nuzzolese

Input

The input consists of the DisGeNET COVID-19 data collection.
The COVID-19 DisGeNET data collection is the result of applying state-of-the art text mining tools developed by MedBioinformatics solutions to the LitCovid dataset Chen et al., 2020,
to identify mentions of diseases, signs and symptoms. The LitCovid dataset contains a selection of papers referring to Coronavirus 19 disease.
You are provided with two tab-separated files:

- disease_evidences.tsv, which contains a list of diseases identified by text mining scientific articles. Each rows contains the following columns:
    - diseaseid: the identifier of a disease within the data collection;
    - sentence: the sentence the representing the evidence from literature about the disease. The sentence is an HTML annotated text;
    - nsentence: the sentences in each article abstract are numerated. This number represents the number of the sentence in the original abstract;
    - pmid: the identifier of the publication on PubMed reporting the gene or disease mention;
    - disease_name: the name of the disease;
    - disease_type: the DisGeNET disease type. Possible values are disease, phenotype and group.
- gene_evidences.tsv, which contains a list of genes identified by text mining scientific articles. Each rows contains the following columns:
    - geneid: the NCBI Entrez Gene Identifier;
    - sentence: the sentence the representing the evidence from literature about the disease. The sentence is an HTML annotated text;
    - nsentence: the sentences in each article abstract are numerated. This number represents the number of the sentence in the original abstract;
    - pmid: the identifier of the publication on PubMed reporting the gene or disease mention;
    - gene_symbol: the official gene symbol;
    - organism: the organism for the gene (Homo sapiens or Severe acute respiratory syndrome coronavirus 2);

General Goal

Write a program able to:

- parse the two files composing the data collection;
- analyse the data;
- compute relevant outcomes based on (i) arithmetic and statistical operations as well as (ii) the combination of the two files;
- present the user with relevant outcomes.

The program must be designed and implemented in Python by using the Object-Oriented paradigm and by applying its associated fundamental concepts (i.e. encapsulation, inheritance, data abstraction, and polymorphism) properly.
The design must be carried out by drawing CRD cards, class and use case diagrams.
Design and implementation choices must be explained into the final project document.
Additionally, the dataset management and presentation must rely on Pandas (and NumPy) and Flask, respectively.

Detailed Instructions

The program is composed of three main parts, consisting of:

- **[Part 1]** the part with classes and their associated methods for:

    1. reading the files. There are two dataset readers that provides same operations;
    2. providing a registry of analytical operations that can be performed on the dataset. The registry collects a number of different operations. An operation can be any but the registry is agnostic the specific operation type;
    3. coordinating the analytical operations with respect to user inputs received from the user interface (cf. Part 3). Those operations are only coordinated by this part. In fact, the actual execution of such operations is delegated to another part (cf. Part 2).

- **[Part 2]** the part with classes and their associated methods for:

    1. analysing the data by applying a specific operation. Each class creates a snapshot of the dataset, thus it implements a specific operation. The following are the operations that the classes should implement:
        1. recording the numerical metadata consisting of the number of rows and columns of each of the two files composing the data collection;
        2. recording the general semantics of the dataset, i.e. the labels of the columns of each of the two files composing the data collection;
        3. recording the number of different genes detected in literature. The list should be sorted in ascending order;

4. given a gene symbol or its ID, provide the list of sentences that are an evidence in literature about the relation of this gene with COVID-19;
5. recording the number of different disease detected in literature. The list should be sorted in ascending order;
6. given a disease name or its ID, provide the list of sentences that are an evidence in literature about the relation of this disease with COVID-19;
7. recording the 10-top most frequent distinct association between genes and diseases;
8. given a gene symbol or its ID, provide the list of diseases such a gene is associated with.;
9. given a disease name or its ID, provide the list of genes such a disease is associated with;

Again, model the software by identifying and defining classes and their methods.

- **[Part 3]** the part that implements the Web-based user interface (UI). Such a UI provides a list of choices, where each choice enables an analytical objective (cf. Part 2). However, the interaction between the UI and the analytical objectives is always mediated by the software Part 1. The list of choices can be represented as a list of hyperlinks. Each hyperlink is associated with a dedicated view (i.e. a Web page) that shows the outcomes of the analytical operation requested by the user. Again the dedicated view provided by the Web page does not communicate directly with the Part 2, but it communicates with the Part 1. Accordingly the Part 1 is responsible for (i) forwarding the request to the appropriate class/method of Part 2 and (ii) returning the result to the view.

The three parts should be implemented as three separate components, i.e. three Python modules consisting in three separate files.
The software must be described into a project document by using UML diagrams in order to point out what are:

- the use cases;
- the structure of classes.