Module 1: Calculating a Feature Vector from an SDF File

`mol-infer/Cyclic_improved`

February 14, 2021

Contents

1 Introduction

This note serves as a manual and explains the procedures to run Module 1 of the `mol-infer/Cyclic_improved`
The input and output of Module 1 are as follows.

Input: A set $D = \{G_1, G_2, \ldots, G_p\}$ of cyclic chemical graphs.

Output: A set $\mathcal{F}(D) \triangleq \{f(G_1), f(G_2), \ldots, f(G_p)\}$ of feature vectors, such that $f(\cdot)$ is a feature vector of chemical graphs as details in the accompanying article [1].

The output is written to a csv (comma-separated value) file. This csv file is used in Module 2 of the project.

The remainder of this note is organized as follows

- Section 2: Summary of essential terminology, as well, as the file organization in this package.
- Section 3: A short computational example.
- Section 4: Detailed explanations of the program's input and output.

2 Preliminaries

2.1 Terminology

Chemical Graph. A **graph** is an abstract combinatorial construction comprising a set of **nodes** and a set of **edges**, where an edge is an unordered pair of nodes. A **cycle** in a graph is a sequence of nodes such that except for the first and the last node, each node is unique, and there is an edge for each pair of consecutive nodes in the sequence.

A graph where each node is assigned a chemical element (such as carbon, nitrogen, oxygen, etc.) and each edge is assigned a multiplicity between 1 and 4, is called a **chemical graph**.

Descriptor. A **descriptor** is a numerical value that indicates a certain characteristic of a chemical graph. In this project, among others, descriptors include the number of non-hydrogen atoms, the number of atoms in the core of the chemical graph, the core height, etc. For a complete list of descriptors, please refer to the accompanying article [1].

Feature vector. A vector that comprises the numerical values for the descriptors of a chemical graph.

2.2 File Structure

The following set of files accompany this note.

- `Makefile`: A makefile for compiling the programs.
- `cycle_checker.cpp`: Source code written in C++ that for a given chemical compound checks if the chemical graph contains a cycle or not
- `eliminate.py`: A Python script that screens chemical compounds that are not considered under this project, such us inorganic compounds with less than four carbon atoms, that contain charged atoms, etc.
- `fv_ec.cpp`: Source code written in C++ of the mail program of Module 1, calculating a feature vector.
- `fv_proj.cpp`: Source code (C++) of a program that given a feature vector function f calculated over a set D of chemical graphs, and a set D' of chemical graphs that does not necessarily have the same descriptors as D does, calculates the set $\mathcal{F}(D')$ of feature vectors projected onto the domain of f. This is an auxiliary program, and usually not essential to the flow of the entire project.
- Folder `data` Contains sample input and output files used to test and demonstrate the execution of the programs in Module 1. The files in this folder as as follows.

- `sample1.sdf`: An SDF file that contains a single chemical compound. (Please check Section 4 for more details on SDF files.)
- `sample1_eli.sdf`: An SDF file obtained as the output of the Python script `eliminate.py` when invoked on the file `sample1.sdf`. The contents of the files `sample1.sdf` and `sample1_eli.sdf` should be identical.
- `sample1.csv`: Contains a single feature vector constructed from the file `sample1_eli.sdf`.
- `sample2.sdf`: An SDF file that contains data on 175 chemical graphs.
- `sample2_eli.sdf`: An SDF file obtained as the output of the Python script `eliminate.py` when invoked on the file `sample2.sdf`. The contents of the files `sample2.sdf` and `sample2_eli.sdf` should be identical.
- `sample2.csv`: Contains the set of feature vectors constructed from the file `sample2_eli.sdf`.
- `sample1_on_2.csv`: Contains a single feature vector whose values are calculated from the file `sample1_eli.sdf`, however, the dimensions of the vector are projected on the domain of the feature vector obtained from the file `sample2_eli.sdf`.

3 Execution Example

3.1 Validation of the Data

Data on chemical compounds (equivalently, chemical graphs) is stored in a standard SDF file (more information on the structure of SDF files is given in Section 4). Each chemical graph G must satisfy the following conditions

(i) G must contain a cycle;

(ii) G must contain at least four carbon atoms, none of the atoms is allowed to be charged, and each atom must have atomic mass as listed in Section 4.3; and

(iii) Must not include an aromatic edge.

The Python script `eliminate.py` included in Module 1 can be used to remove the graphs that do not satisfy condition (ii).For condition (iii), the user must confirm whether it is satisfied or not on his/her own.

Confirming that a chemical graph contains a cycle. Please use the program compiled from the source file `cycle_checker.cpp` included in Module 1 to confirm whether each chemical graph in a given SDF file contains a cycle.

To compile the program, the included `Makefile` can be used by issuing the following command in the command prompt.

```
$ make CHECKER
```

In case the `make` command is not available on the system, then the program can be compiled in the following way.

```
$ g++ -std=c++11 -Wall -O3 -o CHECKER cycle_checker.cpp
```

In order to check if a given SDF file `input.sdf` contains a chemical graph that does not include a cycle by issuing the following command on the terminal.

```
$ ./CHECKER input.sdf
```

- If all chemical graphs have cycles (i.e., all satisfy (i)), then the program CHECKER does not output any message. In this case, one can go to the next step.
- Otherwise, (i.e., there is a chemical graph that does not satisfy (i)), CID of such a chemical graph is output. Before going to the next step, such a graph must be removed from the SDF file manually.

Elimination of chemical graphs that are out-of-scope. To check whether each chemical graph in a given SDF file satisfies condition (ii) or not, please use the Python script named `eliminate.py`. The script generates a new SDF file that consists of all chemical graphs in the input SDF file that satisfy (ii).

To use `eliminate.py`, execute the following command.

```
$ python eliminate.py input.sdf
```

If the `input.sdf` constains a chemical graph that does not satisfy (ii), the CID is output.

After the execution of `eliminate.py`, a new SDF file `input_eli.sdf` is output. The file consists of all chemical graphs in `input.sdf` that satisfy condition (ii). This means that, if all chemical graphs in `input.sdf` satisfy (ii), `input.sdf` and `input_eli.sdf` are equivalent.

3.2 Calculating a Feature Vector

Please use the program compiled from the source file `fv_ec.cpp` included in Module 1 to calculate feature vectors for an SDF file such that every chemical satisfies conditions (i), (ii) and (iii).

To compile the program, the included `Makefile` can be used by issuing the following command in the command prompt.

```
$ make FV_ec
```

In case the `make` command is not available on the system, then the program can be compiled in the following way.

```
$ g++ -std=c++11 -Wall -O3 -o FV_ec fv_ec.cpp
```

In order to calculate feature vectors from `input_eli.sdf` and to output the result in `output.csv`, issue the following command on the terminal.

```
$ ./FV_ec input_eli.sdf output.csv
```

The program `FV_ec` prints on the terminal instructions on how to provide the arguments and halts if the arguments are not provided appropriately.

3.3 Calculating a Feature Vector from Other SDF (Not Mandatory)

The mapping f that transforms a chemical graph into a feature vector is constructed from a given set D of chemical graphs. To calculate a feature vector of a chemical graph in another set $D' \neq D$ using f, please use the program compiled from `fv_proj.cpp`.

To compile the program, the included `Makefile` can be used by issuing the following command in the command prompt.

```
$ make FV_proj
```

In case the `make` command is not available on the system, then the program can be compiled in the following way.

```
$ g++ -std=c++11 -Wall -O3 -o FV_proj fv_proj.cpp
```

Let `descriptor.csv` be the name of the csv file that is obtained by executing `FV_ec` on the original SDF containing D. That is, the mapping f constructed from D. Let `input.sdf` be the SDF file that contains the data on $D' \neq D$. To calculate $\mathcal{F}(D')$ and obtain the result in `output.csv`, issue the following command on the terminal.

```
$ ./FV_proj descriptor.csv input.sdf output.csv
```

For example, one can run the program in the following way, using the sample files in Module 1.

```
$ ./FV_proj data/sample2.csv data/sample1.sdf data/sample1_on_2.csv
```

It is not mandatory to execute `FV_proj` to proceed to Module 2 and afterwards.

Let us describe an example of when to use `FV_proj`. Suppose that a neural network has been constructed from `descriptor.csv` in Module 2, and the neural network can be used to predict the value of a certain chemical property, say π. When one uses the neural network to predict the value of π for a chemical graph in `input.sdf`, the chemical graph must be converted into a feature vector by the mapping f. The program `FV_proj` can be used for this.

4 Details in the Input and Output of the Program

4.1 Input

The programs in Module 1 use SDF (Structure Data File), a standard format, for input. For the detail of SDF, splease check the following reference:

- `http://help.accelrysonline.com/ulm/onelab/1.0/content/ulm_pdfs/direct/reference/ctfileformats2016.pdf` (accessible on Feb 1, 2021)

4.2 Output

The output is in an original FV (Feature Vector) format, which is just a CSV file so that can be opened by many spreadsheet softwares. The first line shows the components of FV and the following lines show the values for those components of FV. For example, let us have a look at the FV file `sample1.csv` that is obtained by running `FV_ec` for `sample1.sdf`.

```
CID,n,cs,ch,bl_2,ms,dg_co_1,dg_co_2,dg_co_3,dg_co_4,dg_nc_1,\
dg_nc_2,dg_nc_3,dg_nc_4,bd_co_2,bd_co_3,bd_in_2,bd_in_3,\
bd_ex_2,bd_ex_3,ns_co_C3,ns_co_C2,ns_nc_O1,ns_nc_N1,ns_nc_C2,ns_nc_C3,\
ec_co_C2_C3_2,ec_co_C2_C2_1,ec_co_C2_C3_1,ec_co_C2_C2_2,\
ec_in_C2_C3_1,ec_in_C3_C2_1,\
ec_ex_C3_N1_1,ec_ex_C3_C3_1,ec_ex_C3_O1_1,ec_ex_C3_O1_2,nsH
6140,12,6,4,1,128.333,0,5,1,0,3,1,2,0,3,0,0,0,1,0,1,5,2,\
1,1,2,1,2,1,2,1,1,1,1,1,1,11
```

The symbol \ at the end of a line indicates that there is no line break between the two lines. Here is the overview of descriptors. See [1] for details.

- **CID:** Compound ID. In this example (`sample1.sdf`), it is 6140. The molecule is Phenylalanine, which is taken from `https://pubchem.ncbi.nlm.nih.gov/compound/6140`.
- **n:** Number of atoms except for the hydrogen.
- **cs:** Number of atoms in the core.
- **ch:** Core height.
- **bl:** Number of 2-leaves.
- **ms:** Average molecular mass defined by $\mathrm{ms} \triangleq \frac{1}{n}\sum_a \lfloor 10 \cdot \mathrm{mass}(a)$, where $\mathrm{mass}(a)$ represents the mass of an atom a.

- **dg_co_1, . . . , dg_co_4:** Number of atoms in the core such that the degree is 1, 2, 3 and 4, resp.

- **dg_nc_1, . . . , dg_nc_4:** Number of atoms not in the core such that the degree is 1, 2, 3 and 4, resp.

- **bd_co_2, bd_co_3:** Number of double and triple bonds in the core paths, resp.

- **bd_in_2, bd_in_3:** Number of double and triple bonds in the internal paths, resp.

- **bd_ex_2, bd_ex_3:** Number of double and triple bonds in the external paths, resp.

- **ns_co_Xd:** Number of atoms in the core such that the element symbol is X and the degree is d. For example, `ns_co_C3` represents the number of carbon atoms in the core such that the degree is 3.

- **ns_nc_Xd:** Number of atoms not in the core such that the element symbol is X and the degree is d.

- **ec_co_Xx_Yy_2, ec_co_Xx_Yy_3:** Number of double and triple bonds in the core paths such that the end nodes have X and Y as element symbols and the degrees x and y, resp. For example, `ec_co_C2_C3_2` represents the number of double bonds in the core paths such that both end nodes are carbon atoms and have the degrees 2 and 3, resp.

- **ec_in_Xx_Yy_2, ec_in_Xx_Yy_3:** Number of double and triple bonds in the internal paths such that the end nodes have X and Y as element symbols and the degrees x and y, resp.

- **ec_ex_Xx_Yy_2, ec_ex_Xx_Yy_3:** Number of double and triple bonds in the external paths such that the end nodes have X and Y as element symbols and the degrees x and y, resp.

- **nsH:** Number of the hydrogen atoms.

For the descriptors whose names begin with `ns_` and `ec_`, only those appearing the input SDF are written in the output CSV file.

4.3 Attention

The mass of each atom is hard-coded in the program. They are written in the function `init_MassMap()` in `fv_ec.cpp` as follows. If one needs to change values or to add another atoms, edit the source code directly and compile again.

```
M["B"]  = 108;
M["C"]  = 120;
M["O"]  = 160;
M["N"]  = 140;
M["F"]  = 190;
M["Si"] = 280;
M["P"]  = 310;
M["S"]  = 320;
M["Cl"] = 355;
M["V"]  = 510;
M["Br"] = 800;
M["Cd"] = 1124;
M["I"]  = 1270;
M["Hg"] = 2006;
M["Pb"] = 2072;
M["Al"] = 269;
```

References

[1] T. Akutsu and H. Nagamochi. A Novel Method for Inference of Chemical Compounds with Prescribed Topological Substructures Based on Integer Programming. Arxiv preprint, arXiv:2010.09203