Module 1における特徴ベクトル生成の手順

mol-infer/Cyclic_improved

2021 年 2 月 14 日

目 次

1　はじめに

本稿では, 本プロジェクト (`mol-infer/Cyclic_improved`) における Module 1 の手順を解説する. この Module 1 の入力と出力は以下の通りである.

入力: 閉路を持つ化学グラフの集合 $D = \{G_1, G_2, \ldots, G_p\}$.

出力: 特徴ベクトルの集合 $\mathcal{F}(D) \triangleq \{f(G_1), f(G_2), \ldots, f(G_p)\}$. ただし f は化学グラフを特徴ベクトルに変換する関数で, 論文 [1] において提案されたものである.

出力は, 特徴ベクトルが記載された csv ファイルとして与えられる. この csv ファイルは, 本プロジェクトの Module 2 で用いられる.

本稿の構成は以下の通りである.

- 第 2 節: 基本的な用語, およびパッケージのファイル構成の説明.
- 第 3 節: 実行例.
- 第 4 節: プログラムの入出力に関する詳細.

2 準備

2.1 用語の説明

化学グラフ． **節点** (node) の集合と, 節点と節点を結ぶ**辺** (edge) の集合の対を**グラフ** (graph) という. グラフにおける**閉路** (cycle) とは, ある節点を出発し, 辺を次々になぞって得られる節点の系列 (ただし始点と終点以外の節点は二度以上訪れない) のうち, 始点と終点が一致するものをいう.

節点に対する元素 (炭素, 窒素, 酸素など) の割当, 辺に対する多重度 (一般に 1 以上 4 以下の整数) の割当が与えられたグラフを**化学グラフ** (chemical graph) という.

記述子． 化学グラフの文脈における**記述子** (descriptor) とは, 化学グラフの特徴を表す指標をいう. 一般に, 化学グラフは一つの記述子に対して一つの数値を取る. 本プロジェクトで用いられる記述子の例として, 水素を除く原子の数, コアに属する原子の数, コアの高さ, などがある. 詳細は論文 [1] を参照のこと.

特徴ベクトル． 化学グラフと記述子の系列が与えられたとき, その化学グラフが各記述子に対して取る数値を順に並べたベクトルを**特徴ベクトル** (feature vector) という.

2.2 ファイル構成

パッケージに含まれるファイルとその役割は以下の通りである.

- `Makefile`: コンパイルのための makefile.
- `cycle_checker.cpp`: 化学グラフに閉路が存在するか否かを判定するためのプログラムのソースコード (C++).
- `eliminate.py`: 炭素原子の数が 4 未満など, 本プロジェクトが想定していない化学グラフを除去するための Python スクリプト.
- `fv_ec.cpp`: **特徴ベクトル生成を行うメインプログラム**のソースコード (C++).
- `fv_proj.cpp`: 写像 f を構築した D とは異なる化学グラフの集合 D' に対して $\mathcal{F}(D')$ を計算するプログラムのソースコード (C++). 補助的なもので, 実行を必須としない.
- `data` 実行テストのためのデータファイルを含むサブディレクトリ. 中身は以下の通り.
  - `sample1.sdf`: 1 つの化学グラフに関するデータを有する SDF ファイル.
  - `sample1_eli.sdf`: `sample1.sdf` に `eliminate.py` を適用して得られた SDF ファイル. 中身は `sample1.sdf` と同一である.

- sample1.csv: sample1_eli.sdf から生成された特徴ベクトル.
- sample2.sdf: 175 個の化学グラフに関するデータを有する SDF ファイル.
- sample2_eli.sdf: sample2.sdf に eliminate.py を適用して得られた SDF ファイル. 中身は sample2.sdf と同一である.
- sample2.csv: sample2_eli.sdf から生成された特徴ベクトル.
- sample1_on_2.csv: sample1_eli.sdf から生成された特徴ベクトル. ただし sample2_eli.sdf から構築された写像を用いている.

3 実行例

3.1 データの妥当性を確認

化合物データは標準的な SDF ファイルによって与えられなければならない.
また各化学グラフ $G \in D$ は以下の条件を満たす必要がある.

(i) G は閉路を持つ.

(ii) G は 4 つ以上の炭素原子を持つ. また電荷のある原子, 価数が標準値[1]と異なる原子を持たない.

(iii) 芳香辺 (aromatic edge) を含まない.

なお (i), (ii) については, すべての化学グラフが条件を満たすかどうか, パッケージ内のプログラムを用いて判定することができる.

閉路の有無を確認. すべての化学グラフが閉路に関する条件 (i) を満たすどうかを確認するには `cycle_checker.cpp` を用いる.

コンパイルの方法は, `Makefile` が使用可能な環境では,

```
$ make CHECKER
```

とすればよい. そうでなければ

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

とする.

閉路を持たない化学グラフが SDF ファイル `input.sdf` 内に存在するか否かを確認するには,

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

とする.

- すべての化学グラフが閉路を持つ (すなわち (i) を満たす) 場合は, プログラムは何も出力しない. この場合は次に進んで問題ない.
- 一方, 閉路を持たない化学グラフが存在する場合は当該化学グラフの CID が出力される. このような場合, 次に進むには SDF ファイルから手動で当該化学グラフのデータを取り除く必要がある.

[1]第 4.3 節を参照せよ.

対象外の化学グラフを除去. すべての化学グラフが条件 (ii) を満たすか否かを判定し, 満たすもののみを別の SDF ファイルにまとめるには `eliminate.py` を用いる.

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

もし条件 (ii) を満たさない化学グラフがあれば, その化学グラフの CID が出力される.

実行後, 条件 (ii) を満たす化学グラフのみをまとめた `input_eli.sdf` という SDF ファイルが生成される. もしすべての化学グラフが (ii) を満たす場合は, `input.sdf` と `input_eli.sdf` の中身は同一となる.

3.2 特徴ベクトルの生成

すべての化学グラフが上記 (i), (ii), (iii) を満たすような SDF ファイルに対して特徴ベクトルを生成するには `fv_ec.cpp` を用いる.

コンパイルの方法は, `Makefile` が使用可能な環境では,

```
$ make FV_ec
```

とすればよい. そうでなければ

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

とする.

`input_eli.sdf` に対して特徴ベクトルを生成し, その結果を `output.csv` に出力するには,

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

とする. 引数を与えずに実行すれば (もしくは引数が適切に与えられなかった場合), 指定すべき引数が出力されて終了する.

3.3 別の化学グラフデータを特徴ベクトル化 (必須ではない)

本プロジェクトで構成する特徴ベクトル変換関数 f は, 元となる化学グラフデータ D に依存する. 別の化学グラフデータ $D' \neq D$ を, f を用いて特徴ベクトルに変換するには, `fv_proj.cpp` を用いる.

コンパイルの方法は, `Makefile` が使用可能な環境では,

```
$ make FV_proj
```

とすればよい. そうでなければ

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

とする.

化学グラフデータ D を FV_ec によって変換して得られた特徴ベクトルファイルを descriptor.csv とする. すなわち変換写像 f は D から作られたものとする. 一方, D とは別の化学グラフデータ D' が記された SDF ファイルを input.sdf とする. $\mathcal{F}(D')$ を output.csv に出力するには以下を実行する.

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

たとえば, テスト用のファイルに対して以下のように実行することができる.

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

なお FV_proj の実行は, Module 2 以降に進むために必須の操作ではない. FV_proj の用途として, すでに descriptor.csv からニューラルネットワークが作られていて, そのニューラルネットワークを使って, input.sdf に含まれる化学グラフの化学的性質の値を推定したい状況などが考えられる.

4　プログラムの入出力に関する詳細

ここでは特徴ベクトル生成のメインプログラム FV_ec (ソースコードは fv_ec.cpp) の入出力および実行に関する注意を述べる.

4.1　入力

このプログラムは入力のフォーマットとして標準的な SDF (Structure Data File) を使用している. SDF の書式について, 以下を参考資料として挙げておく.

- http://help.accelrysonline.com/ulm/onelab/1.0/content/ulm_pdfs/direct/reference/ctfileformats2016.pdf (2021 年 2 月 1 日 アクセス確認)
- https://www.chem-station.com/blog/2012/04/sdf.html (2021 年 2 月 1 日 アクセス確認)

4.2　出力

本プログラムの出力ファイルは, 独自の FV フォーマットを採用している. このテキストファイルは, カンマで区切った CSV ファイルであり, Excel などの表計算ソフトで開くことができる. 具体的には, 一行目には特徴ベクトルの記述子が, 二行目以降の各行には特徴ベクトルの数値データが記入される. 例として、sample1.sdf に対して FV_ec を実行した結果得られる sample1.csv を示す. それぞれの構成要素はそのあとに説明する.

```
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
```

なお上記のバックスラッシュ \ とそれに続く改行は紙面の都合上挿入したものであり, 実際のファイル内では\も現れなければ改行もされない. 記述子の概要は以下の通りである. 詳しい説明は論文 [1] を参照せよ.

- **CID:** 化合物の識別子 (Compound ID). この例 (`sample1.sdf`) では 6140 だが, これは `https://pubchem.ncbi.nlm.nih.gov/compound/6140` から取得可能なフェニルアラニン (Phenylalanine) である.
- **n:** 水素を除く原子の数.
- **cs:** コアに属する原子の数.
- **ch:** コアの高さ.
- **bl:** 2-leaf の個数.
- **ms:** 平均分子質量 ms $\triangleq \frac{1}{n}\sum_a \lfloor 10 \cdot \mathrm{mass}(a)$, ただし $\mathrm{mass}(a)$ は原子 a の原子量を表す.
- **dg_co_1, . . . , dg_co_4:** コアに属する原子で, 次数が 1,2,3,4 のものの個数.
- **dg_nc_1, . . . , dg_nc_4:** コアに属さない原子で, 次数が 1,2,3,4 のものの個数.
- **bd_co_2, bd_co_3:** コア二重結合およびコア三重結合の数.
- **bd_in_2, bd_in_3:** 内部二重結合および内部三重結合の数.
- **bd_ex_2, bd_ex_3:** 外部二重結合および外部三重結合の数.
- **ns_co_Xd:** コアに属する元素記号 X の原子で, 次数が d のものの個数. たとえば `ns_co_C3` は, コアに属する炭素原子 C で次数が 3 のものの個数を示す.
- **ns_nc_Xd:** コアに属さない元素記号 X の原子で, 次数が d のものの個数.
- **ec_co_Xx_Yy_2, ec_co_Xx_Yy_3:** コア二重結合およびコア三重結合であって (無向辺), 次数 x の原子 X と次数 y の原子 Y を結ぶものの本数. たとえば `ec_co_C2_C3_2` は, 次数 2 の炭素原子 C および次数 3 の炭素原子 C を結ぶコア二重結合の本数を示す.
- **ec_in_Xx_Yy_2, ec_in_Xx_Yy_3:** 内部二重結合および内部三重結合であって (親から子への有向辺), 次数 x の原子 X と次数 y の原子 Y を結ぶものの本数.
- **ec_ex_Xx_Yy_2, ec_ex_Xx_Yy_3:** 外部二重結合および外部三重結合であって (親から子への有向辺), 次数 x の原子 X と次数 y の原子 Y を結ぶものの本数.
- **nsH:** 水素原子の個数.

なお ns, ec で始まる記述子, 、入力 SDF 内の化合物に現れるもののみが FV ファイルに出力される.

4.3 注意

原子の質量は、プログラムの中に記載するハードコード仕様になっている. fv_ec.cpp 内の関数 init_MassMap() で以下のように定められているが, 質量の変更や, ほかの原子が必要な場合には, 編集して再度コンパイルすることで利用できる.

```
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;
```

参考文献

[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