クラスタ構造解析アプリケーション (ver1.0)

概要

python による network clustering 計算用モジュールです。
Map equation (Rosvall 2008, 2010, 2011) もしくはModularity (Clauset 2004)によるクラスタリングを行います。
Search algorithm にはLouvain method (Blondel 2008) とModified Louvain method (Rosvall 2010) を採用しています。
Map equation により2階層・多階層へ、Modularity では2階層へのモジュール分割が可能です。
各ノードへの遷移確率は Standard teleportation、Recorded link teleportation、Unrecorded link teleportation の3手法により算出できます。(Lambiotte 2012)
収束計算には Arnoldi method と Power method の2手法を実装しています。

ファイル構成

- README.md : セットアップ説明ドキュメント
- config.py.sample : 設定ファイルサンプル
- clustering.py : 実行メインスクリプト
- data/ : データ用ディレクトリ
- output_files/ : クラスタリング結果保存用ディレクトリ
- lib/ : 各種計算用pythonモジュール
- compile.sh : 幾つかのモジュールをcython化して計算速度を向上させます(未最適化)
- decompile.sh : cython化されたモジュールをもとのネイティブpythonに戻します

セットアップ

- python version 3.5.2 推奨です (pyenv等を用いて環境を用意すると便利です)

パッケージのインストール

以下のパッケージを pip を用いてインストールして下さい。

```
pip install scipy
pip install numpy
pip install cython
pip install ete3
```

クラスタリング実行方法

config.py.sample から設定ファイル config.py を作成して下さい。

config.py で設定を施した後に以下のコマンドで実行して下さい。

```
python clustering.py
```

出力結果はconfig.pyで指定されたファイルとして出力されます。

config.py

- infile_path : 入力データCSV のパス
- infile_directed_type : 入力データの指向性タイプ
  - 1 : 指向性 (directed)
  - 2 : 無指向性 (undirected)
    - link weigh(w) をwij = wjiとして、directedと同じアルゴリズムを使用しています。
- vertices_file_path : ノード名を定義したファイルへのメインディレクトリ(clustering.pyのある階層)からのパス
- total_nodes : 総ノード数
- outfile_path : 出力CSVファイルのパス
- p_algo_type : 遷移確率の算出方法
  - 1 : Power method
  - 2 : Arnoldi method
- p_conv_threshold : Power methodに於けるPαの収束しきい値 (Rosvall(2010)では 1.0e–15)
- teleport_type : Teleportationのタイプ
  - 1 : standard teleportation
  - 2 : smart recorded teleportation
  - 3 : smart unrecorded teleportation
- tau : τの値
- quality_method : 最適化の方法（ここは任意に追加、変更できます。詳しくは“新規手法の導入方法”を御覧ください）
  - 1 : Map equation
  - 2 : Modularity
- division_type : 解析タイプ
  - 1 : two-level
  - 2 : 階層化
- num_trial : 各階層モジュール毎の分割リトライ回数（論文的には100回ですがnode数に合わせて調整した方が現実的と思われます）

python における設定値（特に調整する必要は無いです）

- threshold_search : サーチアルゴリズムの収束判定に用いられる閾値
- myfloat : float の精度
- seed_var : サーチアルゴリズム内で移動試行されるノードの順番を生成するための乱数シード値。0以外に設定すると再現性の有るクラスタリングを、0に設定すると毎回ランダムなクラスタリングを行います。

入力データフォーマット

計算には下記の2つのデータファイルがdataフォルダ内に必要です。
- 1 ノードのグローバルidと対応する名前を格納したファイル
- 2 ノード間のリンクウェイトを記録したファイル

1. ノードのグローバルidと対応する名前を格納したファイル

ノードのネットワーク全体でのグローバルなidと名前を格納したcsvファイルです。
[id,名前]の順に置きます。

参考

```
1,id_1
2,id_2
3,id_3
...
```

2. ノード間のリンクウェイトを記録したファイル

データフォーマットは遷移の重さで表し、 [source node id, target node id, weight] で表現されています。(Link list format)

参考

```
1,4,1.01234
2,1,0.12413
2,8,0.61241
...
```

サンプルデータについて

dataフォルダ内には予め2種のサンプルデータが用意されています。

- n24.csv, n24_vertices.csv
  総ノード数24のdirected, weightedネットワークです。
- n48.csv n48_vertices.csv
  総ノード数48のdirected, weightedネットワークです。

それぞれのデータは Lancichinetti 2009 のベンチマーク用ネットワーク生成プログラムを用いて生成されたものです。`util/test_network_generation/`内にそのプログラムを入れてあります。実行にはまず同フォルダ内にて、
`make`
して、その後に生成される`benchmark`を実行します。n24フォルダ内に総ノード数24のネットワークを作成するための設定ファイル(`flag_new_n24.dat`)と、ネットワークデータの作成-データ・フォーマットの変換を行うシェルスクリプト(`auto.sh`)を入れました。`auto.sh`を実行すると、dataフォルダ内にnew_n24.csvとnew_n24_vertices.csvが生成されます。
このプログラムは
https://sites.google.com/site/santofortunato/inthepress2
からもダウンロードできます。

実行結果

出力ファイル

tree map format の csv で出力されます。
最初の行はクラスタリングの設定、
2行目は最終的な評価指標値
以降各行は:

階層構造(1階層上のモジュールに対するローカルなモジュールidもしくはノードid), 滞在確率, ノード（モジュール）名, ネットワーク全体でのグローバルなノード（モジュール）id

となっています。

参考

```
#quality_method:,2,division_type,1,teleport_type,1,modified_louvain,False,seed_value,2220
# final quality value: 0.608996362345
1,1,1,0.0279559404785,id_1,1
1,1,2,0.0283209927059,id_2,2
1,1,3,0.0242094009322,id_3,3
1,1,4,0.0254089036423,id_4,4
1,1,5,0.0257358750548,id_11,11
1,1,6,0.0427649978218,id_12,12
```

可視化HTML

最適化の手法 MapEquation と Moduralityの解析結果をモジュールごとに色分けしてネットワークをブラウザで確認することが出来ます。各手法で解析を実行した後に vis_html/vis.html を safari か firefox で開いて下さい。（出力結果を読み込む関係で、chromeやIEではセキュリティの制御により表示することが出来ません。）

新規クラスタリング指標の導入方法

1.quarity.py内の__new__関数内に新たな分岐をつくる

```
def __new__(cls):
    if cf.quality_method == 1: # use map equation for communities' quality estimation
        import mapequation as mp
       new_cls = mp.Map
    elif cf.quality_method == 2: # use modularity for communities' quality estimation
        import modularity as ml
       new_cls = ml.Modularity
        elif cf.quality_method == 3: #新規評価方法に対して、新しい番号をふる。config,py内 quality_methodの値に対応。
                import someNewMethod as sn
                new_cls = sn.someNewMethod
        else:
                print("error: in config.py, undefined number of quality_method was selected.")
                sys.exit(1)
```

2.評価式を実装するモジュール(.py)をlibフォルダ内に作成

実装に求められること:

```
bool check_network_got_better(self, ql_before, ql_after)
```

- クラスタリング評価指標の値(i.e. map equationならcode length)を比較し、向上していればTrue、悪くなっていればFalseを返す。
- ql_before : サーチアルゴリズム中に保存される前回試行までの最も良い評価指標の値
- ql__after : 比較対象となる新たに算出された評価指標の値
- Map equationの場合 ql_before > ql_after のとき、Modularityでは ql_before < ql_afterとなったときにTrueを返す。

```
bool check_network_converged(self, ql_before, ql_after)
```

- サーチアルゴリズムの収束を判定する関数。ql_before、ql_afterに関しては同上。

```
float get_quality_value(self, __modules, w, p_a)
```

- 評価指標の値を計算し、算出された値を返す関数。
- __modules : 試行中のモジュールを其々moduleクラスオブジェクトとして格納したリスト。オブジェクトが含む情報は:
  - self.__module_id : このモジュールのid、多階層化されているときは同親モジュール内でのローカルなidを表す
  - self.__node_id_list : このモジュールの含むノードのid。同親モジュール内でのローカルな値
  - self.__global_d_list : ネットワーク全体でのグローバルなノードid
- w : normalize されたlink weight マトリックス(2次元)。w[i,j]は j->i のlinkを表す。localなノードid (node_id_list)に対応。注: node_idはindex 1から始まるのに対し、wは0から始まる点に注意して下さい。つまり node__id = 1　から 3 のリンクはw[2,0]に格納されています。
- pa : 各ノードへの遷移確率を格納した1次元配列。wと同様にlocalなノードid(node_id_list)に対応。Indexについてwと同様。

の３つの関数が必要です。
実際に実装する場合は、util/someNewMethod.pyにサンプルファイルを用意したのでこれを修正、libフォルダに移動して使用します。

Cythonの適用について

計算速度向上を目指し、実験的にpythonモジュール群の一部をC言語化する方法を適用できます。Cythonと呼ばれるライブラリを使用することによってpythonモジュールをC言語化しコンパイルすることで、コードの実行速度を向上します。メインフォルダの中に入ったcompile.shを実行すると、libフォルダ内の一部の.pyモジュールがコンパイルされます。コンパイル対象となるモジュールの指定はメインフォルダ内py2pyx.sh内で指定されています。コンパイル後、通常と同様に`python clusterng.py`により計算を実行します。ネイティブなpythonに戻す場合はdecompile.shを実行することでコンパイルされた実行ファイルの削除、cython化されたモジュールの再度python化が行われます
。cythonはまだ簡易的な導入しかされていないので、現段階ではあまり速度に目立った差は見られません。しかし今後、pythonモジュール内の配列、関数をcythonに最適化することで100倍のオーダーで高速化が可能です。

変更点

ver 1.0

- MapEquation と Modurarity によるクラスタ構造解析処理実装
- クラスタリング結果可視化処理（仮）実装

references (code内に書かれたrefも以下を参照)

- Blondel 2008: Blondel, Vincent D., et al. “Fast unfolding of communities in large networks.” Journal of statistical mechanics: theory and experiment 2008.10 (2008): P10008.
- Clauset 2004:Clauset, Aaron, Mark EJ Newman, and Cristopher Moore. “Finding community structure in very large networks.” Physical review E 70.6 (2004): 066111.
- Lambiotte 2012: Lambiotte, Renaud, and Martin Rosvall. “Ranking and clustering of nodes in networks with smart teleportation.” Physical Review E 85.5 (2012): 056107.
- Lancichinetti 2009: Lancichinetti, Andrea, and Santo Fortunato. “Community detection algorithms: a comparative analysis.” Physical review E 80.5 (2009): 056117.
- Langville 2011: Langville, Amy N., and Carl D. Meyer. Google’s PageRank and beyond: The science of search engine rankings. Princeton University Press, 2011.
- Rosvall 2008: Rosvall, Martin, and Carl T. Bergstrom. “Maps of random walks on complex networks reveal community structure.” Proceedings of the National Academy of Sciences 105.4 (2008): 1118–1123.
- Rosvall 2010: Rosvall, Martin, Daniel Axelsson, and Carl T. Bergstrom. “The map equation.” The European Physical Journal Special Topics 178.1 (2009): 13–23.
- Rosvall 2011: Rosvall, Martin, and Carl T. Bergstrom. “Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems.” PloS one 6.4 (2011): e18209.