MJA
The Medical Journal of Australia

Appendix

This appendix was part of the submitted manuscript and has been peer reviewed.
It is posted as supplied by the authors.

Appendix to: Tran B, Straka P, Falster MO, et al. Overcoming the data drought: exploring general practice in Australia using network analysis of big data. *Med J Aust* 2018; 209: 68-73. doi: 10.5694/mja17.01236.

Supplemental material for "The network structure of general practice in Australia: analysis of national claims data"

This repository documents our computational work for the article

> The network structure of general practice in Australia: analysis of national claims data.

Workflow:

1. Preprocessing of MBS claims data
2. Generation of patient – GP graphs
3. Fitting hierarchical blockmodels
4. Analysis of the block structure

Files:

- The Python module `MBS_analysis.py` (imported as `mbs`) contains our most important methods.
- The Jupyter Notebook `supplemental.ipynb` shows a minimal example.
- The folder `pbs` contains Unix scripts which were run on the high-performance computing cluster at UNSW Science.

Blockmodelling

The network we consider is bipartite: there are two types of nodes (GPs and patients), and there are only edges connecting GPs with patients. Most previous studies have used unipartite networks (or one-mode networks), which discard patient nodes and connect GP nodes according to some (increasing) function of the number of their shared patients. These "projected" networks contains less information, and make it virtually impossible to identify solo GPs. Identifying GP communities (or Provider Practice Communities -PPCs) is a process of graph partitioning, communities detection or finding clusters in the networks. The commonly used approach has been modularity maximisation, which, however, suffers from the "resolution limit problem", the phenomenon that small communities "escape" the separation effort.

In this analysis, we utilise the hierarchical stochastic blockmodelling approach developed by Tiago Peixoto. In a blockmodel, GPs and patients are clustered into blocks, and two connections, e.g. (GP 1, patient 1) and (GP 2, patient 2), are equally likely if GP1 and GP2 are from the same block of GPs and patient 1 and patient 2 are from the same block of patients. (Connections between nodes are equally likely if source and target are from the same block.) A partition which maximizes the likelihood is deemed optimal; such a partition consists of blocks of GPs which tend to be connected to the same group(s) of patients. The hierarchical approach fits blockmodels at different resolution levels, and thus the resolution limit problem is ameliorated. Moreover, overfitting (i.e. identifying too many blocks) is counteracted via a clever choice of prior probabilities for the distribution of node and edge numbers. From a point of view of information theory, a model is deemed 'best' if it best compresses the data (s. Occam's Razor).

Preprocessing of MBS claims data

`mbs.make_df()` turns the `csv` data into a pandas dataframe.

Generation of patient – GP graphs

`mbs.patient_doctor_graph()` turns the dataframe into a `graph_tool.Graph`

Fitting hierarchical blockmodels

`mbs.blockmodel()` fits a hierarchical blockmodel, using the Python library `graph-tool`. It returns a NestedBlockState object, containing the hierarchical clustering structure. The bipartite structure is preserved, in the sense that on each level, a block contains either only patient nodes or only GP nodes. For each year from 1994–2014 and each region from 1–5, this `mbs.blockmodel()` is run 4 times, and the best fitting model (highest likelihood / lowest entropy) is retained; see `fit-blockmodel.py`. The `minimize_nested_blockmodel_dl()` of `graph-tool` is not parallelizable, and runs on one core with up to 32 GB of memory, for up to 6 hours.

Heuristic for further subdivision

The hierarchical blockmodel is suited to overcome the "resolution limit" problem for community detection, but communities of very small size (1-4) still escape detection. It is likely that sometimes multiple PPCs are lumped into the same block, even at the bottom level of the hierarchy. Hence on the bottom level we extract groups of GPs which are disjoint, in the sense that they do not share any patients. Such disjoint groups within the bottom level blocks we interpret as PPCs, provider practice communities. This is implemented in `mbs.extract_PPCs()`, which adds a layer at the bottom with this final subdivision.

Analysis of the block structure

For the analysis of PPCs, we define the following patient- and GP-properties:

Patient properties

- `upc`: usual provider continuity. This is defined as $\max\{n_i\}/\sum n_i$, where n_i are the numbers of visits of a patient to a distinct *provider*.
- `upppc`: usual PPC continuity. This is defined similarly as `upc`, except that the n_i's denote numbers of visits to distinct *PPCs*.
- `ppd`: patient degree in PPC. For each patient in a PPC-centric graph [^1], calculate their degree and average it within the PPC.

[^1] A PPC-centric graph contains only the GP nodes from that PPC and the patients of each GP.

GP property

- `spf`: For each GP, the fraction of shared patients within a PPC. This fraction is averaged over all GPs in the PPC.

These properties are calculated by the method `mbs.add_props()`. Once calculated, `mbs.get_ppc_stats()` collates these into a pandas dataframe, with one row per PPC.

Plots

A PPC-centric graph is extracted using `mbs.PPCgraph()`, and plotted via `mbs.plot_PPCgraph()`.