BIOINFORMATICS Class – PROJECT part 2 Group 9
Neuroscience application
Brain network study during resting states

Starting point : the purpose of this project is to see the functional connectivity of the brain. It's common thought among scientists that different parts of the brain can be connected together.
One representation about brain activity can be the electricity: this electricity is measured using electrodes. The specific parts of the brain are sections in which electrodes are positioned. it's been provided for 64 "sections", and hereunder "sections" will be called channels.
These representation can be seen by looking to the measured values, and if two channels keep often the same levels of electricity in identical timeslots it is possible to say that those two channels are connected.
From this assumption, it can be possible to create a correlation matrix between channels, then achieving and commenting the results obtained.

General informations: Data are from the EEG Motor Movement/Imagery Dataset.
Each of these information about brain activity have columns with the name for each channel, and rows referred to the electricity (represented in μV) passed through the channel in 6 seconds.

Task 1 Connectivity Graph
1.1 In general for the implementation, it's been needed to import some library in order to make computation over the data.
For example in order to take data from csv and "export" them into our Python script, **networkx** library for computing informations over graphs in order to represent them in a clear way, **connectivipy,** that allowed to estimate how much channels correlation exists .
The first problem was to deal with the two tables (the csv about the measurements of eyes open and eyes closed).
Put the csv into one single dataframe, it's been possible to take data as matrix.
This matrix is the representation of our model, and contains the natural signals of the electricity in each channel that are data to process.
In general, it could be possible to "shuffle" this data n times, obtaining different matrices with same values inside but in different places.
This project only took the original matrix.

The first estimation done was based on the Directed Transfer Function (DTF),

DIRECTED TRANSFER FUNCTION (**DTF**)

$$\vartheta_{ij}(f) = \frac{|H_{ij}(f)|^2}{\sum_{m=1}^{L} |H_{mi}(f)|^2}$$

Where H(f) is the model (i.e. the matrix), calculating the influence of j-th channel on the i-th. This kind of estimation is strong in comparison with the other, the partial directed coherence, because it can isolate the noise that can appear during the measurement of the signal. Thus, it can still predict well the estimation of the propagation of the signal.

Using the connectivipy library, the matrix is been fixed linking channels with matrix rows and columns.
The estimation is been conducted with only two commands:

1) Investigating on the connectivity using DTF;
2) Taking the significances matrix using a significance level of .5 .

It's useful to take a look to the connections using the **plot method** provided by connectivipy (figure1.1connDTF).

The significance values of the DTF estimator is a matrix that shows the correlation between these 64 channels.

In order to show on a graph which are the couples of channels that contains values that have more than the 20% in the set of the connectivity values (Figure1.1GraphDTF).

1.2 PDC estimation :
It is built a new matrix made from the original data.
Fitting values of this two matrices using the Yule-Walker algorithm.
After to get the coefficients of the data, the estimation is been done and the connection is been plotted (Figure1.2ConnPDC).

Managing in the same way the computation of the threshold (network density = 20%). For building the graph, the result is in Figure1.2GraphPDC).

It looks more connected than the DTF graph in the same density level.
1.3 Then, it's been done the same with different levels of density, showing how much graphs looks connected.
And it's quite visible in the python file that as the threshold rises, the density of edges in the graph increases.
1.5 Showing the representation of the graph over only 19 channels, the result shows that the right part is not so much included into the overall connectivity of the brain (Figure1.5).

Task 2 Graph theory indices
In this task we are going to implement the Graph Theory analysis on the outcome graph we got from the first task.
2.1.
We will do this analysis on the Graph resulted from the Direct Transfer Function DTF estimation (G_dtf):
Appendix Figure: 2.1.1 the average clustering coefficient of all nodes in the graph is :0.55
Path length: the global average Path Length is almost 1.7
(*Appendix Figure: 2.1.2)*

The Local Graph Indices:

	Node	Degree	In	Out
24	Af7	73	39	34
21	Fp1	67	36	31
22	Fpz	66	33	33
33	Fz	65	33	32
27	Af4	64	31	33
23	Fp2	61	32	29
26	Afz	58	28	30
20	Cp6	56	28	28
28	Af8	56	30	26
35	F4	55	24	31

We loop over the nodes and get the values of their degrees (*Appendix Figure: 2.1.3)*.then we show the top ten nodes (in term of highest degree) Figure above (*Appendix Figure: 2.1.4)*

2.2.

A **small-world network** in its definition is a type of graphs in which most nodes are not neighbors of one another, but the neighbors of any given node are likely to be neighbors of each other,and most nodes can be reached from every other node by a small number of hops or steps.

The network measures that serves for distinguishing regular or random network from the "small world" organization are: high clustering coefficient and a short path length.

Here we will try to validate our graph measures to check if they meet the small world networks characteristics, and we will create a random small world network using Erdos Renyi algorithm, with the same number of nodes and compare its characteristics to our graph's ones, (*Appendix Figure: 2.2.1)*

```
[502]: # the average degree:
       nodes_df['Degree'].mean()/len(nodes_df)

[502]: 0.39990234375

[506]: erdos = nx.erdos_renyi_graph(64,0.4, directed=True)

[507]: nx.average_clustering(erdos.to_undirected())

[507]: 0.6388776780989694

[508]: nx.average_shortest_path_length(erdos)

[508]: 1.6063988095238095
```

Looking back to our graph clustering coefficient and shortest path values:

CC = 0.55, L = 1.7

We notice that they meet the characteristics of the small world networks.

Now let's use networkx Erdos Renyi algorithm to build a random small world graph: we found that both CC and L values are similar to our graph values.

$\lambda := L/L_r$. and $\gamma := C/C_r$. The first value = 0.87 (close to one).

The second value = 1.06 (slightly bigger than 1)

$\lambda \approx 1$ and $\gamma > 1$ so we can say that our graph is a small world graph.

So we can conclude that our graph is a small world network.

2.3.

By using PDC method of estimation we got the below values: The CC is bigger(0.64) and the shortest path is getting smaller (0.87). (*Appendix Figure: 2.3.1 and 2.3.2)*

2.4.

(the choice of this task is advised in the case of selection of task 1.3).

For this task we will create the different graphs we have got when changing the threshold of density in task 1.3.

```
CC of graph of desnisty 0.01  0.0
CC of graph of desnisty 0.05  0.2487725943378517
CC of graph of desnisty 0.10  0.40452085608108457
CC of graph of desnisty 0.20  0.557310890428274
CC of graph of desnisty 0.30  0.5761395587802903
CC of graph of desnisty 0.50  0.7176182189689552
```

We can notice that the CC starts from 0 when the density is too small (almost no edges), and the grows bigger as the density increases and this is normal as the edges are getting more.

For the shortest path, and because the graphs with small density are not connected ones, we had to calculate

```
L of graph of desnisty 0.01  0.0
L of graph of desnisty 0.05  0.65625
L of graph of desnisty 0.10  1.0886824324324325
L of graph of desnisty 0.20  1.523561507936508
L of graph of desnisty 0.30  1.8095238095238095
L of graph of desnisty 0.50  1.5084325396825398
```

the shortest path for each of their connected component and then calculate the average based on the number of the nodes in each component:

The shortest path value starts also from zero with the 0.01 density, and then grows until density value of 30%, and it decreases for the 50% again.

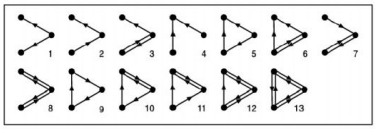
Task 3. Motif analysis

3.1.

For finding the possible 3-nodes motifs we will use our own approach to find them, we calculated the all possible combinations with 3 nodes of the 64 nodes(channels) (*Appendix Figure: 3.1.1)*, then we go determine the possible motifs scenarios of 3-nodes, and go through the graph and compare every 3-nodes combination with all possible 3-node motif scenarios: We have 249984 possible 3-node combinations among our 64 channels.

Now let's consider the possible 3-node motif scenarios:

▶ 13 different isomorphic types of 3-node connected subgraph



We consider a list of 13 item, each one of them refers to the corresponding motif from the figure above (*Appendix Figure: 3.1.2)*.

Our function compares the 3-node combinations to the 13 motif, and if exists it increase the list item related to that motif by one, so here we are counting the presences of all motifs in our graph (*Appendix Figure: 3.1.3)*

```
# the number of the found 3-nodes motifs in the origianl graph:
mo_stats
```

```
[29950, 35526, 9000, 9398, 9398, 5391, 9398, 4966, 9000, 1566, 1566, 866, 435]
```

The resulted list (mo_stats) shows the number of presence of each motif in our graph, now we normalize them to get the concentrations (*Appendix Figure: 3.1.4)*:

```
[119.81, 142.11, 36.0, 37.59, 37.59, 21.57, 37.59, 19.87, 36.0, 6.26, 6.26, 3.46, 1.74]
```

Now in order to check which of the motifs are overrepresented, we create 50 random graphs with the same number of nodes and similar degree, and calculate detect the presence of the motifs in them (*Appendix Figure: 3.1.5)*.

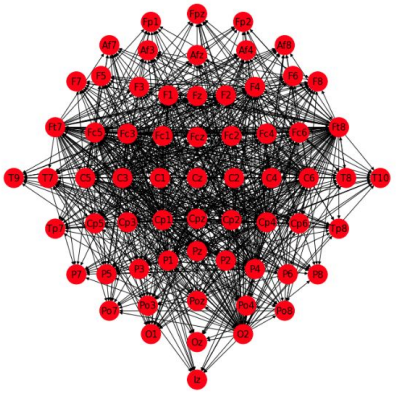
Now we calculate the average presence for each motif in the 50 random graphs and using the below equation we get the statistical representation for each motif in the original graph(*Appendix Figure: 3.1.6)*:

```
[0.37, 0.19, 0.24, -0.16, -0.16, 0.28, -0.15, 1.0, 0.22, 1.0, 1.0, 1.0, 1.0]
```

From the above values we can say that motifs with numbers (09,11,12,13 and maybe 01) are overrepresented.

3.2.

For this task we detect only the motif No.4 (*Appendix Figure: 3.2.1)*and we got the following plot:



Task 4 Community Detection

4.1 Using the Newman algorithm method, the number of communities obtained was 3.

4.2 Each community, when is more than 1 node, looks pretty connected.

References:

Katarzyna J. Blinowska, Aneta Brzezicka, and Jan Kamiński, 2013,
Application of directed transfer function and network formalism for the assessment of functional connectivity in working memory task.
https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.2011.0614
Dominik Krzemiński, Maciej Kamiński (scientific lead), 2015, ConnectiviPy documentation.
https://connectivipy.readthedocs.io/en/latest/conn.html