Ego_networks

November 29, 2021

1 Ego networks

In this notebook, we will build the ego networks using the two dataframes of clean data obtained in the past notebook *clean.ipynb*. In order to achieve this:

- We will use the library *networkx* to transform the dataframe of relationships between the alteri into the different ego networks.
- We will extract some measures of the structure of this ego networks and create with them a dataframe, as our goal is to use them as predictors of the nationality.

1.1 Load the .csv files

We import the different libraries, the usual numpy, pandas, matplotlib and networkx. The methods library that can be seen contains the algorithms to calculate the Dunbar estructure of an individual given an ego network. We will also incorporate this measure in our analysis in order to provide a more complete view. Then we load our .csv files and delete the unformatted columns.

[1]:
```
###Import libraries
from numpy import nan
import pandas as pd
import networkx as nx
from methods import *
import numpy as np
import matplotlib
from matplotlib import pyplot as plt
import warnings
#warnings.filterwarnings('ignore')

###Load .csv files
contactos=pd.read_csv(r'/home/juan/Python/Acculturation/Contactos.
 ↪csv',low_memory=False)
df=pd.read_csv(r'/home/juan/Python/Acculturation/all_data_clean.
 ↪csv',low_memory=False)

###Delete useless columns
del contactos['Unnamed: 0']
del df['Unnamed: 0']
```

1.2 Find out the total number of subjects

In order to begin the calculations, we calculate the total number of subjets. We also transform the datatype of the columns that are going to play a role in the dessign of the ego networks.

[2]:
```
###Change datatypes
contactos['Alter']=contactos['Alter'].astype(int)
contactos['Alter2']=contactos['Alter2'].astype(int)

#Find out the total number of subjects
sujetos = len(contactos['sub/num'].unique())
print("The total number of subjects is {}".format(sujetos))
```

```
The total number of subjects is 473
```

1.3 Creating the ego networks and measuring their properties

At this point, we build the ego networks. Coming from the contacts dataframe, we build a *networkx* graph and we go through all the subjects, one by one, introducing the ties between alteris as links between nodes. And the intensity of their relationships determines the weight associated with the link. Once we have these graphs, we compute different properties: *Average degree, betweenness, closeness, clustering, load centrality, size of the largest component, number of components.*. Apart from these structural measures, we also compute the average intensity with people from their origin country and people from their residence country and their presence (the total number in the ego network). We store all these values in different lists.

[3]:
```
###We create the list in order to store the ego networks and their different␣
 ↪properties.
graficas=[0]*sujetos
avdeg=[0]*sujetos
betw=[0]*sujetos
closs=[0]*sujetos
assort=[0]*sujetos
clustering=[0]*sujetos
load=[0]*sujetos
size=[0]*sujetos
comp=[0]*sujetos
ori=[0]*sujetos
num=[0]*sujetos
res=[0]*sujetos
closnac=[0]*sujetos
closnonac=[0]*sujetos
numnac=[0]*sujetos
mu=[0]*sujetos
numnonac=[0]*sujetos
vect=[[0]*5 for i in range(sujetos)]

###We compute these ego networks and their attributes
```

```python
for k,j in enumerate(contactos["sub/num"].unique()):
    graficas[k]=nx.Graph()
    ###We select the data that corresponds to each subject
    data=contactos[contactos['sub/num']==j]
    if (len(data)>0):
        ###We compute the different number of subjects and their intensity.
        ###This will be used to calculate the Dunbar parameters.
        datamu=df['Clos'][df['sub/num']==j].value_counts()
        for m in range(0,len(datamu)):
            vect[k][datamu.index[m]-1]= datamu[datamu.index[m]]
        ###Building the ego networks
        edges=list(zip(data['Alter'],data['Alter2'],data['Value']))
        graficas[k].add_weighted_edges_from(edges,'Value')
        ###Measuring the networks
        degrees=[val for (node, val) in graficas[k].degree()]
        avdeg[k]=sum(degrees)/len(degrees)
        betw[k]=sum(nx.betweenness_centrality(graficas[k],weight='Value').
 ↪values())/len(nx.betweenness_centrality(graficas[k]).values())
        closs[k]=sum(nx.closeness_centrality(graficas[k]).values())/len(nx.
 ↪closeness_centrality(graficas[k]).values())
        load[k]=sum(nx.load_centrality(graficas[k],weight='Value').values())/
 ↪len(nx.load_centrality(graficas[k]).values())
        assort[k]=nx.
 ↪degree_assortativity_coefficient(graficas[k],weight="Value")
        size[k]= len(max(nx.connected_components(graficas[k]), key=len))/
 ↪len(graficas[k].nodes())
        clustering[k]=nx.average_clustering(graficas[k],weight='Value')

        comp[k]=nx.number_connected_components(graficas[k])
        ego_origin = df['sub/origin'][df['sub/num'] == j].unique()[0]
        ego_residence = df['sub/residence'][df['sub/num'] == j].unique()[0]
        closnac[k]=df['Clos'][(df['sub/num'] == j) & (df['alter/origin'] ==␣
 ↪ego_origin)].mean()
        numnac[k]=df['Clos'][(df['sub/num'] == j) & (df['alter/origin'] ==␣
 ↪ego_origin)].count()
        closnonac[k]=df['Clos'][(df['sub/num'] == j) & (df['alter/origin'] ==␣
 ↪ego_residence)].mean()
        numnonac[k]=df['Clos'][(df['sub/num'] == j) & (df['alter/origin'] ==␣
 ↪ego_residence)].count()
        ori[k]=ego_origin
        num[k]=j
        res[k]=ego_residence
    else: print(j)
```

```
/home/juan/anaconda3/lib/python3.7/site-
packages/networkx/algorithms/assortativity/correlation.py:287: RuntimeWarning:
```

```
invalid value encountered in double_scalars
  return (xy * (M - ab)).sum() / numpy.sqrt(vara * varb)
```

1.4 Create the ego networks dataframe

Coming from the previous lists, we create the networks dataframe, that contains the ego networks and their attributes. At this point, we also compute the Dunbar's parameter μ and add it to our dataframe, with its confidence interval.

[17]:
```
###Creation of the dataframe
redes=pd.DataFrame(data={'Subject origin':ori,'Subject residence':res,'Subject␣
↪ID':num,'Vect':vect,'Graphs':graficas,
                         'Average degree':avdeg,'Betweenness':betw,'Closeness':
↪closs,'Load centrality':load,
                         'Assortativity':assort,'Clustering':clustering,
                         'Number components':comp,'Size largest component':size,
                         'Closeness residence':closnonac,'Closeness origin':
↪closnac,'Number residence':numnonac,
                         'Number origin':numnac})
df.isnull().sum()
redes.fillna(-2,inplace=True)
###Calculating the Dunbar's parameters
redes['Fitted']=redes['Vect'].apply(lambda x:Individual(x).fit_model())
redes[['Mu','Mu-','Mu+']] = pd.DataFrame(redes.Fitted.values.tolist(), index=␣
↪redes.index)
```

```
/home/juan/anaconda3/lib/python3.7/site-packages/scipy/optimize/minpack.py:175:
RuntimeWarning: The number of calls to function has reached maxfev = 400.
  warnings.warn(msg, RuntimeWarning)
/home/juan/Python/Acculturation/methods.py:49: IntegrationWarning: Extremely bad
integrand behavior occurs at some points of the
  integration interval.
  return integrate.quad(integrand_leg,args = (R,L,r), a=0., b=t)[0]
/home/juan/anaconda3/lib/python3.7/site-packages/scipy/optimize/minpack.py:175:
RuntimeWarning: The iteration is not making good progress, as measured by the
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
```

1.5 Some changes in the dataframe

At this point, we make some changes in the ego networks dataframe. We introduce a function that substitutes the confidence interval of the μ by an string that determines the regime. Then we delete the columns associated with the confidence interval, fill the NaN and rename some columns.

[5]:
```
###Creating function to make clearer the regime
def rex(x,y):
    if ((x < 0) & (y < 0)): z='Inverted'
    elif ((x > 0) & (y > 0)): z='Standard'
```

```
    else : z='Unclear'
    return z

###Introducing the former function in our dataframe
regime=[0]*sujetos
for i in range(len(redes)):
    regime[i]=rex(redes['Mu-'][i],redes['Mu+'][i])
redes['Regime']=regime
del redes['Fitted']
del redes['Mu-']
del redes['Mu+']

###Filling NaNs
#redes.fillna(0,inplace=True)

###Renaming columns and getting rid of the networkx graph
redes=redes[['Subject ID','Subject origin','Subject␣
↪residence','Mu','Regime','Average degree','Betweenness',
             'Closeness','Load centrality','Assortativity','Clustering',
             'Number components','Size largest component',
             'Closeness residence','Number residence','Closeness␣
↪origin','Number origin','Graphs']]
del redes['Graphs']
```

1.6 More changes in the networks dataframe and merging

Now we are going to merge the network measures dataframe with the original one *df*, that contains information about egos and alteris. We want to preserve the egos attributes, so we delete all the information about the alteris. We will also introduce the variable *FMIG2* that measures the number of years these migrants have spent in their residence countries. Once we have made all these changes, some typos need to be corrected, and that is what we do at the end of this cell.

[6]:
```
###Deleting the alteri column in df and merging it with the networks dataframe
df.drop(["alter/origin","alter/residence","alter/num"],axis=1,inplace=True)
df.drop([col for col in df.columns if col[0] == "A"],axis=1,inplace=True)
df.rename(columns={'sub/num':'Subject ID','sub/origin':'Subject origin','sub/
↪residence':'Subject residence'},inplace=True)
df = df.drop_duplicates(['Subject origin','Subject ID'])

redes2= pd.merge(redes, df,how='left',on=['Subject ID','Subject␣
↪origin','Subject residence'])
redes2.drop("Subject ID",axis=1,inplace=True)

###Calculating FMIG2
import datetime
FMIG2=[0]*len(redes2)
for i in range(len(redes2)):
```

```
    if (redes2['FMIG'].iloc[i]<0.5):
        FMIG2[i]=0
    elif (redes2['FMIG'].iloc[i]>1500):
        FMIG2[i] = 2005 - redes2['FMIG'].iloc[i]
redes2['FMIG2']=FMIG2
redes2.drop("FMIG",axis=1,inplace=True)

###Renaming columns and changing some typos
redes2.rename(columns={'Subject origin':'Subject_origin','Subject residence':
 ↪'Subject_residence'},inplace=True)
redes2.columns = redes2.columns.str.replace(' ', '_')
redes2.dropna(axis=1,inplace=True)
```

1.7 Save the final results and display a sample

[7]:
```
redes2.to_csv("Redes_2.csv")
```

[8]:
```
redes2.sample(10)
```

[8]:
```
    Subject_origin Subject_residence        Mu    Regime  Average_degree  \
405             do                sp  0.011112   Unclear       31.600000
447             se                sp  0.089195   Unclear       12.533333
348             do                sp  0.503703  Standard       29.954545
422             se                sp -0.325123  Inverted       23.600000
226             ar                sp -0.263139  Inverted        8.454545
380             do                sp  0.066795   Unclear       30.000000
76              do               usa  0.275320  Standard       44.000000
22              do               usa -0.044483   Unclear       44.000000
209             ar                sp -0.215321  Inverted       15.727273
30              do               usa -0.044483   Unclear       44.000000

     Betweenness  Closeness  Load_centrality  Assortativity  Clustering  …  \
405     0.006554   0.801154         0.006554      -0.199161    0.684147  …
447     0.017285   0.590461         0.017212      -0.255307    0.552218  …
348     0.007404   0.780109         0.007404       0.007975    0.801524  …
422     0.016655   0.615225         0.016655       0.499006    0.830791  …
226     0.042399   0.372683         0.042396       0.103047    0.750587  …
380     0.007400   0.767192         0.007400      -0.141919    0.649453  …
76      0.003271   1.000000         0.000070      -0.022727    0.676638  …
22      0.000000   1.000000         0.000000      -0.022727    0.868196  …
209     0.013670   0.574877         0.013668      -0.155336    0.750841  …
30      0.000000   1.000000         0.000000      -0.022727    0.984412  …

      TIEDEN     TIEDC        TIECC      TIEBC  TIECND  TIECCZ      DSET  \
405  0.25253  75.84205     43.82789    9.15293    11.0    0.70       11d
447  0.15758  87.48616     71.56298   51.49595     8.0    0.83      13bs
```

```
348  0.54343  42.39257    44.19432   4.76352    29.0    0.83      l1d
422  0.53636  48.52009    35.27839  11.53614    28.0    0.85      l1s
226  0.15253  86.58537   714.25538  23.80011     1.0    0.71      lba
380  0.35556  67.44186    77.93953  11.68847    12.0    0.78     l3bd
76   0.08384  93.09309    79.60957  30.71690     2.0    0.91  rc2f1de
22   0.62626  39.11205    48.85972   1.49565    17.0    0.93    rcfds
209  0.29293  72.64673  8578.10745  32.64296    12.0    0.83     l5da
30   0.95657   4.54545     6.86379   0.99466     3.0   -0.89    rdfds

    sub/language alter_language  FMIG2
405           es             es    1.0
447           es             es    6.0
348           es             es    1.0
422           es             es    0.0
226           es             es    0.0
380           es             es    2.0
76            en             en    0.0
22            es             es    0.0
209           es             es    3.0
30            es             es    0.0

[10 rows x 106 columns]
```

```
[ ]:
```