Final Course Project

Yueh-Lin Tsai

2024-12-11

Overview

This is Final Course Project

# Import libraries

## R library

library(reticulate)  
#use\_condaenv("smm638", required = TRUE)  
library(car)  
library(sna)  
library(ergm)  
library(dplyr)  
library(tidyr)

## Python library

import pandas as pd  
import networkx as nx  
from networkx.algorithms import bipartite as bp  
import matplotlib.pyplot as plt  
import numpy as np  
import itertools  
from IPython.display import Image  
import matplotlib  
from graph\_tool.all import \*  
import random

# Load data

## HR\_edges.csv

# load data  
fr = pd.read\_csv('/Users/adam/Desktop/Bayes/Class/Network\_Analytics/SMM638/FCP/deezer\_clean\_data/HR\_edges.csv')  
# data preview  
fr.head()

node\_1 node\_2  
0 0 4076  
1 0 29861  
2 0 53717  
3 0 23820  
4 0 39945

## HR\_genres.json

import json  
with open('/Users/adam/Desktop/Bayes/Class/Network\_Analytics/SMM638/FCP/deezer\_clean\_data/HR\_genres.json', 'r') as f:  
 pr\_json = json.load(f)  
pr\_json["11542"]

['Indie Rock', 'Indie Pop/Folk', 'International Pop', 'Rap/Hip Hop', 'Pop', 'Rock', 'Indie Pop', 'Alternative']

### Convert the dictionary into a Pandas

pr = pd.json\_normalize(pr\_json).T  
pr.rename({0: 'genres'}, axis=1, inplace=True)  
pr.head()

genres  
13357 [Pop]  
11542 [Indie Rock, Indie Pop/Folk, International Pop...  
11543 [Dance, Pop, Rock]  
11540 [International Pop, Jazz, Pop]  
11541 [Rap/Hip Hop]

pr\_original = pr

### Separate rows

pr = pr.explode('genres')  
pr.reset\_index(inplace=True)  
pr.rename({'index': 'user\_id'}, axis=1, inplace=True)  
pr.head()

user\_id genres  
0 13357 Pop  
1 11542 Indie Rock  
2 11542 Indie Pop/Folk  
3 11542 International Pop  
4 11542 Rap/Hip Hop

# two-mode (Bipartite) networks

## data botton\_nodes top\_nodes edges

bottom\_nodes = pr["user\_id"].drop\_duplicates().reset\_index(drop=True).to\_numpy()  
top\_nodes = pr["genres"].drop\_duplicates().reset\_index(drop=True).to\_numpy()  
edges = pr[['user\_id', 'genres']].values.tolist()

## Graph creation

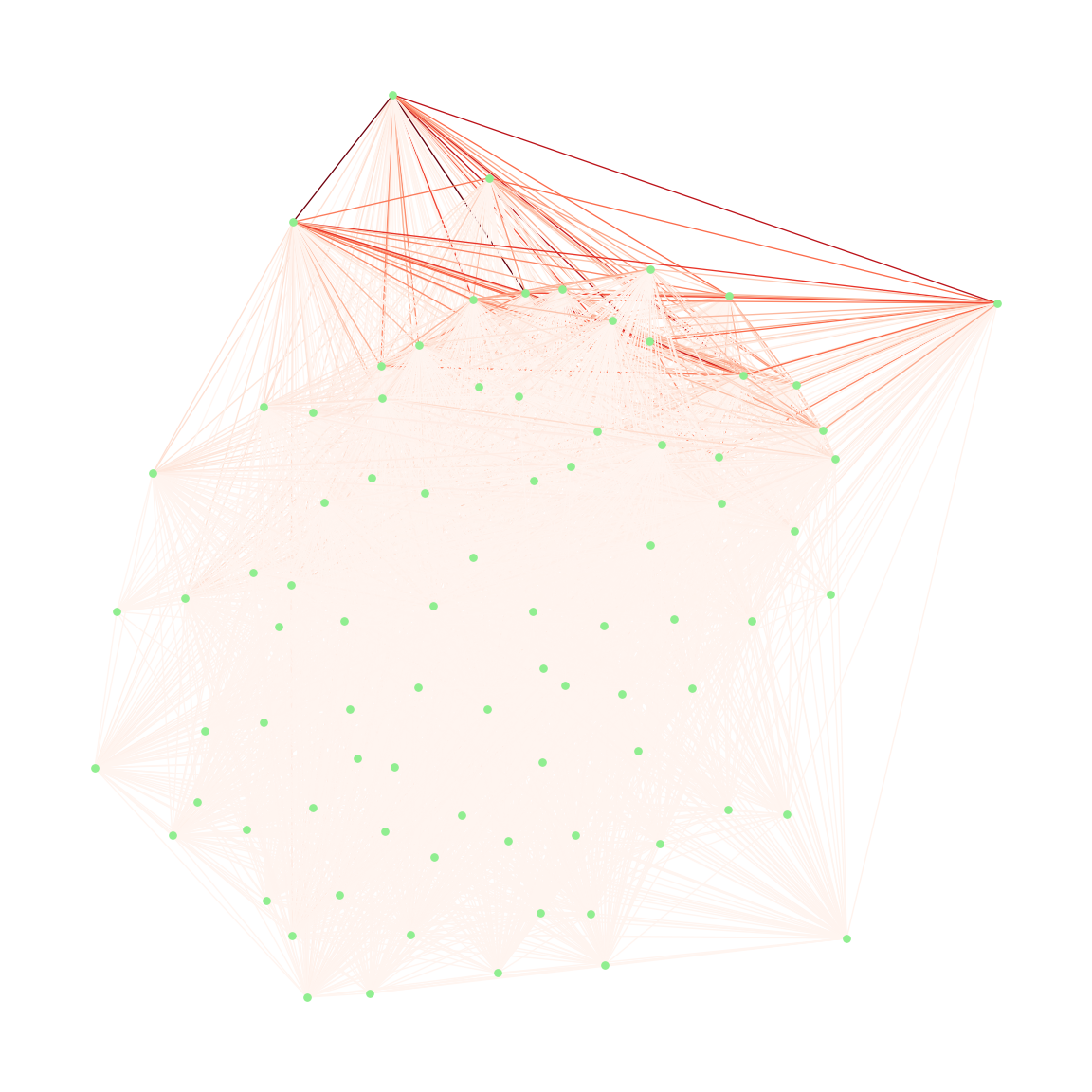
# empty graph  
bg = nx.Graph()  
# add nodes  
bg.add\_nodes\_from(bottom\_nodes, bipartite=0)  
bg.add\_nodes\_from(top\_nodes, bipartite=1)  
# get nx object  
bg.add\_edges\_from(edges)  
# `is bipartite` check  
is\_bip = nx.is\_bipartite(bg)

## Weighted projections of the two-mode networks

#g\_b\_w = bp.weighted\_projected\_graph(bg, bottom\_nodes, ratio=True)  
g\_t\_w = bp.weighted\_projected\_graph(bg, top\_nodes, ratio=True)

## plot two-mode networks network

# Create the figure and specify the size  
plt.figure(figsize=(12, 12))  
  
# Draw the network  
edges = g\_t\_w.edges(data=True)  
weights = [w["weight"] for u, v, w in edges]  
vmin = min(weights)  
vmax = max(weights)  
  
pos = nx.kamada\_kawai\_layout(g\_t\_w)  
nx.draw(  
 g\_t\_w,  
 pos,  
 with\_labels=False,  
 node\_color="lightgreen",  
 node\_size=30,  
 edge\_color=weights,  
 edge\_cmap=plt.cm.Reds,  
 edge\_vmin=vmin,  
 edge\_vmax=vmax,  
)  
  
# Add text annotation if needed  
plt.text(1, 1, "A", fontsize=12, ha="center")  
  
# Show plot  
plt.show()



# Create Z matrix

## create X matrix and Z matrix

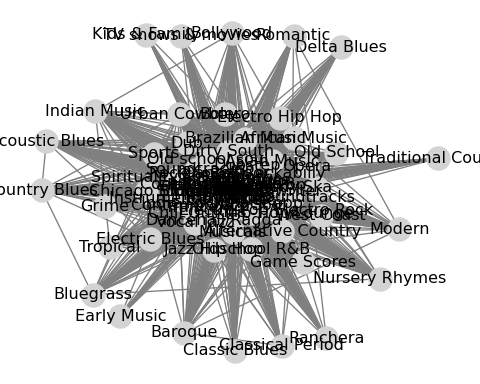
user\_genre\_matrix = pr.pivot\_table(index='user\_id', columns='genres', aggfunc=lambda x: 1, fill\_value=0)  
  
# calculate genre-genre matrix Z  
Z = np.dot(user\_genre\_matrix.T, user\_genre\_matrix)  
# print(Z)  
  
# change ndarray Z to DataFrame  
Z\_df = pd.DataFrame(Z, index=user\_genre\_matrix.columns, columns=user\_genre\_matrix.columns)  
  
Z\_df

genres Acoustic Blues African Music ... Vocal jazz West Coast  
genres ...   
Acoustic Blues 14 2 ... 4 0  
African Music 2 107 ... 9 2  
Alternative 12 80 ... 313 72  
Alternative Country 2 1 ... 14 0  
Asian Music 0 3 ... 4 2  
... ... ... ... ... ...  
Trance 0 8 ... 9 7  
Tropical 0 3 ... 3 0  
Urban Cowboy 0 3 ... 7 1  
Vocal jazz 4 9 ... 374 0  
West Coast 0 2 ... 0 107  
  
[84 rows x 84 columns]

# Community Detection

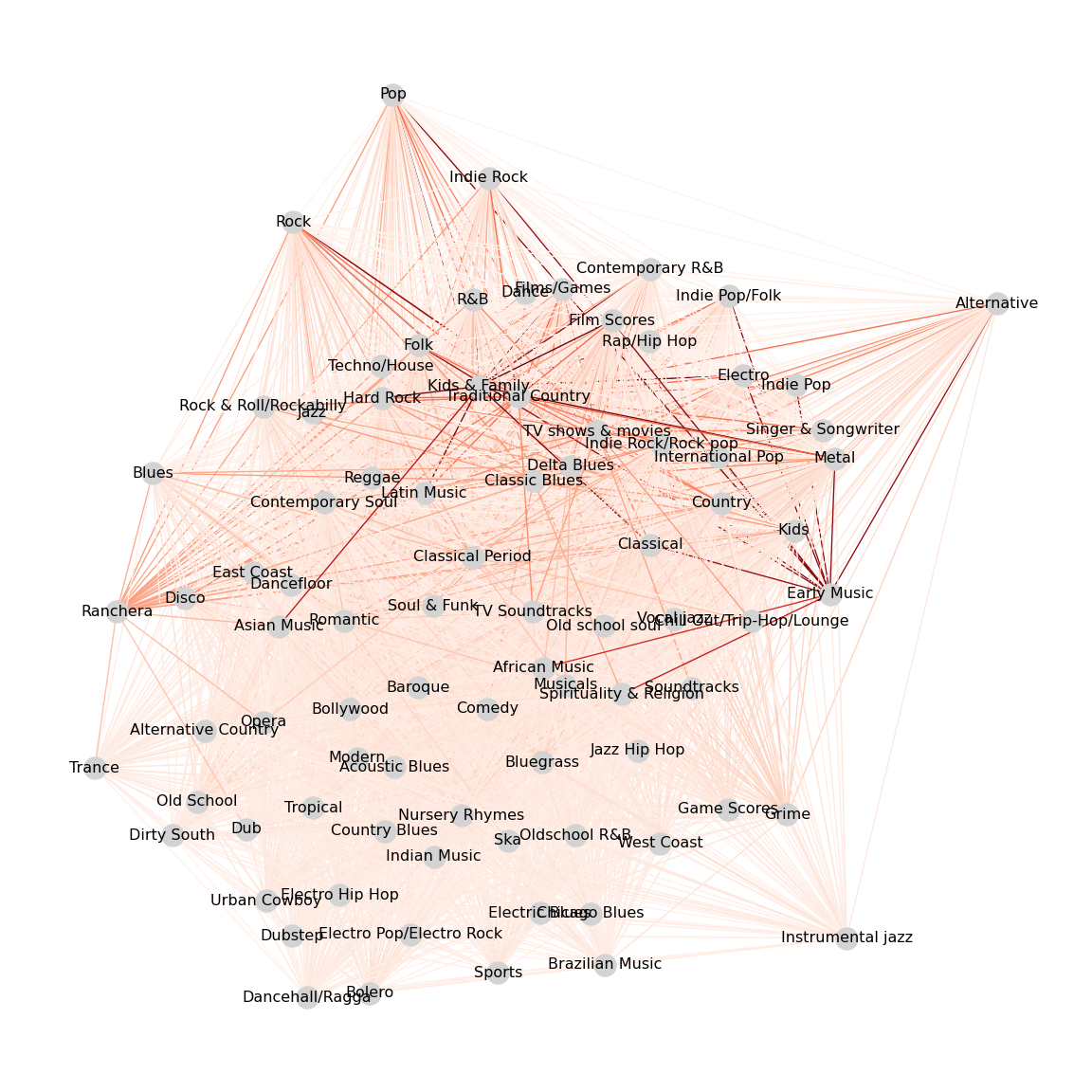
## Visualize the network

# fix node positions for better visualization  
pos = nx.spring\_layout(g\_t\_w, seed=123)  
# draw the network  
nx.draw(  
 g\_t\_w, pos, with\_labels=True, node\_color="lightgray", node\_size=300, edge\_color="gray"  
)



## Community detection using Girvan-Newman’s algorithm

# edge betweenness centrality  
edge\_betweenness = nx.edge\_betweenness\_centrality(g\_t\_w)  
# set the value min and max make color lookable  
vmin = min(edge\_betweenness.values())  
vmax = max(edge\_betweenness.values())  
# network visualization  
pos = nx.kamada\_kawai\_layout(g\_t\_w)  
plt.figure(figsize=(12, 12))  
nx.draw(  
 g\_t\_w,  
 pos,  
 with\_labels=True,  
 node\_color="lightgray",  
 node\_size=300,  
 edgelist=edge\_betweenness.keys(),  
 edge\_color=list(edge\_betweenness.values()),  
 edge\_cmap=plt.cm.Reds,  
 edge\_vmin=vmin,  
 edge\_vmax=vmax,  
)



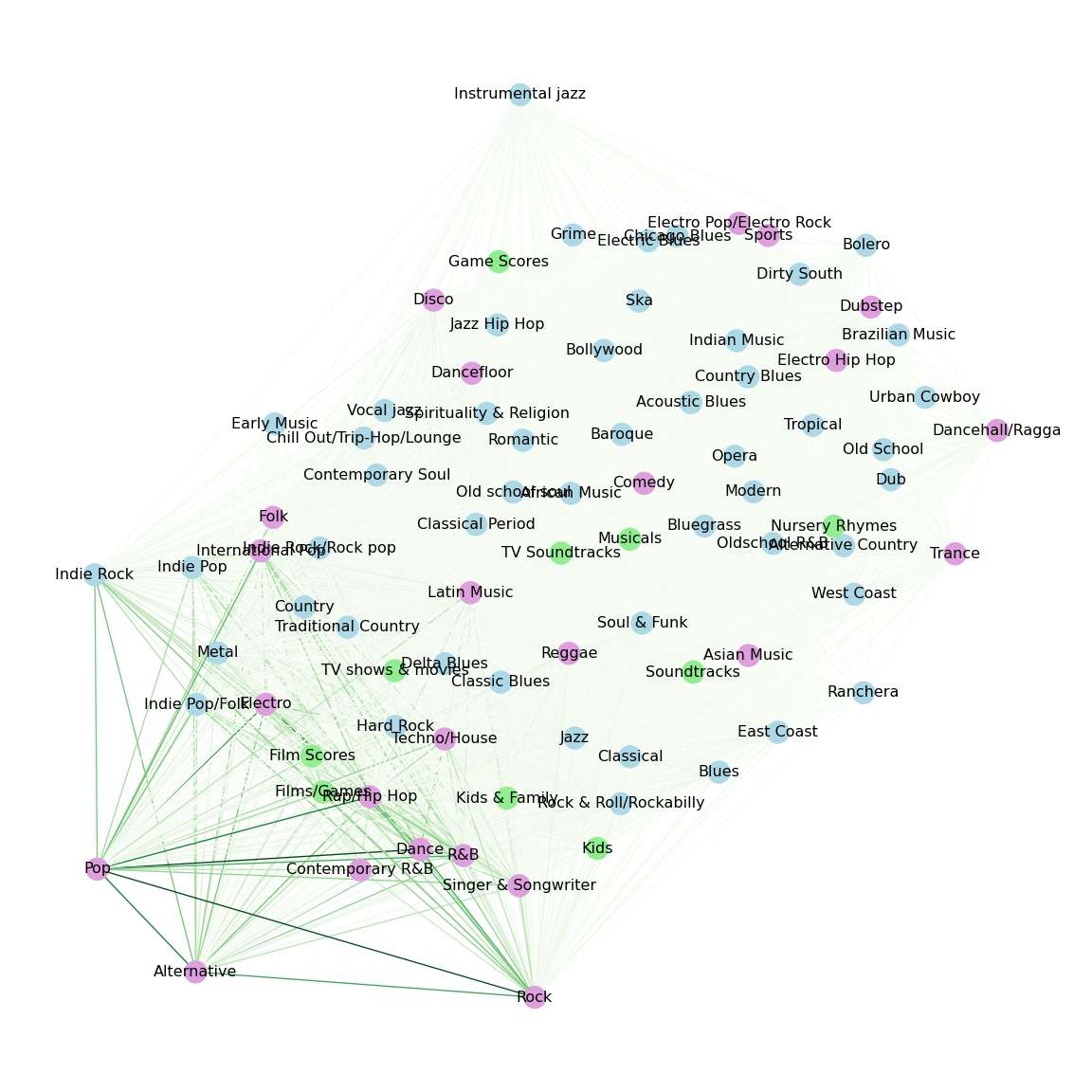
## Community detection using Louvaine’s algorithm (Z matrix)

### Change Z matrix to Graph

g\_z\_matrix = nx.Graph()  
  
# add node  
for node in Z\_df.columns:  
 g\_z\_matrix.add\_node(node)  
  
# add edges  
for i in Z\_df.columns:  
 for j in Z\_df.columns:  
 if i != j and Z\_df.loc[i, j] > 0:  
 g\_z\_matrix.add\_edge(i, j, weight=Z\_df.loc[i, j])

### Fit the Louvain algorithm to the weighted network

# fit the Louvain algorithm to the weighted network  
fit = nx.community.louvain\_communities(g\_z\_matrix, weight="weight")  
# retrieve the communities  
communities = tuple(sorted(c) for c in fit)  
# visualize the network with the identified communities  
colors = [  
 (  
 "plum" if node in communities[0]  
 else "lightgreen" if node in communities[1]  
 else "lightblue"  
 )  
 for node in g\_z\_matrix.nodes  
]  
# visualize the network  
pos = nx.kamada\_kawai\_layout(g\_z\_matrix)  
plt.figure(figsize=(12, 12))  
nx.draw(  
 g\_z\_matrix,  
 pos,  
 with\_labels=True,  
 node\_color=colors,  
 node\_size=300,  
 edge\_color=[g\_z\_matrix[u][v]["weight"] for u, v in g\_z\_matrix.edges],  
 edge\_cmap=plt.cm.Greens,  
 #alpha=0.5,  
)



# Weighted Stochastic Blockmodeling (WSBM)

## Change NetworkX graph into graph-tool graph

# create graph-tool's graph  
g\_z\_matrix\_WSBM = Graph(directed=False)  
  
# add node  
vprops = g\_z\_matrix\_WSBM.new\_vertex\_property("string")  
vertices = {}  
for node in g\_z\_matrix.nodes:  
 v = g\_z\_matrix\_WSBM.add\_vertex()  
 vertices[node] = v  
 vprops[v] = node  
  
g\_z\_matrix\_WSBM.vp["name"] = vprops  
  
# add edge  
eweights = g\_z\_matrix\_WSBM.new\_edge\_property("float")  
for u, v, data in g\_z\_matrix.edges(data=True):  
 edge = g\_z\_matrix\_WSBM.add\_edge(vertices[u], vertices[v])  
 eweights[edge] = data["weight"]  
  
g\_z\_matrix\_WSBM.ep["weight"] = eweights # add weight to graph

## Inferring the Modular Structure of Networks with Weighted Stochastic Blockmodeling

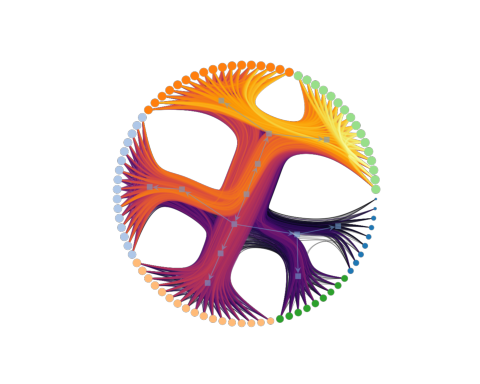
# set random seed  
seed = 41  
random.seed(seed)  
np.random.seed(seed)  
graph\_tool.all.seed\_rng(seed)  
  
# model fit  
state = minimize\_nested\_blockmodel\_dl(  
 g\_z\_matrix\_WSBM, state\_args=dict(recs=[g\_z\_matrix\_WSBM.ep.weight], rec\_types=["real-exponential"])  
)  
# improve solution with merge-split  
for i in range(100):  
 ret = state.multiflip\_mcmc\_sweep(niter=10, beta=np.inf)  
state.draw(  
 edge\_color=prop\_to\_size(g\_z\_matrix\_WSBM.ep.weight, power=1, log=True),  
 ecmap=(matplotlib.cm.inferno, 0.6),  
 eorder=g\_z\_matrix\_WSBM.ep.weight,  
 edge\_pen\_width=prop\_to\_size(g\_z\_matrix\_WSBM.ep.weight, 1, 4, power=1, log=True),  
 edge\_gradient=[],  
 output\_size=(800, 800),  
 output="genre-wsbm.png",  
)

(<VertexPropertyMap object with value type 'vector<double>', for Graph 0x3182515e0, at 0x17fafbb90>, <GraphView object, directed, with 97 vertices and 96 edges, edges filtered by (<EdgePropertyMap object with value type 'bool', for Graph 0x312ef3a40, at 0x17fb6c050>, False), vertices filtered by (<VertexPropertyMap object with value type 'bool', for Graph 0x312ef3a40, at 0x17fb6ecc0>, False), at 0x312ef3a40>, <VertexPropertyMap object with value type 'vector<double>', for Graph 0x312ef3a40, at 0x17fb4c620>)

# show the plot  
#Image(filename="genre-wsbm.png")  
img = matplotlib.image.imread('genre-wsbm.png')  
plt.imshow(img)  
plt.axis('off')

(-0.5, 799.5, 798.5, -0.5)

plt.show()



## Getting a Multi-Level Community Classification

levels = state.get\_bs()  
  
level\_communities = []  
  
for level\_idx, level in enumerate(levels):  
 community\_mapping = {}  
 for v\_idx, community\_id in enumerate(level):  
 if community\_id not in community\_mapping:  
 community\_mapping[community\_id] = []  
 community\_mapping[community\_id].append(g\_z\_matrix\_WSBM.vp["name"][v\_idx])  
   
 level\_communities.append(community\_mapping)  
 print(f"Level {level\_idx} Communities:")  
 for community\_id, nodes in community\_mapping.items():  
 print(f" Community {community\_id}: {nodes}")

Level 0 Communities:  
 Community 61: ['Acoustic Blues', 'Baroque', 'Bluegrass', 'Bollywood', 'Country Blues', 'Indian Music', 'Modern', 'Nursery Rhymes', 'Tropical']  
 Community 46: ['African Music', 'Alternative Country', 'Bolero', 'Chicago Blues', 'Dub', 'Electric Blues', 'Electro Hip Hop', 'Game Scores', 'Grime', 'Jazz Hip Hop', 'Old School', 'Oldschool R&B', 'Opera', 'Ska', 'Sports', 'Urban Cowboy', 'West Coast']  
 Community 70: ['Alternative', 'Contemporary R&B', 'Dance', 'Electro', 'Film Scores', 'Films/Games', 'Indie Pop', 'Indie Pop/Folk', 'Indie Rock', 'International Pop', 'Pop', 'R&B', 'Rap/Hip Hop', 'Rock', 'Singer & Songwriter', 'Techno/House']  
 Community 21: ['Asian Music', 'Brazilian Music', 'Chill Out/Trip-Hop/Lounge', 'Comedy', 'Dancehall/Ragga', 'Dirty South', 'Dubstep', 'Electro Pop/Electro Rock', 'Instrumental jazz', 'Musicals', 'Old school soul', 'Soundtracks', 'Spirituality & Religion', 'TV Soundtracks', 'Trance', 'Vocal jazz']  
 Community 31: ['Blues', 'Classical', 'Contemporary Soul', 'Country', 'Dancefloor', 'Disco', 'East Coast', 'Folk', 'Hard Rock', 'Indie Rock/Rock pop', 'Jazz', 'Kids', 'Latin Music', 'Metal', 'Reggae', 'Rock & Roll/Rockabilly', 'Soul & Funk']  
 Community 0: ['Classic Blues', 'Classical Period', 'Delta Blues', 'Early Music', 'Kids & Family', 'Ranchera', 'Romantic', 'TV shows & movies', 'Traditional Country']  
Level 1 Communities:  
 Community 2: ['Acoustic Blues', 'Baroque', 'Classic Blues', 'Classical', 'Contemporary Soul', 'Hard Rock', 'Indian Music', 'Indie Rock', 'Musicals', 'Nursery Rhymes', 'Opera', 'Pop', 'Rock', 'Singer & Songwriter', 'Soundtracks', 'TV Soundtracks', 'TV shows & movies', 'Tropical']  
 Community 0: ['African Music', 'Alternative', 'Alternative Country', 'Asian Music', 'Bluegrass', 'Bolero', 'Bollywood', 'Brazilian Music', 'Chicago Blues', 'Comedy', 'Contemporary R&B', 'Country Blues', 'Dancefloor', 'Dancehall/Ragga', 'Delta Blues', 'Dirty South', 'Disco', 'Dub', 'East Coast', 'Electric Blues', 'Electro', 'Electro Hip Hop', 'Electro Pop/Electro Rock', 'Film Scores', 'Films/Games', 'Folk', 'Grime', 'Indie Pop', 'Indie Pop/Folk', 'Indie Rock/Rock pop', 'International Pop', 'Jazz Hip Hop', 'Kids', 'Kids & Family', 'Latin Music', 'Metal', 'Old school soul', 'Oldschool R&B', 'R&B', 'Ranchera', 'Rap/Hip Hop', 'Reggae', 'Rock & Roll/Rockabilly', 'Ska', 'Spirituality & Religion', 'Sports', 'Techno/House', 'Traditional Country', 'Trance', 'West Coast']  
 Community 3: ['Blues', 'Chill Out/Trip-Hop/Lounge', 'Country', 'Dance', 'Soul & Funk', 'Urban Cowboy', 'Vocal jazz']  
 Community 6: ['Classical Period']  
 Community 4: ['Dubstep', 'Early Music', 'Game Scores', 'Instrumental jazz', 'Jazz', 'Modern', 'Old School', 'Romantic']  
Level 2 Communities:  
 Community 2: ['Acoustic Blues', 'Baroque']  
 Community 3: ['African Music']  
 Community 0: ['Alternative', 'Alternative Country', 'Asian Music']  
 Community 1: ['Bluegrass']  
Level 3 Communities:  
 Community 0: ['Acoustic Blues', 'Alternative', 'Asian Music']  
 Community 1: ['African Music', 'Alternative Country']  
Level 4 Communities:  
 Community 0: ['Acoustic Blues', 'African Music']  
 Community 1: ['Alternative']  
Level 5 Communities:  
 Community 0: ['Acoustic Blues']  
 Community 1: ['African Music']  
Level 6 Communities:  
 Community 0: ['Acoustic Blues', 'African Music']  
Level 7 Communities:  
 Community 0: ['Acoustic Blues']  
 Community 1: ['African Music']

## Visualisation of WSBM

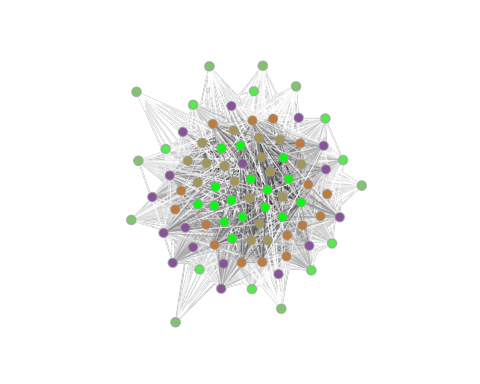
# choose level  
selected\_level = 0  
  
level\_blocks = levels[selected\_level]  
  
# color  
g\_z\_matrix\_WSBM.vp["color"] = g\_z\_matrix\_WSBM.new\_vp("vector<double>")  
color\_map = {community\_id: [np.random.rand(), np.random.rand(), np.random.rand()] for community\_id in set(level\_blocks)}  
for v in g\_z\_matrix\_WSBM.vertices():  
 g\_z\_matrix\_WSBM.vp["color"][v] = color\_map[level\_blocks[int(v)]]  
  
# draw  
graph\_draw(  
 g\_z\_matrix\_WSBM,  
 vertex\_fill\_color=g\_z\_matrix\_WSBM.vp["color"],  
 edge\_color=prop\_to\_size(g\_z\_matrix\_WSBM.ep.weight, power=1, log=True),  
 output\_size=(800, 800),  
 output="level\_0\_visualization.png",  
)

<VertexPropertyMap object with value type 'vector<double>', for Graph 0x3182515e0, at 0x318251400>

#Image(filename="level\_0\_visualization.png")  
img = matplotlib.image.imread('level\_0\_visualization.png')  
plt.imshow(img)  
plt.axis('off')

(-0.5, 720.5, 799.5, -0.5)

plt.show()



# Genres similarity network

## Building Genres similarity Networks

### calculate node\_degree, clustering\_coefficent

g = nx.from\_pandas\_edgelist(fr, source="node\_1", target="node\_2")  
nx.is\_directed(g)

False

nx.is\_weighted(g)

False

# node degree  
g\_node\_degree = nx.degree(g)  
g\_degree\_dict = dict(g\_node\_degree)  
# clustering coefficent  
g\_clustering = nx.clustering(g)

node\_metrics = pd.DataFrame({  
 'user\_id': list(g\_degree\_dict.keys()),  
 'node\_degree': list(g\_degree\_dict.values()),  
 'clustering\_coefficient': [g\_clustering[user\_id] for user\_id in g\_degree\_dict.keys()]  
})  
pr\_new = pr  
pr\_new['user\_id'] = pd.to\_numeric(pr\_new['user\_id'], errors='coerce')  
  
combined\_data = pr\_new.merge(node\_metrics, on='user\_id', how='left')  
  
genres\_attribute\_data = combined\_data.groupby('genres', as\_index=False).agg(  
 avg\_node\_degree=('node\_degree', 'mean'),  
 avg\_clustering\_coefficient=('clustering\_coefficient', 'mean')  
)

### load data from python

fr\_r <- py$fr  
pr\_r <- py$pr  
# Z matrix  
Z\_df <- py$Z\_df  
genres\_attribute\_data\_r <- py$genres\_attribute\_data

### create network from Z matrix

Z\_matrix <- as.matrix(Z\_df)

attribute\_list <- do.call(list, genres\_attribute\_data\_r)

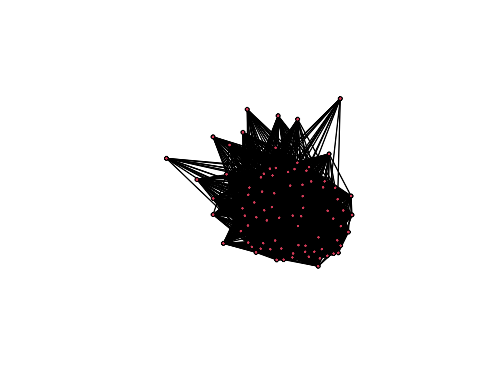
genres\_net <- network(  
 x = Z\_matrix, directed = FALSE,  
 vertex.attr = attribute\_list  
)

genres\_net

Network attributes:  
 vertices = 84   
 directed = FALSE   
 hyper = FALSE   
 loops = FALSE   
 multiple = FALSE   
 bipartite = FALSE   
 total edges= 2779   
 missing edges= 0   
 non-missing edges= 2779   
  
 Vertex attribute names:   
 avg\_clustering\_coefficient avg\_node\_degree genres vertex.names   
  
 Edge attribute names not shown

## Network visualization

plot(genres\_net)



## Descriptive statistics

### density

gden(genres\_net)

[1] 0.7971888

### cug test

cug.test(  
 dat = genres\_net, FUN = "gden", cmode = "size"  
)

Univariate Conditional Uniform Graph Test  
  
Conditioning Method: size   
Graph Type: digraph   
Diagonal Used: FALSE   
Replications: 1000   
  
Observed Value: 0.7971888   
Pr(X>=Obs): 0   
Pr(X<=Obs): 1

### reciprocity

grecip(genres\_net, measure = "dyadic.nonnull")

Mut   
 1

cug.test(  
 dat = genres\_net, FUN = "grecip",  
 FUN.args = list(measure = "dyadic.nonnull"), cmode = "edges"  
)

Univariate Conditional Uniform Graph Test  
  
Conditioning Method: edges   
Graph Type: digraph   
Diagonal Used: FALSE   
Replications: 1000   
  
Observed Value: 1   
Pr(X>=Obs): 0   
Pr(X<=Obs): 1

## ERGM estimation

### Simple ‘edge’ model

mod\_rand <- ergm(formula = genres\_net ~ edges)  
summary(mod\_rand)

Call:  
ergm(formula = genres\_net ~ edges)  
  
Maximum Likelihood Results:  
  
 Estimate Std. Error MCMC % z value Pr(>|z|)   
edges 1.36882 0.04212 0 32.5 <1e-04 \*\*\*  
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
  
 Null Deviance: 4833 on 3486 degrees of freedom  
 Residual Deviance: 3516 on 3485 degrees of freedom  
   
AIC: 3518 BIC: 3524 (Smaller is better. MC Std. Err. = 0)

### Edges & node attributes

mod\_homoph1 <- ergm(genres\_net ~ edges +   
 absdiff("avg\_node\_degree") +  
 absdiff("avg\_clustering\_coefficient")  
)  
summary(mod\_homoph1)

Call:  
ergm(formula = genres\_net ~ edges + absdiff("avg\_node\_degree") +   
 absdiff("avg\_clustering\_coefficient"))  
  
Maximum Likelihood Results:  
  
 Estimate Std. Error MCMC % z value Pr(>|z|)  
edges 2.79217 0.08566 0 32.60 <1e-04  
absdiff.avg\_node\_degree -0.15128 0.01458 0 -10.38 <1e-04  
absdiff.avg\_clustering\_coefficient -16.00433 1.43762 0 -11.13 <1e-04  
   
edges \*\*\*  
absdiff.avg\_node\_degree \*\*\*  
absdiff.avg\_clustering\_coefficient \*\*\*  
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
  
 Null Deviance: 4833 on 3486 degrees of freedom  
 Residual Deviance: 2926 on 3483 degrees of freedom  
   
AIC: 2932 BIC: 2950 (Smaller is better. MC Std. Err. = 0)

mod\_homoph2 <- ergm(genres\_net ~ edges +   
 absdiff("avg\_node\_degree") +  
 absdiff("avg\_clustering\_coefficient") +  
 nodecov("avg\_node\_degree") +  
 nodecov("avg\_clustering\_coefficient"))  
summary(mod\_homoph2)

Call:  
ergm(formula = genres\_net ~ edges + absdiff("avg\_node\_degree") +   
 absdiff("avg\_clustering\_coefficient") + nodecov("avg\_node\_degree") +   
 nodecov("avg\_clustering\_coefficient"))  
  
Maximum Likelihood Results:  
  
 Estimate Std. Error MCMC % z value  
edges 0.587854 0.347280 0 1.693  
absdiff.avg\_node\_degree -0.091987 0.016128 0 -5.704  
absdiff.avg\_clustering\_coefficient -14.339354 1.523784 0 -9.410  
nodecov.avg\_node\_degree -0.019085 0.008942 0 -2.134  
nodecov.avg\_clustering\_coefficient 11.528102 1.068248 0 10.792  
 Pr(>|z|)   
edges 0.0905 .   
absdiff.avg\_node\_degree <1e-04 \*\*\*  
absdiff.avg\_clustering\_coefficient <1e-04 \*\*\*  
nodecov.avg\_node\_degree 0.0328 \*   
nodecov.avg\_clustering\_coefficient <1e-04 \*\*\*  
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
  
 Null Deviance: 4833 on 3486 degrees of freedom  
 Residual Deviance: 2794 on 3481 degrees of freedom  
   
AIC: 2804 BIC: 2835 (Smaller is better. MC Std. Err. = 0)