**CODE**:

import random

import numpy as np

import pandas as pd

import networkx as nx

import matplotlib.pyplot as plt

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

from networkx.algorithms import community

from networkx.algorithms.community.quality import modularity

random.seed(0)

np.random.seed(0)

# ---------------------------

# 1) Synthetic data generator

# ---------------------------

def generate\_synthetic\_interactions(n\_users=80, p\_connect=0.06, avg\_messages=5):

"""

Generate synthetic interactions with text.

Returns a DataFrame with columns: source, target, text

"""

# create an underlying graph to drive interactions

G = nx.erdos\_renyi\_graph(n\_users, p\_connect, seed=1)

users = list(G.nodes())

rows = []

topics = [

("sports", ["team", "win", "match", "score"]),

("politics", ["vote", "policy", "election", "government"]),

("health", ["vaccine", "health", "doctor", "clinic"]),

("tech", ["app", "phone", "update", "feature"])

]

# assign each user to a "leaning" topic to create homophily

user\_topic = {u: random.choice(topics)[0] for u in users}

# produce messages: if user interacts with neighbor, produce a text biased to their topic

for u in users:

neighbors = list(G.neighbors(u))

n\_msgs = max(1, int(np.random.poisson(avg\_messages)))

for \_ in range(n\_msgs):

if neighbors and random.random() < 0.8:

v = random.choice(neighbors)

else:

v = random.choice(users)

topic = user\_topic[u]

# generate a short synthetic sentence with topic terms

word\_pool = []

for t, kw in topics:

if t == topic:

word\_pool += kw \* 5 # overweight own topic

else:

word\_pool += kw

text = " ".join(random.choices(word\_pool, k=6))

rows.append((u, v, text))

df = pd.DataFrame(rows, columns=["source", "target", "text"])

return df

# ---------------------------

# 2) Build user-interaction graph

# ---------------------------

def build\_interaction\_graph(df, weight\_by\_count=True):

"""

Build directed (or undirected) graph from interactions DataFrame.

We'll use undirected weighted graph for community detection.

"""

G = nx.Graph()

for \_, row in df.iterrows():

u, v = int(row["source"]), int(row["target"])

if u == v:

continue

if G.has\_edge(u, v):

G[u][v]["weight"] += 1

else:

G.add\_edge(u, v, weight=1)

return G

# ---------------------------

# 3) Sentiment scoring (VADER)

# ---------------------------

def compute\_user\_sentiment(df):

"""

Compute average sentiment per user based on outgoing message texts.

Returns dict: user -> avg\_sentiment (compound)

"""

analyzer = SentimentIntensityAnalyzer()

user\_scores = {}

grouped = df.groupby("source")["text"].apply(list)

for u, texts in grouped.items():

scores = []

for t in texts:

s = analyzer.polarity\_scores(t)["compound"]

scores.append(s)

user\_scores[int(u)] = float(np.mean(scores)) if scores else 0.0

return user\_scores

# ---------------------------

# 4) Community detection & echo chamber identification

# ---------------------------

def detect\_communities\_and\_echo\_chambers(G, user\_sentiment, echo\_modularity\_thresh=0.4, sentiment\_var\_thresh=0.08):

"""

Detect communities using greedy modularity. For each community compute:

- modularity contribution (we use global modularity but threshold on global value)

- intra-community sentiment variance

Flag community as echo chamber if:

- global modularity >= echo\_modularity\_thresh (indicates strong clustering)

- intra-community sentiment variance <= sentiment\_var\_thresh (homogeneous sentiment)

Returns communities list and set of echo chamber community indices.

"""

communities = list(community.greedy\_modularity\_communities(G, weight="weight"))

# compute global modularity for the partition

partition = [set(c) for c in communities]

Q = modularity(G, partition, weight="weight")

echo\_flags = []

intra\_vars = []

for i, comm in enumerate(communities):

sentiments = [user\_sentiment.get(u, 0.0) for u in comm]

var = float(np.var(sentiments)) if sentiments else 0.0

intra\_vars.append(var)

# community-level condition: low var; overall modularity used for structural strength

is\_echo = (Q >= echo\_modularity\_thresh) and (var <= sentiment\_var\_thresh)

echo\_flags.append(bool(is\_echo))

return communities, Q, intra\_vars, echo\_flags

# ---------------------------

# 5) Visualization

# ---------------------------

def plot\_graph\_with\_communities(G, communities, echo\_flags, user\_sentiment):

pos = nx.spring\_layout(G, seed=2)

# color map for communities

cmap = plt.get\_cmap("tab10")

node\_color = {}

for idx, comm in enumerate(communities):

color = cmap(idx % 10)

for n in comm:

node\_color[n] = color

colors = [node\_color.get(n, (0.5,0.5,0.5)) for n in G.nodes()]

plt.figure(figsize=(10, 8))

nx.draw\_networkx\_edges(G, pos, alpha=0.3)

nx.draw\_networkx\_nodes(G, pos, node\_color=colors, node\_size=200)

# highlight echo chambers by drawing thicker boundaries around community nodes

for idx, comm in enumerate(communities):

if echo\_flags[idx]:

# draw circle around community centroid

xs = [pos[n][0] for n in comm]

ys = [pos[n][1] for n in comm]

cx, cy = np.mean(xs), np.mean(ys)

size = max(0.12, 0.02 \* len(comm))

circle = plt.Circle((cx, cy), size, fill=False, edgecolor='k', linewidth=2.2, linestyle='--')

plt.gca().add\_patch(circle)

plt.text(cx, cy+size+0.02, f"Echo #{idx+1}", ha='center', fontsize=9, fontweight='bold')

plt.title("User Interaction Graph — colored by detected community\nDashed circles highlight detected Echo Chambers")

plt.axis('off')

plt.show()

# ---------------------------

# 6) Main pipeline

# ---------------------------

def main(use\_synthetic=True, csv\_path=None):

if use\_synthetic:

print("Generating synthetic interaction data...")

df = generate\_synthetic\_interactions(n\_users=120, p\_connect=0.05, avg\_messages=6)

else:

if csv\_path is None:

raise ValueError("csv\_path required when use\_synthetic=False")

df = pd.read\_csv(csv\_path)

# expect columns: source,target,text

print(f"Interactions: {len(df)} rows; unique users: {len(pd.concat([df.source, df.target]).unique())}")

G = build\_interaction\_graph(df)

print(f"Built graph: {G.number\_of\_nodes()} nodes, {G.number\_of\_edges()} edges")

user\_sentiment = compute\_user\_sentiment(df)

print("Computed user sentiment for", len(user\_sentiment), "users (compound VADER scores).")

communities, Q, intra\_vars, echo\_flags = detect\_communities\_and\_echo\_chambers(

G, user\_sentiment, echo\_modularity\_thresh=0.35, sentiment\_var\_thresh=0.06

)

print(f"Detected {len(communities)} communities. Global modularity Q = {Q:.3f}")

for i, comm in enumerate(communities):

print(f" Community {i+1}: size={len(comm)}, sentiment\_var={intra\_vars[i]:.4f}, echo\_flag={echo\_flags[i]}")

plot\_graph\_with\_communities(G, communities, echo\_flags, user\_sentiment)

if \_\_name\_\_ == "\_\_main\_\_":

main(use\_synthetic=True)