CS 5001 – Applied Social Network Analysis

Bipartite Graphs (continued)

Here's a more complex example (implemented in Python)

We want to **understand (and visualize) the relationship between trauma types** by constructing a graph

Part of this analysis will involve **bipartite graphs**

- We have a dataset (in **csv** format) produced by the Boston Justice Resource Institute:

  19 columns each representing a trauma type

  SEXUAL_ABUSE, SEXUAL_ASSAULT, PHYSICAL_ABUSE, PHYSICAL_ASSAULT, PSYC_MALTX, NEGLECT, DOMESTIC_VIOLENCE, WAR, WAR_NOT_US, MEDICAL_TRAUMA, INJURY_ACCIDENT, NATURAL_DISASTER, KIDNAP, TRAUMTIC_LOSS, FORCED_DISPLACEMENT, IMPAIRED_CAREGIVER, EXT_INTERPER_VIOLENCE, COMMUNITY_VIOLENCE, SCHOOL_VIOLENCE

  Value of **1** if patient had that trauma; otherwise, **0**

  618 rows, each representing one patient

  No identifying info about a patient (i.e., ID, name, gender, age, etc.)

- We want to analyze the data using **4 methods**:

  Hamming similarity: # of equal components in 2 vectors divided by length of the vectors

  Ex: HammingSim((0,1,0,1,1), (1,0,0,1,0)) = 2/5 = 0.4

  Cosine similarity: for vectors x and y, computed as $(x \cdot y) / (\|x\| \, \|y\|)$, where $\|x\| = \mathrm{sqrt}(x_1^2 + x_2^2 + \ldots + x_p^2)$ for $x = (x_1, x_2, \ldots, x_p)$

  Ex: CosineSim((0,1,0,1,1), (1,0,0,1,0)) = 1/2.447 = 0.409

Pearson correlation: for vectors x and y, computed as

$$r = \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i (x_i - \overline{x})^2}\sqrt{\sum_i (y_i - \overline{y})^2}}$$

If **r = 1**, then perfect **positive correlation**; if **r = -1**, then perfect **negative correlation**.

The nearer the value of **r is to 0 (including negative)**, the **weaker** is the relationship between x and y.

Ex: x = (0,1,0,1,1), y = (1,0,0,1,0)

Sum of x values = 3, mean of x values = 0.6, sum of (x values - means)^2 = 1.2

Sum of y values = 2, mean of y values = 0.4, sum of (y values - means)^2 = 1.2

Numerator = -0.2

r = -0.2 / sqrt(1.2 * 1.2) = -0.1667

See https://www.socscistatistics.com/tests/pearson/ for calculator

Generalized similarity: similar to how Pearson correlation is computed except that:

Compute several iterations starting as O_t, where initially t = 2; repeat until process converges (i.e., until $|O_{t+1} - O_t| < \epsilon$, where ϵ is a pre-defined convergence threshold)

Multiply numerator and each of the 2 terms in denominator by O_{t-1}; O_1 is the identity matrix

For more information, see "A Generalized Model of Relational Similarity", Balazs Kovacs, Social Networks 32, 2010, pp. 197-211 and/or https://github.com/dzinoviev/generalizedsimilarity/blob/master/generalized.py

- First, we'll make **a list of the entries in the data matrix that have traumas** (ignore the 0 entries); we'll keep the entries as **(row #, Trauma)**

- **Make a graph from those entries**: there's 2 "groups" of nodes (row #'s and Trauma's) and edges between them, so it's **bipartite**!

- **Determine similarity between trauma types**:

  Each **column** in the data matrix (i.e., our bipartite adjacency matrix) represents the **vector for a trauma type**

  Want to **compare pairs of columns** using each of our 4 similarity metrics (i.e., 2 trauma types are similar if their vectors are similar)

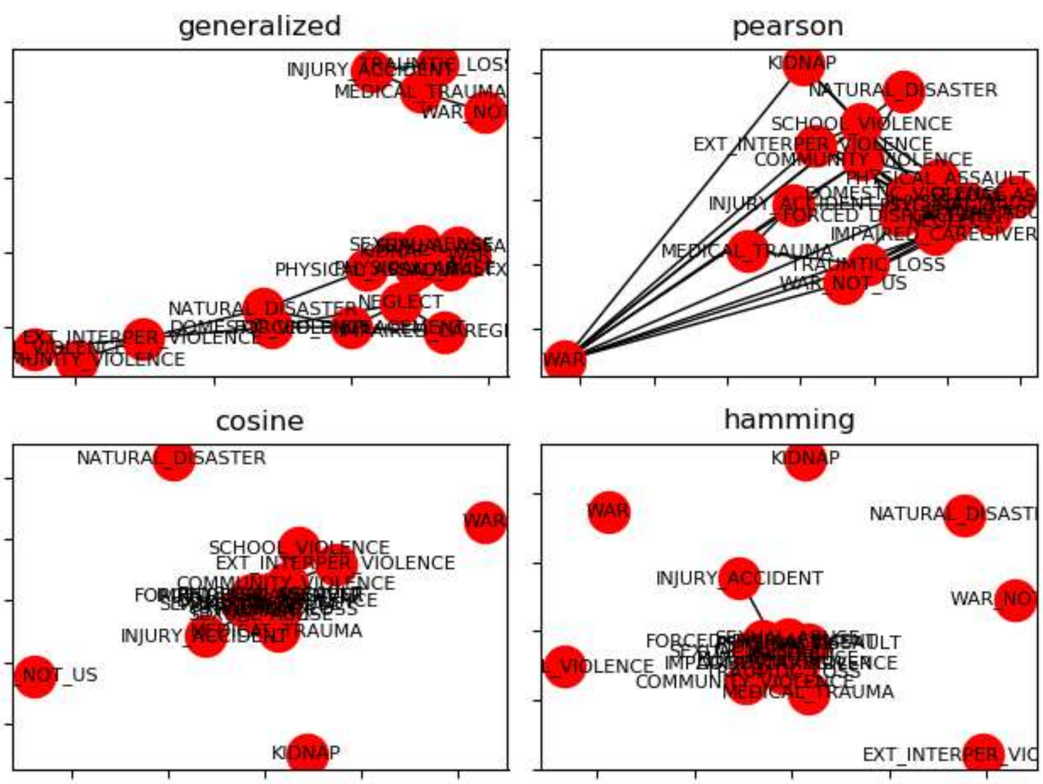
  There are 19 columns so there will be 18+17+16+…+1 comparisons for each metric; we'll put the results for each similarity metric in a **19x19 similarity matrix** (e.g., Sim[PHYSICAL_ABUSE][COMMUNITY_VIOLENCE]); note that the diagonal values will be 0

- **From each similarity matrix, we'll build a graph**:

  To make the size a little smaller, we'll only take the highest 50% of similarity values in a matrix

  We'll make a **list of entries** that each look like **(Trauma$_1$, Trauma$_2$, SimilarityValue)**

  Make a **weighted (undirected) graph** from those entries



Python Code

```
# Analyze trauma data using 4 similarity metrics
# and display graphs showing similarity between traumas
# exec(open("bipartite_trauma.py").read())

# NOTE: You need to install python-louvain
import pandas as pd
import numpy as np
import networkx as nx
from networkx.algorithms.bipartite import sets, weighted_projected_graph
import scipy.spatial.distance as dist
from scipy.stats import pearsonr
import community

import matplotlib.pyplot as plt

# Read in the data (19 columns, 618 rows)
matrix = pd.read_csv("jri_data.csv")
print(matrix.columns, matrix.shape)

# Make a multi-index (as a pandas.core.series.Series) of patients+traumas
# Length is 11742; each entry has a Trauma and 0/1
stacked = matrix.stack()

# Select the patients who _have_ traumas
# Each entry in edges list will have index from index (0..617) and a Trauma
edges = stacked[stacked > 0].index.tolist()

# Make a graph
patients_traumas = nx.Graph(edges)
print(nx.is_bipartite(patients_traumas))   # it is!

l, r = nx.bipartite.sets(patients_traumas)

# Not a pretty display
pos = nx.spring_layout(patients_traumas, scale=100)
plt.figure(figsize=(50,50))
nx.draw_networkx(patients_traumas, pos=pos, with_labels=True, font_size=12)
plt.axis('off')
plt.show()
```

```
# Convert a bi-adjacency matrix to a similarity matrix,
# based on the distance measure
def similarity_mtx(biadj_mtx, similarity_f):
   similarity = [[similarity_f(biadj_mtx[x], biadj_mtx[y])
               for x in biadj_mtx] for y in biadj_mtx]
   # Discard the main diagonal of ones
   similarity_nodiag = similarity * (1 - np.eye(biadj_mtx.shape[1]))
   similarity_df = pd.DataFrame(similarity_nodiag,
                        index=biadj_mtx.columns,
                        columns=biadj_mtx.columns)
   return similarity_df


# Convert a similarity to a sliced similarity network
# (i.e., make a graph out of the highest percentage of
# similarity values; that percentage is specified by density)
def similarity_net(sim_mtx, threshold=None, density=None):
   stacked = sim_mtx.stack()
   if threshold is not None:
      stacked = stacked[stacked >= threshold]
   else:
      count = int(sim_mtx.shape[0] * (sim_mtx.shape[0] - 1) * density)
      stacked = stacked.sort_values(ascending=False)[:count]
   edges = stacked.reset_index()
   edges.columns = "source", "target", "weight"
   network = nx.from_pandas_edgelist(edges, *edges.columns)
   # Some nodes may be isolated; they have no incident edges
   network.add_nodes_from(sim_mtx.columns)
   return network

# This represents the percentage of the highest similarity
# values we'll pull out to display
DENSITY = 0.5

def cosine_sim(x, y):
   return 1 - dist.cosine(x, y)

cosine_mtx = similarity_mtx(matrix, cosine_sim)
cosine_network = similarity_net(cosine_mtx, density=DENSITY)
```

```
def pearson_sim(x, y):
   return pearsonr(x, y)[0]

pearson_mtx = similarity_mtx(matrix, pearson_sim)
pearson_network = similarity_net(pearson_mtx, density=DENSITY)

def pearson_sim_sign(x, y):
   r, pvalue = pearsonr(x, y)
   return r if pvalue < 0.01 else 0

pearson_mtx_sign = similarity_mtx(matrix, pearson_sim_sign)
pearson_network_sign = similarity_net(pearson_mtx_sign, density=DENSITY)

# Slice a projected similarity network by threshold or
# density
def slice_projected(net, threshold=None, density=None):
   if threshold is not None:
      weak_edges = [(n1, n2) for n1, n2, w in net.edges(data=True)
              if w["weight"] < threshold]
   else:
      count = int(len(net) * (len(net) - 1) / 2 * density)
      weak_edges = [(n1, n2) for n1, n2, w in
              sorted(net.edges(data=True),
                  key=lambda x: x[2]["weight"],
                  reverse=True)[count:]]
   net.remove_edges_from(weak_edges)


net1, net2 = sets(patients_traumas)
_, traumas = (net1, net2) if "WAR" in net2 else (net2, net1)
hamming_network = weighted_projected_graph(patients_traumas,
                          traumas, ratio=True)
slice_projected(hamming_network, density=DENSITY)
```

```
# Calculate generalized similarities between nodes in a
# bipartite graph
# https://github.com/dzinoviev/generalizedsimilarity/
# blob/master/generalized.py
def generalized_similarity(graph, min_eps=0.01, max_iter=50, weight="weight"):
    if not nx.is_bipartite(graph):
        raise ValueError("Not a bipartite graph")
    s = nx.bipartite.sets(graph)
    arcs = nx.bipartite.biadjacency_matrix(graph, s[0], s[1],
                                           weight = weight).toarray()
    arcs0 = arcs - arcs.mean(axis=1)[:, np.newaxis]
    arcs1 = arcs.T - arcs.mean(axis=0)[:, np.newaxis]
    eps = min_eps + 1
    N = np.eye(arcs.shape[1])
    iters = 0
    while eps > min_eps and iters < max_iter:
        M = arcs0.dot(N).dot(arcs0.T)
        m = np.sqrt(M.diagonal())
        M = ((M / m).T / m).T
        Np = arcs1.dot(M).dot(arcs1.T)
        n = np.sqrt(Np.diagonal())
        Np = ((Np / n).T / n).T
        eps = np.abs(Np - N).max()
        N = Np
        iters += 1
    f = nx.relabel_nodes(nx.Graph(M), dict(enumerate(s[0])))
    g = nx.relabel_nodes(nx.Graph(Np), dict(enumerate(s[1])))
    return (f, g, eps, iters)

net1, net2, eps, n = generalized_similarity(patients_traumas)
_, generalized_network = (net1, net2) if "WAR" in net2 else (net2, net1)
slice_projected(generalized_network, density=DENSITY)
generalized_network.remove_edges_from(nx.selfloop_edges(generalized_network))

# These are our networks for comparison
networks = {
    "generalized" : generalized_network,
    "pearson" : pearson_network_sign,
    "cosine" : cosine_network,
    "hamming" : hamming_network,
    }
```

```
# Find partitions with highest modularity
partitions = [community.best_partition(x) for x in networks.values()]

# Output statistics
statistics = sorted([
        (name,
         community.modularity(best_part, netw),
         len(set(best_part.values())),
         len(list(nx.isolates(netw)))
         ) for (name, netw), best_part in zip(networks.items(), partitions)],key=lambda x:
x[1], reverse=True)
print(statistics)

# Display results
for i, (name, _, _, _) in enumerate(statistics):
    net = networks[name]
    axes = plt.subplot(2, 2, i + 1)
    axes.tick_params(labelbottom=False)
    axes.tick_params(labelleft=False)
    axes.set_title(name)
    pos = nx.spring_layout(net, scale=50)
    nx.draw_networkx(net, pos=pos, font_size=8)
    plt.draw()

plt.tight_layout()
plt.show()
```