Social Network Analysis in Fraud Detection

COMPSCIX404.1-012 Final Project

Introduction to the algorithm

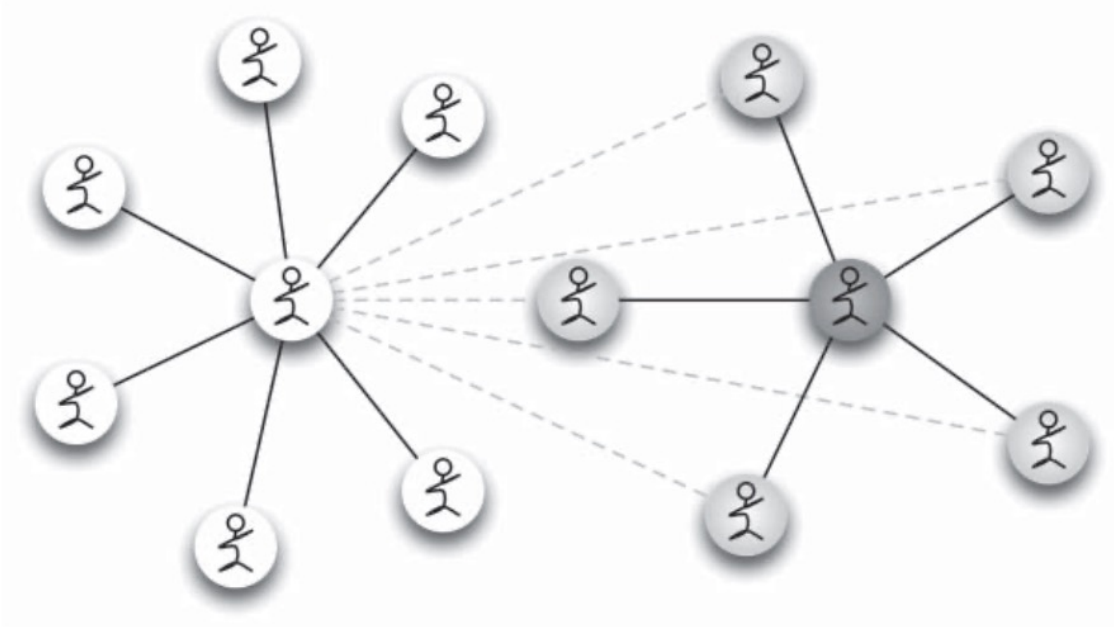
- Social Network Analysis (SNA) is a general study of the social network utilizing network and graph theory concepts which analyzes the behavior of individuals at the micro-level, their relationships (social structure) at the macro level, and the connection between the two.

- Some example methods/algorithms in Social Network Analysis:
  - Network visualization: to represent the structure of the network graphically to better understand its overall properties and patterns, e.g. **NetworkX**.
  - Community detection: to identify groups of nodes that are more densely connected to each other than to the rest of the network, e.g. **homophily**, modularity optimization, hierarchical/spectral clustering.
  - Fraud detection: **PageRank** and some supervised machine learning algorithms (e.g. Graph Neural Network).

- Social Network Analysis in fraud analytics is a technique to represent the entities as nodes and relationships between the entities as links. Some information that could be extracted from SNA:
  - This assumes that the probability of someone committing fraud is dependent on the people s/he is connected to, so-called guilt-by-associations (Koutra et al. 2011).
  - The relationship between two best friends would be very different from the relationship between two distant acquaintances. Such relationships and the intensity are an important source of information exchange.
  - A homophilic network: if nodes with label x (e.g., fraud) are to a larger extent connected to other nodes with label x. It is worthwhile to investigate a network that exhibits evidence of homophily more thoroughly.
  - Advanced network techniques factor the time dimension (e.g. prevent new fraud from growing and the existing fraud from expanding).

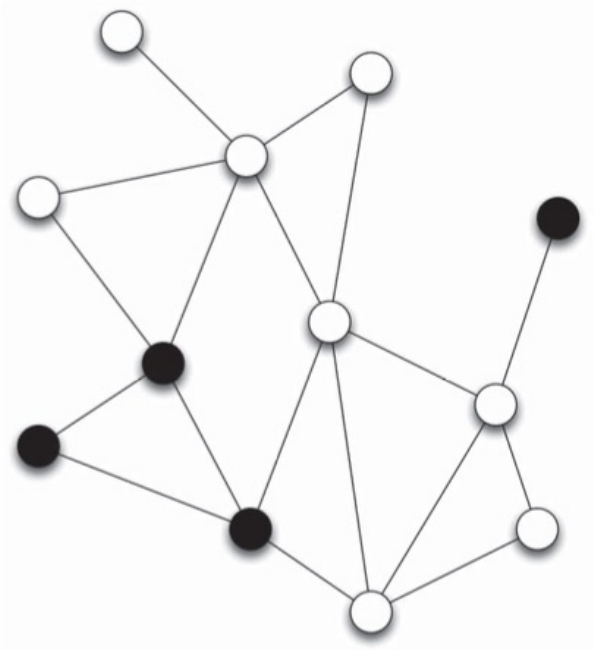
Problems to Solve

- Compared to just listing out the properties of the entities in structured format/relational database, representing the relationships in graph networks reveals more information.
- It is challenging to extract correlations and useful insights from real-life data where there are billions of transactions. Network analysis visualization makes it easy to interpret and understand hidden information from networks.
- Can be used to identify suspicious individuals, groups, relationships, unusual transactions/changes over time/geography and anomalous networks within the overall graph structure.
- For example, in the figure on right, the frequent contact list of a person is suddenly extended with other contacts (light gray nodes). This might indicate that a fraudster (dark gray node) took over that customer's account and "shares" his/her contacts) [1].



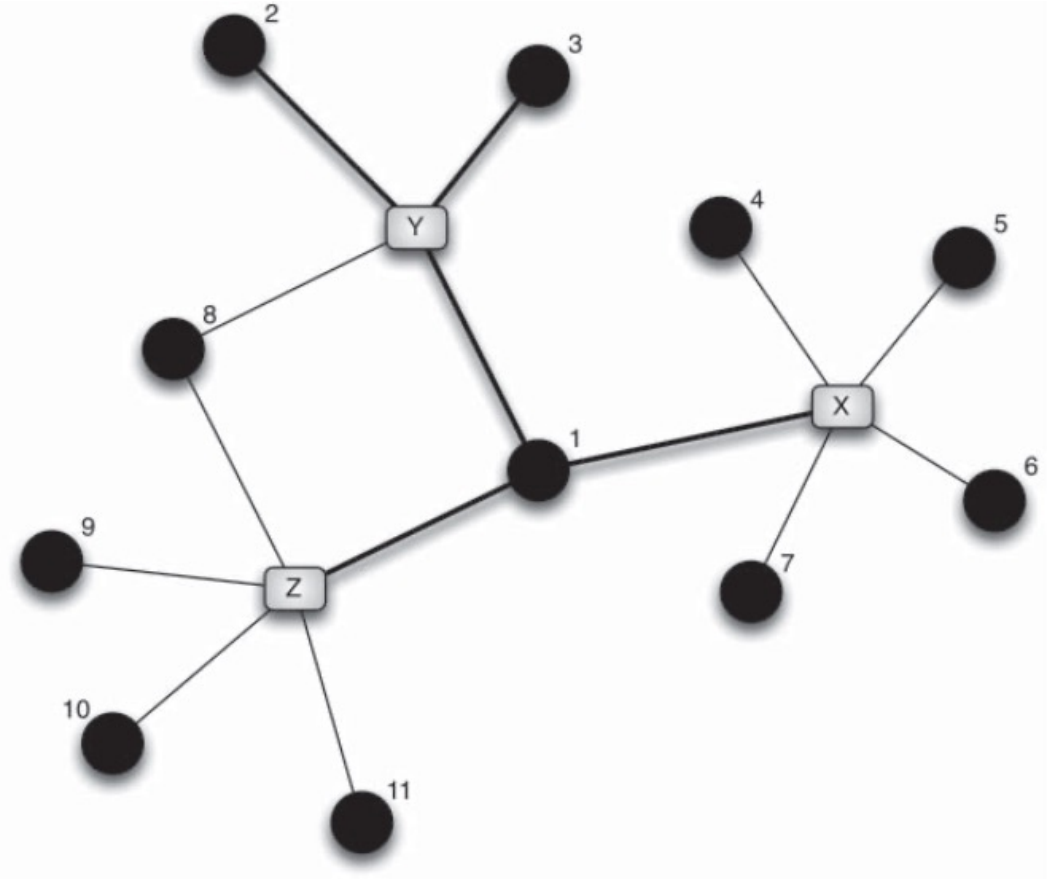
Pseudocode and Algorithm - Homophily

l = the fraction of legitimate nodes in the network
f = the fraction of fraudulent nodes in the network
2*lf* = the expected probability that an edge connects two dissimilar labeled nodes (these are called cross-labeled edges)
$\hat{r}$ = the observed fraction of cross-labeled edges
if the null hypothesis ($\hat{r} \geq 2lf$) is rejected, then a network is homophilic



- The black nodes are the fraudsters and the white are legitimate accounts
- Total nodes = 12, legitimate nodes = 8, fraudulent nodes = 4, total edges = 18
- The network has has 5 cross-labeled edges, and 3 fraud and 10 legit same-labeled edges. $\hat{r} = \frac{5}{18}$
- In a random network, would expect $2lf = 2 * \frac{8}{12} * \frac{4}{12} = \frac{8}{18}$
- The null hypothesis is rejected (at $a = 0.05$)
- The network is homophilic

Pseudocode and Algorithm - Pagerank



- Load transaction data into a dataframe. Input is as follow
  - Rectangles = credit card holders (X, Y, Z)
  - Circles = merchants (1 – 11)
  - Thin edges = legitimate transactions
  - Thick edges = fraudulent transactions
- Create a directed graph adding nodes for the credit card holders and merchants and edges between the credit card holders and merchants
- Compute the total number of transactions for each credit card holder and merchant
- Run the PageRank algorithm to detect fraud (defined as label=1)
- Get PageRank scores for each node
- Based on threshold, fraud flagged in (Y, 1), (Y, 2), (Y, 3), (X, 1), (Z, 1)
- Conclusion
  - the credit card of user Y is stolen.
  - merchant 1 acts suspiciously.

Time and Space Complexity

The time and space complexity of the PageRank algorithm generally depend on the size of the graph and the convergence rate of the algorithm. For a graph with N nodes and M edges:

Time complexity is:

- Initializing PageRank values: O(N)
- Iterating until convergence: O(I * (M + N)), where I is the number of iterations required for convergence
- Sorting of PageRank scores: O(N log N)
- Combining above three steps, the overall time complexity of the PageRank algorithm is O(I * (M + N) + N log N)

Space complexity is:

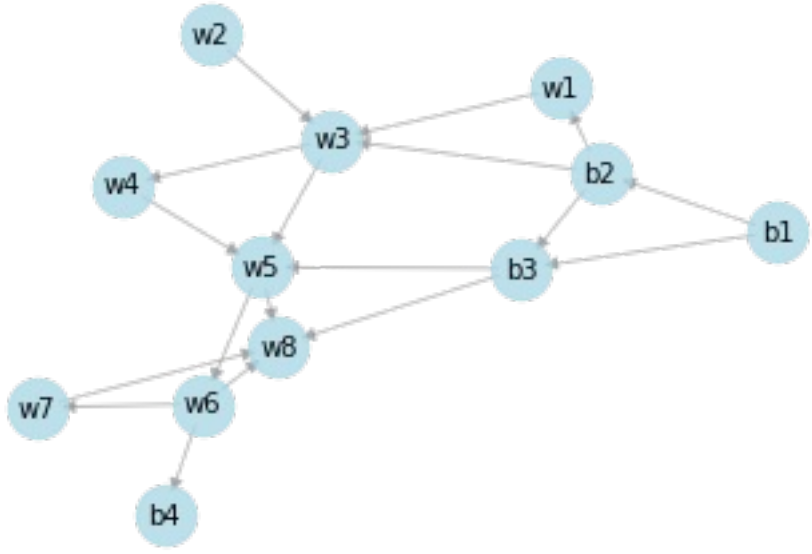
- Storing graph data: O(M + N)
- Storing PageRank values: O(N)
- Combining the above two steps, the overall space complexity of the PageRank algorithm is O(M + 2N)

Code and Example for homophily

```python
# Create the graph
G = nx.Graph()
G.add_nodes_from(['b1', 'b2', 'b3', 'w1', 'w2', 'w3', 'w4', 'w5', 'w6', 'w7', 'w8'])
G.add_edges_from([('b1', 'b2'), ('b1', 'b3'), ('b2', 'b3'), ('b2', 'w1'), ('b4', 'w6'),
                  ('w1', 'w3'), ('b2', 'w3'), ('w2', 'w3'), ('w3', 'w4'), ('w3', 'w5'),
                  ('w4', 'w5'), ('w5', 'b3'), ('b3', 'w8'), ('w8', 'w5'), ('w8', 'w6'),
                  ('w8', 'w7'), ('w5', 'w6'), ('w6', 'w7')])

# Create a dictionary to store the counts of edges between groups
edges_between_groups = {'black-white': 0, 'black-black': 0, 'white-white': 0}

# Iterate over all edges in the graph and count the number of edges between different groups
for edge in G.edges():
    node1 = edge[0]
    node2 = edge[1]
    if (node1.startswith('b') and node2.startswith('w')) or (node1.startswith('w') and node2.startswith('b')):
        edges_between_groups['black-white'] += 1
    elif node1.startswith('b') and node2.startswith('b'):
        edges_between_groups['black-black'] += 1
    elif node1.startswith('w') and node2.startswith('w'):
        edges_between_groups['white-white'] += 1

# Print the results
print("Number of edges between black and white nodes:", edges_between_groups['black-white'])
print("Number of edges between black nodes:", edges_between_groups['black-black'])
print("Number of edges between white nodes:", edges_between_groups['white-white'])
```

```
Number of edges between black and white nodes: 5
Number of edges between black nodes: 3
Number of edges between white nodes: 10
```

```python
# Draw the graph
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos, node_color='lightblue', node_size=500, alpha=0.8)
nx.draw_networkx_edges(G, pos, edge_color='gray', arrows=True, alpha=0.5)
nx.draw_networkx_labels(G, pos, font_color='black', font_size=10)
nx.draw_networkx_edge_labels(G, pos, edge_labels=nx.get_edge_attributes(G, 'label'))
plt.axis('off')
plt.show()
```



Code and Example for homophily, cont.

```
# Define the types of nodes
fraud_nodes = ['b1', 'b2', 'b3', 'b4']
legit_nodes = ['w1', 'w2', 'w3', 'w4', 'w5', 'w6', 'w7', 'w8']

# Calculate the homophily and expected homophily
total_edges = G.number_of_edges()
r_hat = edges_between_groups['black-white'] / total_edges
expected = 2*(len(legit_nodes)/(len(legit_nodes)+len(fraud_nodes)))*(len(fraud_nodes)/(len(legit_nodes)+len(fraud_nodes

# Calculate r_hat and print the results
print("Homophily of the graph:", r_hat)
print("Expected homophily in a random network:", expected)
if r_hat >= expected:
    print('Null hypothesis is not rejected.')
else:
    print('Null hypothesis is rejected.')
```

```
Homophily of the graph: 0.2777777777777778
Expected homophily in a random network: 0.4444444444444444
Null hypothesis is rejected.
```

Code and Example for Pagerank

```python
import pandas as pd
import networkx as nx

# Create a Pandas DataFrame with the data
data = {'credit_card_holder': ['X', 'X', 'X', 'X', 'X', 'Y', 'Y', 'Y', 'Y', 'Z', 'Z', 'Z', 'Z', 'Z'],
        'merchant': [1, 7, 6, 5, 4, 1, 3, 2, 8, 1, 8, 9, 10, 11],
        'label': [1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0]}
df = pd.DataFrame(data)

# Create a directed graph
G = nx.DiGraph()

# Add nodes for the credit card holders and merchants
for holder in df['credit_card_holder'].unique():
    G.add_node(holder, node_type='credit_card_holder')
for merchant in df['merchant'].unique():
    G.add_node(merchant, node_type='merchant')

# Add edges between the credit card holders and merchants
for _, row in df.iterrows():
    user = row['credit_card_holder']
    merchant = row['merchant']
    if row['label'] == 1:
        # Add a fraud edge from the user to the merchant
        G.add_edge(user, merchant, label='fraud')
    else:
        # Add an approved edge from the user to the merchant
        G.add_edge(user, merchant, label='approved')
```

```python
# Compute the total number of transactions for each credit card holder and merchant
total_transactions = {}
for node in G.nodes:
    if G.nodes[node]['node_type'] == 'credit_card_holder':
        total_transactions[node] = sum([1 for _, _, v in G.out_edges(node, data=True) if v['label'] in {'approved', 'fr
    elif G.nodes[node]['node_type'] == 'merchant':
        total_transactions[node] = sum([1 for _, _, v in G.in_edges(node, data=True) if v['label'] in {'approved', 'fra
```

Code and Example for Pagerank, cont.

```
# Run the PageRank algorithm to detect fraud (defined as label=1)
pr = nx.pagerank(G, alpha=0.85, personalization=None, max_iter=100, tol=1e-06,
                 nstart=None, weight='weight', dangling=None)
```

```
# Identify suspicious merchants and stolen credit cards
suspicious_merchants = []
stolen_cards = []
for node, score in pr.items():
    if G.nodes[node]['node_type'] == 'merchant':
        fraud_count = sum([1 for _, _, v in G.in_edges(node, data=True) if v['label'] == 'fraud'])
        total_count = total_transactions[node]
        if total_count > 0 and fraud_count / total_count > 0.5:
            suspicious_merchants.append(node)
    elif G.nodes[node]['node_type'] == 'credit_card_holder':
        fraud_count = sum([1 for _, _, v in G.out_edges(node, data=True) if v['label'] == 'fraud'])
        total_count = total_transactions[node]
        if total_count > 0 and fraud_count / total_count > 0.5:
            stolen_cards.append(node)
```

```
# This simple logic is too strict and has faulse positive for merchant 3 and 2
# So use some threshold instead (see next cell)
print("Suspicious merchants:", suspicious_merchants)
print("Stolen credit cards:", stolen_cards)
```

```
Suspicious merchants: [1, 3, 2]
Stolen credit cards: ['Y']
```

```
# Identify the nodes with the highest PageRank scores
fraud_nodes = [node for node, score in pr.items() if score > 0.09 and G.nodes[node]['node_type'] == 'merchant']
print(f'The following nodes have a high PageRank score and may be associated with fraud: {fraud_nodes}')
```

```
The following nodes have a high PageRank score and may be associated with fraud: [1]
```

Practical applications

- The previous slides focus on applying Social Network Analysis (SNA) in Fraud Detection
- But SNA could be applied to other analysis in various fields such as,
  - Supply Chain Management: helps to improve the operation efficiency via identifying and eliminating less important nodes which represents suppliers/warehouses.
  - Human Resources: helps to identify critical resources/leaders and understand their contribution to the organization, collaboration, and participation.
  - Transmission of Infectious Diseases: assist in detecting and isolating individuals and groups with high betweenness and out-degree centrality (those who spread disease), and execute effective contact tracing measures to mitigate their impact.

Sources

1. Baesens, Bart, Veronique Van Vlasselaer, and Wouter Verbeke. *Fraud analytics using descriptive, predictive, and social network techniques: a guide to data science for fraud detection*. John Wiley & Sons, 2015. pp. 207-277
2. "Fraud through the Eyes of a Machine." *Medium*, 22 Nov. 2022, *https://www.*towardsdatascience.com/fraud-through-the-eyes-of-a-machine-1dd994405e6e.
3. "PageRank." *Wikipedia*, https://en.wikipedia.org/wiki/PageRank.
4. "PageRank Algorithm, Fully Explained." *Medium*, 19 Dec. 2020, *https://www.*en.wikipedia.org/wiki/PageRank.
5. "A Guide to Social Network Analysis and Its Use Cases." *https://www.Latentview.Com/Blog/A-guide-to-social-network-analysis-and-its-use-cases/*, 23 Apr. 2021.
6. Koutra, Danai, et al. "Unifying guilt-by-association approaches: Theorems and fast algorithms." *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011, Proceedings, Part II 22*. Springer Berlin Heidelberg, 2011.
7. https://aws.amazon.com/blogs/machine-learning/detect-financial-transaction-fraud-using-a-graph-neural-network-with-amazon-sagemaker/
8. https://www.publichealth.columbia.edu/research/population-health-methods/social-network-analysis