REPORT - Project 2.2

Recommending Missing links in the Network

April 24, 2024

Name	Charvi Pahuja
Registration number	2023CSB1114

1 What are missing links?

In a network graph, **missing links** are connections or edges between nodes that do not exist but should be there or are expected to potentially exist in the graph. This is decided on the basis of the structure or characteristics of the network. In a **social network** like ours, these could represent friendships that haven't been discovered yet or undiscovered relationships, potential interactions, or simply gaps in the data. They're like pieces of a puzzle that are absent but could fit in based on what we know about the network.

2 Why is identification of missing links important?

Identifying missing links in a network is crucial for several reasons. It helps in understanding the complete structure of the network. This understanding is essential for various applications such as:

1. **Prediction:** Predicting missing links can anticipate future interactions or relationships within the network. For instance, in a social network, predicting friendships or potential collaborations in a professional network.
2. **Recommendation Systems:** Knowing missing links can improve recommendation systems. By predicting potential connections, these systems can suggest new friends, products, or collaborators based on the network's characteristics.
3. **Network Growth Analysis:** Identifying missing links aids in understanding the growth dynamics of the network. It helps in understanding the patterns of connectivity and how the network evolves over time.
4. **Resource Allocation:** In networks like transportation or infrastructure, identifying missing links helps optimize resource allocation. It allows planners to focus on areas lacking connections and improve efficiency.
5. **Error Detection:** Sometimes, missing links indicate errors in the data or measurement. Detecting these gaps helps in data validation and ensures the accuracy of network representations.
6. **Security and Anomaly Detection:** In security networks, identifying missing links can help in detecting anomalies or potential security threats. Sudden changes or unexpected connections could indicate malicious activity.

3 Methods of finding missing links:

To identify missing links, various techniques are employed, including statistical models, machine learning algorithms, and network analysis methods. Several link prediction approaches have been proposed which include:

1. Matrix method
2. Common neighbors
3. Jaccard measure
4. Adamic-Adar measure
5. Eucledian distance
6. Cosine similarity
7. and many more ...

3.1 Methods that we are using:

3.1.1 Matrix method-

This method uses the adjacency matrix of the network to find out the missing links. Let's say the elements of the adjacency matrix are a_{ij} where i stands for the row index of the element and j for the column one. Then, if a_{ij} is 0 means that there is no edge between the nodes i and j in the network graph. So, we can check for the possibility of missing links for these a_{ij}.

For one a_{ij},we follow these steps to check:

Step1 We first eliminate the i^{th} row and j^{th} column from the matrix to get a matrix A.

Step2 Then to find out vector b, we take the i^{th} row of the adjacency matrix and delete the j^{th} element from this.

Step3 Then we use the Numpy dictionary of Python to calculate the coefficients for the linear combination using the ***Least square approximation*** method.

Step4 Now we use these coefficients (Let's say $[\alpha_1, \alpha_2, \dots]$) and the column vector of j^{th} column of adjacency matrix without the i^{th} element ($[a_{1j}, a_{2j}, \dots, a_{nj}]$) to write the element a_{ij} as the ***Linear combination*** of $[\alpha_1 * a_{1j} + \alpha_2 * a_{2j} + \dots\ ]$

Step5 Finally, we use these values and threshold to decide which of these are actually missing links.

```
#matrix method-------------------------------------------------------------------------------
#making adjacency matrix for our network graph
m = nx.adjacency_matrix(G).toarray()

#function for creating missing link matrix using m
def missing_link_matrix(m,threshold):
    #creating dictionary to store if missing link or not
    B={}

    #running loop in the matrix for all i,j
    for i in range(m.shape[0]):
        for j in range(m.shape[1]):
            #deleting row and column from matrix to get A for the formula
            A = np.delete(m, i, axis=0)
            A = np.delete(A, j, axis=1)

            #for finding b,making vector with column deleted
            b = np.delete(m[i], j, axis=0)

            #using linear algebra least square approximation, finding out coefficients for calculation
            lsa_coeff = np.linalg.lstsq(A, b, rcond=None)[0]

            #calculating linear combination of column vector with coefficients we found
            P = np.delete(m, i, axis=0)
            comb = P * lsa_coeff[:, np.newaxis]
            V = np.sum(comb, axis=0)
            ml = V[j]

            #boolean to store missing link
            boole=False
            #change if missing link
            if ml > threshold:
                boole=True

            #appending the value of bool to dictionary
            B[(i,j)]=boole

    #creating a matrix that has ones only for missing link of the graph
    missing_link_matrix = np.zeros_like(m)  #initiating as zeroes
    #runnning loop for i,j in the graph
    for i in range(m.shape[0]):
```

```
    #runnning loop for i,j in the graph
    for i in range(m.shape[0]):
        for j in range(m.shape[1]):
            #if not present in original matrix
            if m[i, j] == 0:
                #if it is missing link
                if B[(i,j)]==True:
                    #append 1 in the matrix
                    missing_link_matrix[i][j] =1

    return missing_link_matrix

#function to find missing links in the graph
def missing_links(m, missing_link_matrix, nodes):
    #empty list to store the missing links
    missing_links = []

    #runnning loop for i,j in the matrix
    for i in range(missing_link_matrix.shape[0]):
        for j in range(missing_link_matrix.shape[1]):

            #append if missing link
            if missing_link_matrix[i, j] == 1:
                missing_links.append((nodes[i], nodes[j]))

    return missing_links

#for our graph
nodes = list(G.nodes())
threshold = 0.99

#calling functions for our graph to printing the number of missing links in our network
missing_link_matrix = missing_link_matrix(m, threshold)
missing_links = missing_links(m, missing_link_matrix, nodes)
print("Number of missing links using Matrix Method:", len(missing_links))
print(missing_links[i])
```

Figure 1: Code snippet for matrix method

3.1.2 Jaccard Coefficient Method-

This method uses the Jaccard measure value for two nodes to determine whether there exists a missing link between these two nodes.

Jaccard measure

The Jaccard measure, also known as the Jaccard similarity coefficient, is a statistic used for comparing the similarity of two nodes by calculating the ratio of the size of the intersection of the sets of neighbors of the two nodes to the size of the union of the sets.
Mathematically, the Jaccard measure $J(A, B)$ for two nodes A and B is defined as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Where:
$|A \cap B|$ represents the size of the intersection of sets of neighbors of nodes A and B, i.e., the number of elements common to both sets. $|A \cup B|$ represents the size of the union of sets of neighbors of nodes A and B, i.e., the total number of distinct elements in both sets.

The Jaccard measure ranges from 0 to 1. A value of 0 indicates no similarity between the nodes, while a value of 1 indicates that the nodes are identical.

In our code, a threshold is set for the measure for two nodes and if the value surpasses the threshold for a certain pair of nodes, then there exists a missing link between those two nodes.

```python
#jaccard coefficient method-------------------------------------------------------------------------
#first creating a threshold value for Jaccard coefficient
threshold_j = 0.134

#creating empty list to store missing links by this method
missing_links_j = []

#for all the nodes in the graph
for i in G.nodes():
    for j in G.nodes():
        if i != j and not G.has_edge(i, j): #when there is no edge between chosen nodes
            #storing neighbors for both nodes
            neighbors_1 = set(G.neighbors(i))
            neighbors_2 = set(G.neighbors(j))
            #calculating Jaccard coefficient for i,j by definition
            intersection = len(neighbors_1.intersection(neighbors_2))
            union = len(neighbors_1.union(neighbors_2))
            if union != 0: #preventing division by zero
                jaccard_coeff= intersection / union
                #appending to list if coeff exceeds our set threshold
                if jaccard_coeff > threshold_j:
                    missing_links_j.append(((i,j), jaccard_coeff))

#printing number of missing links in network using jaccard coefficient
print("Number of missing links using Jaccard coefficient:", len(missing_links_j))
print(missing_links_j[9])
```

Figure 2: Code snippet for jaccard method

3.2 Running our code:

```
In [2]: runfile('C:/Users/nares/OneDrive/Desktop/cs101/project2/2023CSB1114-Project
2.2.py', wdir='C:/Users/nares/OneDrive/Desktop/cs101/project2')
Number of missing links using Matrix Method: 3607
('2023CSB1109', '2023CSB1095')
Number of missing links using Jaccard coefficient: 3488
(('2023CSB1109', '2023CSB1095'), 0.2)
```

Figure 3: Output Observed

On running our code, we observe the number of missing links using both methods as follows:

Matrix	Jaccard
3607	3488

As both methods provide almost the same number of missing links in the graph. Hence, our result is verified.