Solution Approach

Solution Part 1:

a) Finding Missed Links Using Link Prediction Methods(Using only Local Heuristics)

In this section, we outline the mathematical approach and source code implementation for solving the problem of finding missed links using link prediction methods.

Mathematical Approach

Local Heuristics:

Common Neighbors:

Implementation on Instagram: if two users have common followers or followees,it suggests a potential link between them.

$$CN(n_1, n_2) = \text{Number of common followers between } n_1 \text{ and } n_2$$

Adamic Adar:

Implementation on Instagram: In our analysis, we leverage Adamic Adar to identify potentially overlooked connections between users.

$$AA(n_1, n_2) = \sum_{z \in \text{Common Followers}} \frac{1}{\log(\text{Degree of } z)}$$

Preferential Attachment:

Implementation on Instagram: This heuristic would predict new connections forusers with many followers or followees, reflecting their growing network.

$$PA(n_1, n_2) = \text{Degree of } n_1 \times \text{Degree of } n_2$$

Jaccard Coefficient:

$$JC(n_1, n_2) = \frac{\text{Number of common followers}}{\text{Number of unique followers of } n_1 \text{ and } n_2}$$

Implementation on Instagram: The likelihood of a connection is determined by the ratio of shared connections to the total number of unique connections of bothusers.

Implementation in Source Code

```python
def calculate_scores(node_1, node_2):
    followers_node1 = followers_map.get(node_1, set())
    followers_node2 = followers_map.get(node_2, set())
    common_followers = followers_node1.intersection(followers_node2)



    # Jaccard's Coefficient
    jc = len(followers_node1.intersection(followers_node2)) / len(followers_node1.union(followers_r

    # Adamic Adar
    aa = sum(1 / math.log(degree_map.get(z, 1.0)) for z in common_followers if degree_map.get(z, 1.

    # Preferential Attachment
    pa = degree_map.get(node_1, 0) * degree_map.get(node_2, 0)

    # Common Neighbors
    cn = len(common_followers)

    return cn, jc, aa, pa
```

`

```python
# Process chunk within given range.
def process_node_chunks(nodePairs, current_chunk_id, existing_followers):
    predicted_links = []
    for node_1, node_2 in nodePairs:
        if node_1 in existing_followers and node_2 in existing_followers[node_1]:
            # if current one is a follower, skip
            continue
        cn, jc, aa, pa = calculate_scores(node_1, node_2)

        # Using lesser thresholds as we have very few connections given in second data second. Most
        cn_threshold = 1
        jc_threshold = 0.01
        aa_threshold = 0.6
        pa_threshold = 1000

        if cn > cn_threshold and jc > jc_threshold and aa > aa_threshold and pa > pa_threshold:
            predicted_links.append((node_1, node_2))
```

Mapping with Source Code

Data Reading and Initialization

- followers_count_dataSet: Reads the dataset with user statistics (followers count).
- dataset_with_few_edges: Reads the dataset with a subset of edges.
- Initialization of column names for easy identification.

Data Mapping and Preprocessing

- Creation of maps for followers and degrees for efficient retrieval.Link

Prediction Function - calculate_scores

- Takes two nodes as input.
- Calculates Common Neighbors (cn), Jaccard Coefficient (jc), Adamic Adar (aa), and Preferential Attachment (pa) scores.

Chunk Processing - process_node_chunks

- Processes node pairs within a given chunk.
- Applies thresholds for link prediction based on calculated scores.
- Saves the predicted links to a new CSV file for further analysis.Node

Pair Iteration

- Iterates over all unique node pairs in chunks.
- Calls process_node_chunks for each chunk.

b) Finding Missed Links Using Link Prediction Methods(Using both Local andGlobal Heuristics)

In this section, we describe the solution method mathematically and map it with the provided source code implementation. The objective is to predict missed linksin the Instagram network using various local heuristics and global heuristics.

Apart from the local heuristics used in the previous method we are using global heuristics as well.

Katz Centrality:

Implementation in Source Code:

```
# Finding Katz Centrality using the power method

def find_katz_centrality(adj_matrix_subset, alpha=0.005, max_iter=1200,
tolerance=1e-5):
      #...refer original code

# Function we use to generate the adjacency matrix for a subset of nodes

def generate_adjacency_matrix(nodes_subset, followers_map):

       #...refer original code
```

```python
# Process chunk within given range.
def process_node_chunks(nodes_subset, chunk_index):
    # Generating adjacency matrix for the current subset as given in the index range based on chunk size
    adj_matrix_subset = generate_adjacency_matrix(nodes_subset, followers_map)

    katz_centrality = find_katz_centrality(adj_matrix_subset)

    # Here we store the predicted data links
    predicted_links_df = pd.DataFrame(columns=['source_node', 'destination_node'])

    # Iterate over all pairs and compute heuristic scores
    for i, node_1 in enumerate(nodes_subset):
        for j, node_2 in enumerate(nodes_subset):
            if i != j:
                # Calculate heuristic scores
                cn, jc, aa, pa = calculate_scores(node_1, node_2)

                # Thresholds for deciding whether to connect nodes
                cn_threshold = 1
                jc_threshold = 0.01
                aa_threshold = 0.5
                pa_threshold = 1000
                katz_threshold = 0.1  # Katz score threshold

                # Check if pair exceeds all thresholds
                if ((cn > cn_threshold and jc > jc_threshold and aa > aa_threshold and pa > pa_threshold) or
                    (katz_centrality[i] > katz_threshold and katz_centrality[j] > katz_threshold)):
                    new_row = pd.DataFrame({
                        'source_node': [node_1],
                        'destination_node': [node_2]
                    })
                    predicted_links_df = pd.concat([predicted_links_df, new_row], ignore_index=True)
```

Mapping with Source Code

Data Reading and Initialization

- Reads the dataset with user statistics (followers_count_dataSet).
- Reads the dataset with a subset of edges (dataset_with_few_edges).
- Initialization of column names for easy identification.

Data Mapping and Preprocessing

- Creation of maps for followers and degrees for efficient retrieval.Link

Prediction Function - calculate_scores

- Takes two nodes as input.
- Calculates Common Neighbors (cn), Jaccard Coefficient (jc), Adamic Adar (aa), and Preferential Attachment (pa) scores.

Katz Centrality Calculation - find_katz_centrality

- Uses the power iteration method to calculate Katz Centrality.

Adjacency Matrix Generation - generate_adjacency_matrix

- Generates the adjacency matrix for a subset of nodes.

Chunk Processing - process_node_chunks

- Processes node pairs within a given chunk.
- Applies thresholds for link prediction based on calculated scoresand Katz Centrality.

- Saves the predicted links to a new CSV file for further analysis.Node

Pair Iteration

- Iterates over all unique node pairs in chunks.
- Calls process_node_chunks for each chunk.

Solution Part 2: Identifying Suspicious Instagram Accounts

In this solution, we aim to identify suspicious Instagram accounts using various metrics such as Eigenvector Centrality, Network Density, Engagement Rates, Growth Rates, and Outsider Interaction Ratios. The approach involves mathematical analysis and mapping with the provided source code for clarity.

1. Eigenvector Centrality Analysis

- Mathematical Description:
  - Calculate Eigenvector Centrality for each account in the predictedlinks.
  - Normalize the centrality scores.
- Source Code Mapping:

```
# Finding the  eigenvector centrality

new_graph = nx.from_pandas_edgelist(predicted_links, 'source_node',
'destination_node')

ev_centrality = nx.eigenvector_centrality_numpy(new_graph)



# Normalizing the centrality scores

normalized_centrality = minmax_scale(list(ev_centrality.values()))
```

2. Network Density Calculation

- Mathematical Description:
  - Calculate the network density of the predicted links graph.
- Source Code Mapping:

```
# Calculating the network density. An account with high density and low engagement rate
and low growth rate and low outsider reach is also suspicious


density = nx.density(new_graph)
```

3. Suspicious Account Identification based on Eigenvector Centrality and Engagement Rate

- Mathematical Description:
  - Identify accounts with high eigenvector centrality but low engagement rates.
  - **1. Eigenvector Centrality:**

$$x_i = \frac{1}{\lambda} \sum_{j=1}^{n} A_{ij} x_j$$

Where:

- A_{ij} is the element in the adjacency matrix indicating whether there is a connection between nodes i and j.
- n is the number of nodes in the network.
- λ is a constant.

2.Engagement Rate:

$$ER = \frac{\text{Total Engagement Activities}}{\text{Total Followers or Connections}}$$

- Source Code Mapping:

```
ret_suspicious_wrt_centrality = insta_data[

    (insta_data['eigenvector_centrality'] >
insta_data['eigenvector_centrality'].median()) &

    (insta_data['er'] < low_engagement_threshold)

]
```

4. Suspicious Pattern Detection based on Density, Engagement Rate,Growth Rate, and Outsider Reach

- Mathematical Description:
    - Identify accounts with high density, low engagement rate, low growth rate, and low outsider reach.
- Source Code Mapping:

```
suspicious_accounts_high_density =

   insta_data[ (insta_data['er'] < low_engagement_threshold) &

   (insta_data['fg'] < low_growth_threshold) &

   (insta_data['op'] < insta_data['op'].median())
```

5. Outsider Interaction Analysis

- Mathematical Description:
    - Examine the ratio of outsider interactions (op) to the engagement grade (eg) to identify anomalies.
    - Normalize the outsider interaction ratio.

- Source Code Mapping:

```
# Outsider Interaction Analysis:- We will examine the ratio of outsider interactions (op) to the engagement grade (eg) to identify anomalies.

insta_data['op_to_eg_ratio'] = insta_data['op'] / insta_data['eg']



# Normalize the op_to_eg_ratio

insta_data['op_to_eg_ratio_norm'] = minmax_scale(insta_data['op_to_eg_ratio'])
```

6. Identifying Accounts with Abnormal Outsider Interaction Ratios

- Mathematical Description:
  - Identify accounts with abnormal outsider interaction ratios.
- Source Code Mapping:

```
# Identifying accounts with abnormal outsider interaction ratios

abnormal_outsider_interaction_threshold =
insta_data['op_to_eg_ratio_norm'].quantile(0.85)

suspicious_accounts_outsider_interaction =

   insta_data[ insta_data['op_to_eg_ratio_norm'] >

   abnormal_outsider_interaction_threshold
```

7. Plotting for Visualization

- Mathematical Description:
  - Visualize the Engagement Rate against various suspicious patterns.
- Source Code Mapping:

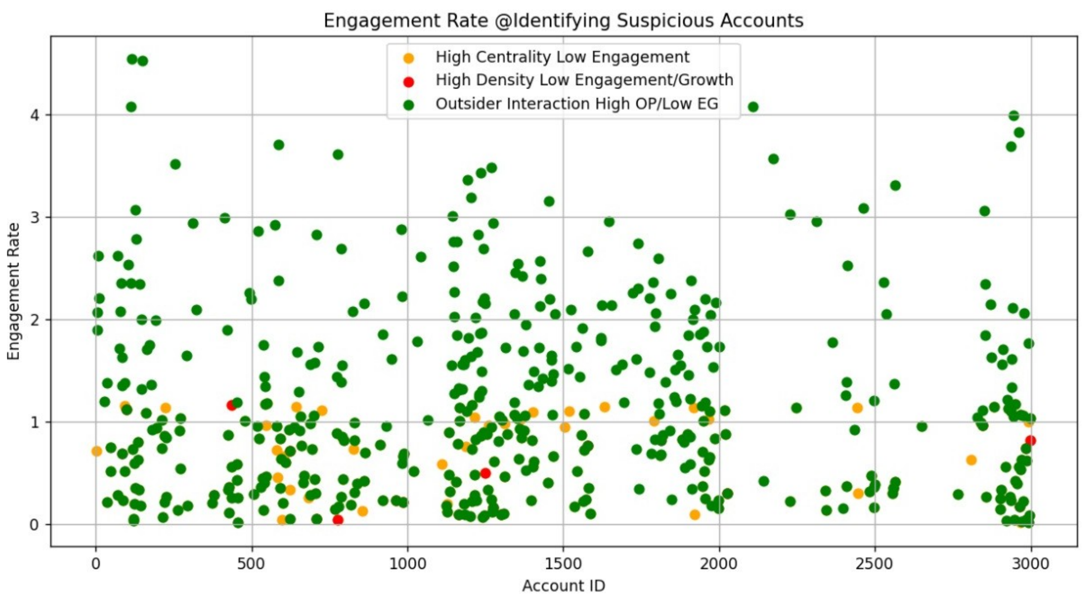
```
# Plotting the graph to find outliers

plt.figure(figsize=(14, 7))

# … (Plotting code)

plt.show()
```

Results

The application of these heuristics is expected to reveal **fake** accounts. A subsetof predicted edges and the actual instagram data has been taken along with various parameters and has been analyzed.

As shown in the below graph the coincidence of two or more points would indicate that there is a greater possibility that the specific user is a bot / suspicious account.

Higher outsiders interaction with lower engagement rate could be a sign that the account might have purchased bot followers to artificially inflate its followers count.



Insights

Combination of Heuristics Enhances Link Prediction:

Employing both local (like Common Neighbors, Adamic Adar) and global heuristics (such as Katz Centrality) provides a robust method for uncoveringhidden connections in the Instagram network, indicating the importance of amulti-faceted approach in network analysis.

Eigenvector Centrality as an Indicator of Potential Inauthenticity:

The analysis reveals that accounts with high eigenvector centrality but low engagement rates are potential flags for inauthentic behavior, suggesting either inflated influence or low-quality engagement.

Network Density Versus Engagement Insights:

Accounts characterized by high network density but low engagement and growthrates potentially indicate non-genuine interactions or inactive user bases, offering a new dimension for identifying suspicious accounts.

Outsider Interaction Ratio as a Novel Anomaly Indicator:

The ratio of outsider interactions to engagement grades emerges as a significant metric for anomaly detection, helping to identify accounts with unusually high engagement from non-followers, which could be indicative of artificial boosting techniques.