Individual Project 1

Seed-Based De-anonymization

De-anonymization, within the realm of privacy and data security, refers to the procedure of revealing or associating anonymous or pseudonymous data with particular individuals or entities. This method involves the identification of information deliberately obscured or anonymized in G1 (V, E), the original graph containing privacy-related data. G2 (V, E) serves as an auxiliary graph containing identity-related information. The objective of de-anonymization is to unveil concealed details and establish connections between anonymized data in G1 and the corresponding identities present in G2.

In our approach, we engage with two graphs: G1(V, E) and G2(V, E). G1 encompasses sensitive information, whereas G2 comprises identifiable data. Within these graphs, there exist specific nodes that we've matched, denoted as seed pairs. Our objective is to use these established matches as a foundation to discover additional analogous pairs between the graphs.

The algorithm capitalizes on the inherent structure of the network and utilizes past mapping efforts to intelligently unearth new connections. To initiate this process, we utilize the networkx package to import the edge list data of both graph 1 and graph 2.

Subsequently, we incorporate the seed-pairs data, stored in a .txt file within the project data and presented in JSON format in the mini-dataset. When loading the JSON file, it seamlessly transforms into a dictionary.

We segment the seed-pair dictionary into two distinct node lists: one comprising all nodes associated with graph 1, including the seed-pair list, and the other consisting of nodes corresponding to graph 2.

For each node within graph 1, we calculate its degree and determine the degree of the corresponding node in graph 2. Subsequently, we examine their respective neighbors, checking for existing mappings and the presence of their pairs in the seed pair list. This systematic process aids in identifying unmapped nodes and assesses their similarity through the computation of a Score.

$$score(node_1, node_2) = \frac{count\ of\ already\ mapped\ neighbors\ between\ node_1\ \&\ node_2}{\sqrt{Degree\ of\ node_1} * \sqrt{Degree\ of\ node_2}}$$

```
def load_score(node1_degree, node2_degree, cnt):
  score = cnt/((math.sqrt(node1_degree))*(math.sqrt(node2_degree)))
  return score


def load_ECCE(score_list):
  std = np.std(score_list)
  max1 = max(score_list)
  score_list.remove(max1)
  max2 = max(score_list)
  if std == 0:
    result = 0
  else:
    result = (max1-max2)/std
  return result

def load_key(dict, value):
  for k, v in dict.items():
    if v == value:
      a=k
  return a
```

We compute the scores for all these nodes, storing them in a set. Subsequently, we calculate the eccentricity for each iteration using the formula:

$ECCE$=score(max1)−score(max2) / σ

During each round, we keep a record of all the calculated similarity scores. We gauge the dispersion of these scores using standard deviation. Following that, we identify the two highest scores and assess their variance. This variance serves as a metric for determining the "eccentricity" of our findings.

When the eccentricity value exceeds 0.5, it signifies a robust match. At this point, we establish a link between the node in the first graph and the node in the second graph that yielded the highest score. This iterative process continues, continuously searching for new matches until no additional linked pairs can be identified among the initial seed pairs.

Ultimately, we incorporate all these matched pairs into the initial set, constituting our final output – a file listing all the node pairs we have discovered.

```
uploaded_G1 = files.upload()
uploaded_G2 = files.upload()
uploaded_seed_pairs = files.upload()

s1 = 'G1.edgelist'
s2 = 'G2.edgelist'
s_pair = 'seed_mapping.txt'
v = 0.5
G1_pair, G2_pair, G1_pair_dir, G1_unpair, G2_unpair, old_G1_pair, old_G2_pair, old_G1_pair_dir = running(s1, s2, s_pair, v)
count = 0

# Changing the loop range to use the length of old_G1_pair
for i in range(len(old_G1_pair)):
    # Checking if the item is in G1_pair and G1_pair_dir matches old_G1_pair_dir
    if old_G1_pair[i] in G1_pair and G1_pair_dir[old_G1_pair[i]] == old_G1_pair_dir[old_G1_pair[i]]:
        count += 1

# Accuracy calculation
accuracy = count / len(old_G1_pair)
print('The accuracy is: ' + str(accuracy * 100) + '%')  # Multiply by 100 to get the percentage

print(len(G1_pair))
print(len(G1_unpair))


with open('file_pairs.txt', 'w')as f:
  for node, direction in zip(G1_pair, G1_pair_dir.values()):
    s = f"{node} {direction}\n"
    f.write(s)

files.download('file_pairs.txt')
```

The total number of pairs I got are: **24965**

Seed-Free De-anonymization

This project facilitates the comparison of node characteristics between two graphs, namely unseed_G1 and unseed_G2, following the upload of their respective edge list files. The process involves extracting and analyzing node features such as degree and clustering coefficient to identify similar node pairs across these graphs.

Data Upload: Users have the option to upload edge list files (unseed_G1.edgelist and unseed_G2.edgelist) representing graphs G1 and G2, respectively.

Graph Loading: The uploaded data is loaded into NetworkX graphs, named unseed_G1 and unseed_G2, respectively.

Feature Extraction: Node features, including degree (the number of connected edges) and clustering coefficient (indicating neighbor interconnectivity), are computed for each node in unseed_G1 and unseed_G2. These features are stored in arrays (features_unseed_G1 and features_unseed_G2), where rows correspond to nodes, and columns represent degree and clustering coefficient values.

Matching Procedure: An empty list named matched_pairs is initialized to store pairs of nodes that exhibit significant similarity based on their features.

It conducts a feature comparison between nodes in unseed_G1 and unseed_G2 utilizing cosine similarity. For each node in unseed_G1, the algorithm computes the cosine similarity with all nodes in unseed_G2. It identifies the node in unseed_G2 that exhibits the highest similarity score with the node in unseed_G1.

If the computed similarity score surpasses 0.5, indicating substantial similarity, the algorithm records the node pair (node index in unseed_G1, best-matching node index in unseed_G2) in the matched_pairs list.

```
set_B = set(list_B)

print("Set from A:", len(set_A))
print("Set from B:", len(set_B))
```

```
Set from A: 8561
Set from B: 4442
```

```
n = set()

with open('unseed_G1.edgelist', 'r') as f:
    for line in f:
        node = int(line.split()[0])
        n.add(node)

print(len(n))
```

```
8561
```

```
n = set()

with open('unseed_G2.edgelist', 'r') as f:
    for line in f:
        node = int(line.split()[0])
        n.add(node)

print(len(n))
```

```
8547
```

Here the set from G1 – 8561

Set from G2 - 8547