Assignment 2

Course: Social Networks

Name: Mohamed Hassan Ahmed
ID: 2205200

Introduction

This report studies the Facebook Ego Network dataset to detect bots in a social network. We compute graph metrics, train a baseline classifier, and analyze how two attacks—Structural Evasion and Graph Poisoning—affect both the graph structure and detection performance

1. Import Libraries

Explanation: We import all necessary Python libraries for graph analysis, machine learning, and visualization.

[1]:
```
import networkx as nx
import pandas as pd
import numpy as np
from networkx.algorithms.community import label_propagation_communities
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
```

2. Load Facebook Ego Dataset

Explanation: We load the edges from the dataset and build the network graph.

[3]:
```
# Load edges
edges_file = "facebook/0.edges"
edges = pd.read_csv(edges_file, sep=" ", header=None, names=["source", "target"])

# Build graph
G = nx.from_pandas_edgelist(edges, "source", "target")
print(f"Graph loaded with {G.number_of_nodes()} nodes and {G.number_of_edges()} edges")
```

```
Graph loaded with 333 nodes and 2519 edges
```

3. Add Node Labels (Simulated Bots)

Explanation: We randomly select 5% of nodes as bots for simulation.

[5]:
```
# Simulate some bots (5% of nodes)
num_bots = int(0.05 * G.number_of_nodes())
bot_nodes = np.random.choice(G.nodes(), size=num_bots, replace=False)
labels = {n: 1 if n in bot_nodes else 0 for n in G.nodes()}  # 1=bot, 0=normal
```

4. Compute Graph Features

Explanation: We calculate node metrics: degree, clustering coefficient, betweenness centrality, and communities. These are used as features for classification.

[7]:
```
# Compute structural features
degree = dict(G.degree())
clustering = nx.clustering(G)
betweenness = nx.betweenness_centrality(G)
communities = list(label_propagation_communities(G))
community_map = {n: i for i, c in enumerate(communities) for n in c}

# Create feature DataFrame
features = pd.DataFrame({
    "node": list(G.nodes()),
    "degree": [degree[n] for n in G.nodes()],
    "clustering": [clustering[n] for n in G.nodes()],
    "betweenness": [betweenness[n] for n in G.nodes()],
    "community": [community_map[n] for n in G.nodes()],
})
features["label"] = features["node"].map(labels)
features.head()
```

[7]:

	node	degree	clustering	betweenness	community	label
0	236	36	0.376190	0.003643	0	1
1	186	43	0.473976	0.005513	0	0
2	122	62	0.365415	0.022754	0	0
3	285	46	0.407729	0.013690	0	0
4	24	15	0.885714	0.000107	1	0

5. Baseline Bot Detection

Explanation: Train a Random Forest classifier on the original graph to detect bots. Evaluate using accuracy and classification report.

[9]:
```
X = features[["degree", "clustering", "betweenness", "community"]]
y = features["label"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print("=== Baseline Bot Detection ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
=== Baseline Bot Detection ===
Accuracy: 0.96
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        96
           1       0.00      0.00      0.00         4

    accuracy                           0.96       100
   macro avg       0.48      0.50      0.49       100
weighted avg       0.92      0.96      0.94       100
```

6. Structural Evasion Attack

Explanation: Bots modify their edges—removing some old connections and adding new ones to normal nodes—to blend in and evade detection

[11]:
```
G_evasion = G.copy()
for bot in bot_nodes:
    neighbors = list(G_evasion.neighbors(bot))
    # Remove half of existing edges
    remove_count = max(1, len(neighbors)//2)
    to_remove = np.random.choice(neighbors, remove_count, replace=False)
    G_evasion.remove_edges_from([(bot, n) for n in to_remove])

    # Add 2 new edges to random normal nodes
    normal_nodes = [n for n in G.nodes() if n not in bot_nodes]
    new_edges = np.random.choice(normal_nodes, 2, replace=False)
    for n in new_edges:
        G_evasion.add_edge(bot, n)
```

7. Features After Evasion Attack

Explanation: Recalculate graph features after the evasion attack to retrain the classifier.

[13]:
```
degree_ev = dict(G_evasion.degree())
clustering_ev = nx.clustering(G_evasion)
betweenness_ev = nx.betweenness_centrality(G_evasion)
communities_ev = list(label_propagation_communities(G_evasion))
community_map_ev = {n: i for i, c in enumerate(communities_ev) for n in c}

features_ev = pd.DataFrame({
    "node": list(G_evasion.nodes()),
    "degree": [degree_ev[n] for n in G_evasion.nodes()],
    "clustering": [clustering_ev[n] for n in G_evasion.nodes()],
    "betweenness": [betweenness_ev[n] for n in G_evasion.nodes()],
    "community": [community_map_ev[n] for n in G_evasion.nodes()],
})
features_ev["label"] = features["node"].map(labels)
```

8. Train Classifier After Evasion Attack

Explanation: Train the classifier on the evasion-attacked graph and measure performance drop.

[15]:
```
X_ev = features_ev[["degree", "clustering", "betweenness", "community"]]
y_ev = features_ev["label"]

X_train_ev, X_test_ev, y_train_ev, y_test_ev = train_test_split(X_ev, y_ev, test_size=0.3, random_state=42)

clf_ev = RandomForestClassifier(n_estimators=100, random_state=42)
clf_ev.fit(X_train_ev, y_train_ev)
y_pred_ev = clf_ev.predict(X_test_ev)

print("=== Bot Detection AFTER Structural Evasion ===")
print("Accuracy:", accuracy_score(y_test_ev, y_pred_ev))
print(classification_report(y_test_ev, y_pred_ev))
```

```
=== Bot Detection AFTER Structural Evasion ===
Accuracy: 0.96
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        96
           1       0.00      0.00      0.00         4

    accuracy                           0.96       100
   macro avg       0.48      0.50      0.49       100
weighted avg       0.92      0.96      0.94       100
```

9. Graph Poisoning Attack

Explanation: Bots create many new connections among themselves and to normal users, plus random noise edges in the graph, heavily altering network structure.

[17]:
```
G_poison = G.copy()
# Bots connect with each other and extra random nodes
for bot in bot_nodes:
    # Connect to 3 random bots
    bot_partners = np.random.choice(bot_nodes, 3, replace=False)
    for b in bot_partners:
        if bot != b:
            G_poison.add_edge(bot, b)
    # Connect to 5 normal nodes
    normal_nodes = [n for n in G.nodes() if n not in bot_nodes]
    normal_partners = np.random.choice(normal_nodes, 5, replace=False)
    for n in normal_partners:
        G_poison.add_edge(bot, n)
# Add 30 extra random edges between normal nodes
normal_nodes = [n for n in G.nodes() if n not in bot_nodes]
for _ in range(30):
    u, v = np.random.choice(normal_nodes, 2, replace=False)
    G_poison.add_edge(u, v)
```

10. Features After Poisoning Attack

Explanation: Recalculate all features after the poisoning attack for classifier retraining.

[19]:
```
degree_p = dict(G_poison.degree())
clustering_p = nx.clustering(G_poison)
betweenness_p = nx.betweenness_centrality(G_poison)
communities_p = list(label_propagation_communities(G_poison))
community_map_p = {n: i for i, c in enumerate(communities_p) for n in c}

features_p = pd.DataFrame({
    "node": list(G_poison.nodes()),
    "degree": [degree_p[n] for n in G_poison.nodes()],
    "clustering": [clustering_p[n] for n in G_poison.nodes()],
    "betweenness": [betweenness_p[n] for n in G_poison.nodes()],
    "community": [community_map_p[n] for n in G_poison.nodes()],
})
features_p["label"] = features["node"].map(labels)
```

11. Train Classifier After Poisoning Attack

Explanation: Retrain the classifier on the poisoned graph to see the performance drop caused by heavy structural changes.

21]:
```
X_p = features_p[["degree", "clustering", "betweenness", "community"]]
y_p = features_p["label"]

X_train_p, X_test_p, y_train_p, y_test_p = train_test_split(X_p, y_p, test_size=0.3, random_state=42)

clf_p = RandomForestClassifier(n_estimators=100, random_state=42)
clf_p.fit(X_train_p, y_train_p)
y_pred_p = clf_p.predict(X_test_p)

print("=== Bot Detection AFTER Graph Poisoning ===")
print("Accuracy:", accuracy_score(y_test_p, y_pred_p))
print(classification_report(y_test_p, y_pred_p))
```

```
=== Bot Detection AFTER Graph Poisoning ===
Accuracy: 0.98
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        96
           1       1.00      0.50      0.67         4

    accuracy                           0.98       100
   macro avg       0.99      0.75      0.83       100
weighted avg       0.98      0.98      0.98       100
```

12. Visualize Graph Changes

Explanation: Compare graph structure visually before attacks, after structural evasion, and after poisoning. Bots are red; normal nodes are blue.

[23]:
```
plt.figure(figsize=(14,6))
pos = nx.spring_layout(G, seed=42)

# Before attack
plt.subplot(1,3,1)
nx.draw(G, pos, node_color=["red" if n in bot_nodes else "skyblue" for n in G.nodes()], node_size=40)
plt.title("Original Graph")

# After Structural Evasion
plt.subplot(1,3,2)
nx.draw(G_evasion, pos, node_color=["red" if n in bot_nodes else "skyblue" for n in G_evasion.nodes()], node_size=40)
plt.title("After Evasion Attack")

# After Poisoning
plt.subplot(1,3,3)
nx.draw(G_poison, pos, node_color=["red" if n in bot_nodes else "skyblue" for n in G_poison.nodes()], node_size=40)
plt.title("After Poisoning Attack")

plt.show()
```