Bot Detection Under Adversarial Attacks on Facebook Egonet Graph

---

1. Introduction

Social networks contain large numbers of real users and automated accounts (bots). Detecting these bots is essential for protecting online communities from misinformation, spam, and coordinated manipulation.

In this assignment, we analyze the Facebook "Ego Network" dataset from SNAP. We build a graph-based bot detection model, apply two adversarial attacks—Structural Evasion and Graph Poisoning—and evaluate their impact on the classifier and the graph structure.

---

2. Dataset Description

Facebook Egonet Dataset
Source: Stanford SNAP
Format: .edges file
Nodes = **4039**
Edges = **88,234**

Each edge represents a friendship relation.

Node attributes are not provided → all features must be extracted from the graph structure.

---

3. Graph Construction

- Loaded .edges file using Network X to build an **undirected graph**.
- Nodes = users, Edges = friendships
- Graph is connected and exhibits **small-world behavior**.

---

4. Feature Extraction

To train a machine learning classifier, we extracted structural features for every node in the graph. These features quantify how each user behaves in the network and help distinguish real users from bot-like anomalies.

Extracted Features:

- **Degree** – Number of direct connections for each node.
- **Clustering Coefficient** – Measures how tightly the node's neighbors are connected.
- **Betweenness Centrality** – Measures how often a node lies on shortest paths between other nodes.
- **Community Label** – Assigned using the **Louvain community detection algorithm**.

computed features:

node	degree	clustering	betweenness	community
0	0	347	0.041962	1.463059e-01	0
1	1	17	0.419118	2.783274e-06	0
2	2	10	0.888889	7.595021e-08	0
3	3	17	0.632353	1.685066e-06	0
4	4	10	0.866667	1.840332e-07	0

---

6. Methodology (Step-by-Step Implementation)

This section describes the exact steps performed in the experiment, following the same order used in the implementation code. Each step corresponds to a major stage in the bot-detection pipeline.

Step 1 – Downloading the Facebook Egonet Dataset

- Used urllib.request.urlretrieve() to automatically download the file facebook_combined.txt from the SNAP repository.
- The dataset represents an undirected friendship network where each line contains one edge:
  node_u node_v
- Saved the dataset locally to enable repeated experiments without re-downloading.

Purpose: Obtain the raw Facebook ego network for further preprocessing.

Step 2 – Constructing the Graph with Network X

- Loaded the dataset using nx.read_edgelist()
- Built an undirected graph G.
- Verified:
  - Number of nodes = 4039
  - Number of edges = 88,234
- Checked connectivity and graph summary statistics.

Purpose: Convert raw text edges into a usable graph data structure.

Step 3 – Computing Structural Features for Each Node

For every node in the graph, the following features were computed:

1. Degree
2. nx.degree(G)

Measures number of immediate friends each user has.

3. Clustering Coefficient
4. nx.clustering(G)

Shows how strongly the user's neighborhood is interconnected.

5. Betweenness Centrality
6. nx.betweenness_centrality(G)

Measures the importance of a user in connecting paths in the network.

7. Community Detection (Louvain Algorithm)
8. community_louvain.best_partition(G)

Assigns each user to a community based on modularity optimization.

- Stored all extracted features in a Pandas DataFrame.

Purpose: Encode graph structure into numerical features suitable for machine learning.

Step 4 – Assigning Bot vs. Real User Labels

- Because the dataset contains no ground-truth labels, synthetic bot labels were generated for experimentation.
- Randomly selected 10% of the nodes (403 nodes) as bots:
- bot_nodes = random.sample(G.nodes(), 403)
- Added a new column label where:
  - 1 = bot
  - 0 = real user

Purpose: Create a binary classification task to evaluate robustness against attacks.

Step 5 – Splitting Data into Train/Test Sets

- Split features & labels using train_test_split (70% train, 30% test).
- Ensured stratification so the proportion of bots remains consistent.

Purpose: Prepare data for supervised learning while preventing data leakage.

Step 6 – Training the Baseline Random Forest Classifier

- Trained a baseline model using:
- RandomForestClassifier(n_estimators=200)
- Evaluated using accuracy & classification report.

Purpose: Establish a reference performance before applying adversarial attacks.

Step 7 – Structural Evasion Attack Simulation

To simulate evasion:

- Increased the degree of bot nodes by connecting them to high-degree real users.
- Reduced their clustering coefficients by adding long-range edges.
- Recomputed all features for the modified graph:
  degree_evasion, clustering_evasion,……
- Re-labeled the same bot nodes.
- Trained a new model on the modified features.

Purpose: Make bots mimic normal users through structural manipulation.

Step 8 – Graph Poisoning Attack

This attack modifies the *training data*, not the graph structure.

Procedure:

1. Added 200 synthetic bot-like nodes to the graph.
2. Connected them in a way that mimics real users (high clustering, mid-degree).
3. Recomputed features for the expanded graph.
4. The new dataset includes:
    - Original nodes
    - Poisoned nodes
5. Trained the classifier on this poisoned dataset.

Purpose: Teach the model false patterns so it misclassifies real bots.

Step 9 – Testing the Poisoned Model on the Clean Graph

- Took the model trained on poisoned data.
- Evaluated it on the original clean graph features.
- Measured generalization under adversarial contamination.

Purpose: Assess whether poisoning causes long-term classifier degradation.

Step 10 – Visualization and Graph Plotting

Generated the following plots:

1. Original Graph using nx.draw()
2. Structural Evasion Graph
3. Graph Poisoning Graph
4. Accuracy Comparison Bar Chart
5. Optional distribution plots of degree/clustering before & after attacks.

Purpose: Visually inspect how each attack modifies the network.

5. Experimental Setup

We applied **Random Forest Classifier** to detect bots based on graph features.
The experiment includes four main conditions:

Baseline (No Attack)

- Model trained on clean data extracted from the original graph.
- Serves as reference for comparison.

Structural Evasion Attack

Attackers modify connections between nodes to make bots mimic normal user behavior.
Examples:

- Adding edges between bot nodes and high-degree real users.
- Reducing clustering around bot nodes.

This changes the graph topology without altering labels.

Graph Poisoning Attack

The training data is corrupted by injecting adversarial bot nodes that deliberately mimic "benign-like" structures, causing the classifier to learn misleading patterns.

Poisoned Model Tested on Clean Graph

The model is trained on poisoned data but tested on the original clean graph to evaluate generalization.

---

7. Results & Analysis

1. Baseline Bot Detection Performance

Class	Precision	Recall	F1-score	Support
Real	0.90	0.98	0.94	1096

Bot	0.11	0.02	0.03	116
Accuracy	0.8919			1212

Observation:

- The model is biased toward real users due to the small proportion of bots (10%).
- Overall accuracy is acceptable, but bot detection is extremely low.

---

2. Structural Evasion Attack

Class	**Precision**	**Recall**	**F1-score**	**Support**
Real	1.00	1.00	1.00	1096
Bot	0.96	0.99	0.97	116
Accuracy	0.9950			1212

Observation:

- The attack connected bots to high-degree real users.
- Feature distributions (degree, betweenness) changed → bots became easier to detect.
- Accuracy significantly improved compared to the baseline.

---

3. Graph Poisoning Attack

Class	**Precision**	**Recall**	**F1-score**	**Support**
Real	0.90	0.99	0.94	1093
Bot	0.90	0.51	0.65	240
Accuracy	0.9017			1333

Observation:

- Adding fake bot nodes distorted the natural graph structure.

- The classifier learned incorrect decision boundaries → bot recall decreased.
- Poisoning reduces the model's generalization ability.

---

4. Poisoned Model Tested on Clean Graph

Class	Precision	Recall	F1-score	Support
Real	0.91	0.99	0.95	3636
Bot	0.58	0.08	0.14	403
Accuracy	0.9024			4039

Observation:

- The model trained on poisoned data fails to detect bots on the clean graph.
- Demonstrates the severe impact of data poisoning on generalization.

---

5. Performance Comparison Table

Condition	Accuracy
Baseline	0.892
Structural Evasion	0.995
Graph Poisoning	0.902
Poisoned → Clean Test	0.902

Observation:

- Structural evasion unexpectedly improves detection performance.
- Graph poisoning decreases robustness and harms classifier stability.

---

8. Figures / Visualization

1. **Figure 1 – Original Graph**
    - Spring layout visualization of the clean Facebook Egonet.
    - Shows natural community clusters.
2. **Figure 2 – Structural Evasion Graph**
    - Bots connected to high-degree real users.
    - Graph becomes slightly denser; bots move closer to cluster centers.
3. **Figure 3 – Graph Poisoning Graph**
    - Many fake nodes introduced.
    - Community boundaries are distorted; connectivity is noisier.
4. **Figure 4 – Accuracy Comparison Chart**
    - Bar chart showing accuracies across all four scenarios.
    - Highlights the reduction of robustness due to poisoning.
5. **Figure 5 (Optional)**
    - Distributions of degree, clustering coefficient, and betweenness before and after attacks.

---

9. Analysis & Discussion

- **Structural Evasion Attack:**
    - Bots gained additional edges → appeared more “human-like.”
    - Increased centrality inadvertently → easier to detect.
    - Result: Accuracy improved over baseline.
- **Graph Poisoning Attack:**
    - Introduced artificial nodes → classifier learned misleading patterns.
    - Result: Bot recall decreased; model generalization reduced.
- **Poisoned Model on Clean Data:**
    - Shows almost complete inability to detect bots.

- Confirms that poisoning severely weakens model robustness.

---

10. Conclusion

- The baseline classifier performs reasonably well on clean data.
- Structural evasion unexpectedly improves detection performance.
- Graph poisoning severely damages model integrity and harms generalization.
- Poisoned model fails on clean data (only 8% recall for bots).
- **Key takeaway:** Adversarial attacks can significantly affect graph-based bot detection systems, with poisoning attacks posing the greatest risk.