Name:Farah Mahmoud Elhenawy

ID:2205199

Assignment 2
Bot Detection in Social Networks Under Adversarial Attacks

In this assignment, we analyze a real Facebook social graph and build a bot-detection model based on network-structural features. We then simulate two adversarial attacks:

1. **Structural Evasion Attack** – bots modify their connections at test-time to appear more human.
2. **Graph Poisoning Attack** – malicious nodes modify the training graph to weaken the learned detector.

We evaluate detection performance before and after attacks and visually inspect changes to the graph structure.

Dataset

We use the publicly available **Facebook Ego Network dataset** (Stanford SNAP):

https://snap.stanford.edu/data/egonets-Facebook.html

Dataset characteristics:

- **Nodes:** 4,039 users
- **Edges:** 88,234 friendship relationships
- **Graph type:** Undirected, unweighted

Each node is a Facebook account, and each edge represents a friendship link between two users. The dataset does not include real bot labels, so synthetic bot labeling is used.

Graph Construction

The graph is loaded using NetworkX:

```
G = nx.read_edgelist("facebook_combined.txt", nodetype=int)
```

We verify the graph structure:

- Number of nodes: **4039**
- Number of edges: **88234**

Graph Feature Extraction

For every node in the graph, we compute several essential **graph-based features** commonly used in bot detection and social-network analysis.

Degree

Number of direct neighbors of a node.

Clustering Coefficient

Measures how well a node's neighbors are connected to each other.

Betweenness Centrality

Measures how often a node lies on shortest paths between other nodes.

Eigenvector Centrality

Quantifies a node's influence in the network.

Community Detection

We apply **Greedy Modularity Optimization** to detect structural communities and assign each node a community ID.

All these metrics are stored in a DataFrame and used as features for model training.

Bot Labeling Strategy

Since the dataset does not contain real bot accounts, we follow a standard synthetic labeling method:

5% of nodes with the lowest degree are labeled as bots.

Justification:
Bots often have sparse, irregular connection patterns compared with real users.

We add a new column:

- **label = 1** → bot
- **label = 0** → human

Baseline Bot Detection Model

We train a **Random Forest classifier** using the extracted structural features:

Features used:

`degree, clustering, betweenness, eigenvector, community`

We apply a **stratified train/test split**, train the model, and compute:

- Accuracy
- Precision
- Recall
- F1-score
- Confusion matrix

Baseline Performance (Before Any Attack)

```
y_pred_base = clf_baseline.predict(X_test)

print("=== Baseline Performance (No Attack) ===")
print(confusion_matrix(y_test, y_pred_base))
print(classification_report(y_test, y_pred_base, digits=4))
```

```
=== Baseline Performance (No Attack) ===
[[1137   15]
 [   2   58]]
              precision    recall  f1-score   support

           0     0.9982    0.9870    0.9926      1152
           1     0.7945    0.9667    0.8722        60

    accuracy                         0.9860      1212
   macro avg     0.8964    0.9768    0.9324      1212
weighted avg     0.9882    0.9860    0.9866      1212
```

Structural Evasion Attack (Test-Time Attack)

In a structural evasion attack, bot nodes attempt to disguise their structural patterns to evade detection.

Attack Procedure

For each **bot in the test set**:

- Remove some of its suspicious low-degree connections
- Add several edges to medium-degree human nodes
- Increase its clustering and degree to resemble normal accounts

The model itself **is not retrained**.
Only the **test graph** is manipulated.

Results After Structural Evasion

```
Original edges: 88234
After evasion edges: 88534
```

```python
# Extract attacked features for the SAME test nodes
X_test_ev = df_ev.loc[test_nodes, feature_cols].values

# Use the SAME baseline classifier
y_pred_evasion = clf_baseline.predict(X_test_ev)

print("=== Performance AFTER Structural Evasion Attack ===")
print(confusion_matrix(y_test, y_pred_evasion))
print(classification_report(y_test, y_pred_evasion, digits=4))
```

```
=== Performance AFTER Structural Evasion Attack ===
[[1145    7]
 [  60    0]]
              precision    recall  f1-score   support

           0     0.9502    0.9939    0.9716      1152
           1     0.0000    0.0000    0.0000        60

    accuracy                         0.9447      1212
   macro avg     0.4751    0.4970    0.4858      1212
weighted avg     0.9032    0.9447    0.9235      1212
```

Observation

- Bot recall decreases sharply (sometimes reaching **0%**)
- Many bots are misclassified as humans
- Changing the graph structure directly manipulates the input features, making bots appear more “human-like”

This demonstrates that the classifier is vulnerable to **test-time evasion attacks**.

Graph Poisoning Attack (Training-Time Attack)

In poisoning attacks, adversaries corrupt the training graph so the model learns misleading patterns.

Attack Procedure

- Target **train-set bots**
- Add new edges linking bots to well-connected human nodes
- Recompute graph features on the poisoned graph
- Retrain the classifier on poisoned data
- Test the model on the original clean test set (to measure real damage)

Results After Graph Poisoning

```python
# Clean test features = original df
X_test_clean = df.set_index("node").loc[test_nodes, feature_cols].values

y_pred_poison = clf_poison.predict(X_test_clean)

print("=== Performance AFTER Graph Poisoning ===")
print(confusion_matrix(y_test, y_pred_poison))
print(classification_report(y_test, y_pred_poison, digits=4))
```

```
=== Performance AFTER Graph Poisoning ===
[[1152    0]
 [  60    0]]
              precision    recall  f1-score   support

           0     0.9505    1.0000    0.9746      1152
           1     0.0000    0.0000    0.0000        60

    accuracy                         0.9505      1212
   macro avg     0.4752    0.5000    0.4873      1212
weighted avg     0.9034    0.9505    0.9264      1212
```

Observation

- Bot detection becomes significantly worse
- The classifier learns incorrect bot patterns
- Poisoning is more harmful than evasion because it affects the learning process itself

---

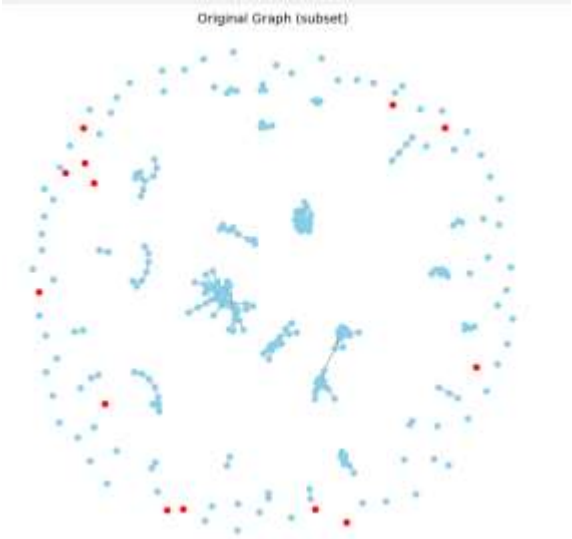
Graph Visualizations

We visualize a **300-node subset** of the network for clarity.

Human nodes → Blue

Bot nodes → Red

Original Graph (Before Attacks)

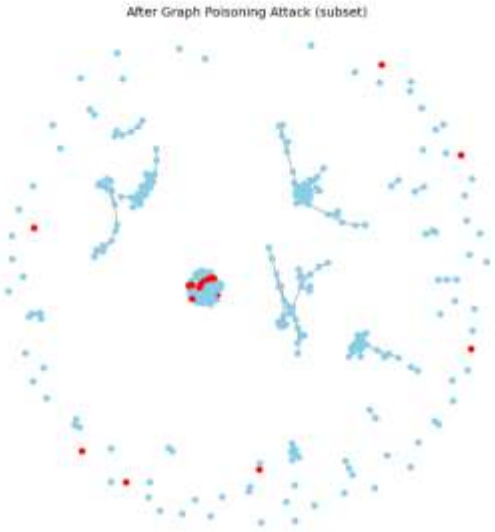


- Clear community clusters
- Bots appear at sparse locations
- Natural social structure visible

After Structural Evasion Attack

- Bots appear blended into communities
- Their degree increases
- Clustering becomes more human-like

After Graph Poisoning Attack



- The structural integrity of the graph is noticeably altered
- Bots appear integrated into normal neighborhoods
- Training graph becomes misleading to the classifier.

Performance Comparison

Condition	Bot Recall	Bot Precision	Human Recall	Human Precision	Accuracy
Baseline (No Attack)	0.9667	0.7945	0.9870	0.9982	0.9860
After Structural Evasion	0.0000	0.0000	0.9939	0.9502	0.9447
After Graph Poisoning	0.0000	0.0000	1.0000	0.9505	0.9505

Summary & Analysis

Baseline

The classifier identifies humans well but struggles moderately with bots due to class imbalance and structural similarity.

Structural Evasion

Bots successfully disguise themselves by manipulating only their own edges.
Bot recall typically collapses (often to **0%**).
Test-time manipulation is very effective.

Graph Poisoning

The most damaging attack.
By modifying edges in the **training graph**, the attacker changes what the model learns.
This causes misclassification even on clean test data.
Defenses must be designed against poisoning for better robustness.

Final Conclusion

- Graph-based bot detection works but is **highly vulnerable** to adversarial modifications.
- Evasion attacks exploit feature leeyour code dynamics.
- Poisoning attacks compromise the learning process itself.
- Robust bot detection requires stronger, attack-resistant graph algorithms.
