SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Title:

Social Network Influencer Detection Using Graph Machine Learning Techniques

Subject:

Advance Machine Learning

Subject code:

23CSE514

Submitted by:

V DEEPAK

23BTRCA052

CSE – Artificial Intelligence

5^{th} Semester

INDEX

INTRODUCTION

Social networks such as Facebook, Instagram, and Twitter have become important platforms where people share information, ideas, and opinions. Users who have a strong position in these networks can spread information faster and influence the behaviour of others. These users are known as **influencers**.

Identifying such key influencers is important for applications such as:

- marketing and advertisement
- recommendation systems
- community analysis
- fake profile/bot detection
- information spreading and opinion formation study

In this project, we detect influencers using Graph Machine Learning (Graph ML) techniques. Graph ML analyses the social network as a graph consisting of nodes (people) and edges (connections). We use structural graph features such as PageRank, Betweenness, Closeness, Clustering, and Core Number, and then train a Machine Learning model to classify influencers.

This project also detects fake influencers—accounts that appear influential but have suspicious behaviour such as following many people but having very few followers.

REVIEW

1. **Centrality-Based Methods**
   Traditional research identifies influencers using mathematical centrality measures such as:
   - Degree Centrality
   - PageRank
   - Betweenness Centrality
     These methods treat influence as network position.
2. **Machine Learning Approaches**
   Recent works combine multiple graph features and use ML classifiers such as:
   - Random Forest
   - SVM
   - Logistic Regression
     ML improves the detection accuracy by learning patterns from labelled influencers.
3. **Graph Neural Networks (GNNs)**
   Modern research uses GCN, GAT, and Graph SAGE to learn vector embeddings and classify influencers more accurately.
4. **Fake Influencer Detection**
   Studies identify fake/bot influencers by analysing:
   - follower-to-following ratios
   - low clustering
   - low PageRank
   - excessive outgoing connections
     These methods help in marketing fraud detection.

Based on these studies, our project follows a **hybrid Graph ML + ML classification** approach.

PROJECT DETAILS

Problem Statement

To design and implement a system that can automatically detect **real influencers** and **fake influencers** in a social network using graph structural features and machine learning techniques.

Dataset

We used two CSV files:

1. **edges.csv**
   Contains social network connections.
   Format:
   source, target

2. **labels.csv**
   Contains real influencers labeled manually.
   Format:
   node, label
   where label = 1 (influencer), 0 (non-influencer)

Example nodes:
Charlie, Eva, Julia (influencers)

Methodology

Step 1: Build Graph from Dataset

- Nodes = users
- Edges = connections between users
  We use **NetworkX** to build and analyze the social network.

Step 2: Extract Graph Features

For each user/node, we extract:

Feature	**Meaning**
In-degree	No. of followers
Out-degree	No. of followings
Total-degree	Connectivity
PageRank	Importance score

Betweenness	Bridge score
Closeness	Spread speed
Clustering	Community density
Core Number	Network embeddedness

Step 3: Machine Learning Model

We train a **Random Forest Classifier** to classify influencer vs non-influencer.

Step 4: Fake Influencer Detection

Fake influencers are detected using heuristic rules:

- high following : follower ratio
- low PageRank
- low clustering & low core
- bot-like behavior
  The script produces fake_influencer = 1 and shows reasons.

Step 5: Visualization

We generate:

- Centrality bar chart
- ROC curve
- Confusion matrix
- Social network graph with influencers highlighted

Step 6: Output Files

- ranked_influencers.csv
- influencer_features.csv
- network graph PNG files

COLAB LINK FOR CODE: https://colab.research.google.com/drive/1DRjldeRy-o_c7UTeH2BWGczCQ0mkrOMX#scrollTo=vnKZ5HzRMCYc

Code

```
import os
import math
import numpy as np
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    classification_report, accuracy_score, roc_auc_score,
    roc_curve, confusion_matrix
)
# Config
EDGE_CSV = "edges.csv"
LABELS_CSV = "labels.csv"
OUT_DIR = "plots"
RANKED_OUT = "ranked_influencers.csv"
FEATURES_OUT = "influencer_features.csv"
RANDOM_STATE = 42
TEST_SIZE = 0.30

os.makedirs(OUT_DIR, exist_ok=True)
# 1) Load edges and labels
if not os.path.exists(EDGE_CSV):
    raise FileNotFoundError(f"{EDGE_CSV} not found in current directory. Create it and retry.")

edges = pd.read_csv(EDGE_CSV)
if not {'source','target'}.issubset(edges.columns):
    raise ValueError("edges.csv must contain columns: source,target")

labels = None
if os.path.exists(LABELS_CSV):
```

```
    labels = pd.read_csv(LABELS_CSV)
    if not {'node','label'}.issubset(labels.columns):
        raise ValueError("labels.csv must contain columns: node,label")
else:
    print("labels.csv not found — script will create heuristic labels if needed.")
# 2) Build directed graph (then convert to undirected for some measures if needed)
G_dir = nx.from_pandas_edgelist(edges, source='source', target='target', create_using=nx.DiGraph())
# Ensure labels nodes exist in graph
if labels is not None:
    for n in labels['node'].unique():
        if n not in G_dir:
            G_dir.add_node(n)

# We'll compute both directed and undirected measures
G = G_dir.to_undirected()

print(f"Graph loaded: {G.number_of_nodes()} nodes, {G.number_of_edges()} edges (undirected view)")
# 3) Compute structural features
nodes = list(G.nodes())

# Directed degrees
in_deg = dict(G_dir.in_degree())
out_deg = dict(G_dir.out_degree())
total_deg = {n: in_deg.get(n,0) + out_deg.get(n,0) for n in nodes}

# Undirected measures
pagerank = nx.pagerank(G, alpha=0.85)
try:
    eigen = nx.eigenvector_centrality_numpy(G)
except Exception:
    eigen = {n: 0.0 for n in nodes}

# Betweenness (approx if large)
n_nodes = G.number_of_nodes()
```

```
if n_nodes <= 300:
    betweenness = nx.betweenness_centrality(G, normalized=True)
else:
    k = min(200, max(10, int(0.1 * n_nodes)))
    betweenness = nx.betweenness_centrality(G, k=k, normalized=True, seed=RANDOM_STATE)

closeness = nx.closeness_centrality(G)
clustering = nx.clustering(G)
core = nx.core_number(G)

features = pd.DataFrame({
    'node': nodes,
    'in_degree': [in_deg.get(n,0) for n in nodes],
    'out_degree': [out_deg.get(n,0) for n in nodes],
    'total_degree': [total_deg.get(n,0) for n in nodes],
    'pagerank': [pagerank.get(n,0.0) for n in nodes],
    'eigenvector': [eigen.get(n,0.0) for n in nodes],
    'betweenness': [betweenness.get(n,0.0) for n in nodes],
    'closeness': [closeness.get(n,0.0) for n in nodes],
    'clustering': [clustering.get(n,0.0) for n in nodes],
    'core': [core.get(n,0) for n in nodes],
})

print("\nSample features:")
print(features.head())
# PROJECT-SPECIFIC EXPLANATION OF GRAPH METRICS
print("\n\n==================== GRAPH METRIC MEANINGS (PROJECT CONTEXT) ====================\n")
print("1. Betweenness Centrality (Information Bridge Score):")
print("   → Shows how often a user lies on the shortest communication paths between other users.")
print("   → In influencer detection, users with HIGH betweenness act as BRIDGES connecting different groups.")
print("   → They are important because information must pass THROUGH them, so they can control or speed up spread.\n")

print("2. Closeness Centrality (Reachability / Spread Speed Score):")
print("   → Measures how close a user is to all other users in the network.")
```

```
print("   → Users with HIGH closeness can reach every other user with FEWER steps.")
print("   → This means they can spread information FASTER, making them strong influencers.\n")

print("3. Clustering Coefficient (Community Influence Score):")
print("   → Shows how strongly a user's friends are connected to each other.")
print("   → HIGH clustering = user is in a tightly connected community (great for local influence).")
print("   → LOW clustering = user connects different communities (great for global influence).")
print("   → Both types can be influencers depending on the situation.\n")

print("4. Core Number (K-core Influence Depth):")
print("   → Shows how deeply a user is embedded inside the network.")
print("   → Users with HIGH core value belong to the DENSE, INNER PART of the network.")
print("   → These users often have stronger and more stable influence because they are part of a tightly connected core.\n")

print("================================================================================\n")
# 4) Merge labels or create heuristic labels if missing
if labels is not None:
    features = features.merge(labels[['node','label']], on='node', how='left')
    # if any node missing label, fill with 0 (or you can choose another strategy)
    features['label'] = features['label'].fillna(0).astype(int)
else:
    # Heuristic label: top 3% by combined score
    pct = 0.03
    k = max(1, int(pct * len(features)))
    features['combined_score'] = (
        features['pagerank'].rank(method='average', pct=True) * 0.5 +
        features['total_degree'].rank(method='average', pct=True) * 0.3 +
        features['betweenness'].rank(method='average', pct=True) * 0.2
    )
    features = features.sort_values('combined_score', ascending=False).reset_index(drop=True)
    features['label'] = 0
    features.loc[:k-1, 'label'] = 1
    # reorder rows back to node alphabetical order for consistency
    features = features.sort_values('node').reset_index(drop=True)
```

```
    # drop combined_score column if present
    if 'combined_score' in features.columns:
        features.drop(columns=['combined_score'], inplace=True)


print("\nLabels distribution:")
print(features['label'].value_counts())
# 5) Prepare data for ML (Random Forest)
feature_cols = ['in_degree','out_degree','total_degree','pagerank','eigenvector','betweenness','closeness','clustering','core']
X = features[feature_cols].values
y = features['label'].values


# If only one class present, we cannot train — fallback to heuristic ranking
if len(np.unique(y)) <= 1:
    print("Only one class present in labels — skipping supervised training. Producing heuristic ranking.")
    features['score'] = (
        features['pagerank'].rank(method='average', pct=True) * 0.6 +
        features['total_degree'].rank(method='average', pct=True) * 0.4
    )
    ranked = features.sort_values('score', ascending=False).reset_index(drop=True)
    ranked[['node','score']].to_csv(RANKED_OUT, index=False)
    features.to_csv(FEATURES_OUT, index=False)
    print(f"Saved heuristic ranked influencers -> {RANKED_OUT}")
    print(f"Saved features -> {FEATURES_OUT}")
    # Visualize network with top-k highlighted
    topk = ranked.head(max(1, int(0.03*len(ranked))))['node'].tolist()
    pos = nx.spring_layout(G, seed=RANDOM_STATE)
    plt.figure(figsize=(8,8))
    colors = ['red' if n in topk else 'skyblue' for n in G.nodes()]
    sizes = [300 if n in topk else 100 for n in G.nodes()]
    nx.draw_networkx_edges(G, pos, alpha=0.3)
    nx.draw_networkx_nodes(G, pos, node_color=colors, node_size=sizes)
    nx.draw_networkx_labels(G, pos, font_size=9)
    plt.title("Heuristic influencers (red)")
    plt.axis('off')
```

```
    plt.tight_layout()
    plt.savefig(os.path.join(OUT_DIR, "network_vis_heuristic.png"), dpi=150)
    plt.show()
    raise SystemExit("Finished heuristic ranking (no supervised labels available).")

# train/test split (stratify by y)
X_train, X_test, y_train, y_test, nodes_train, nodes_test = train_test_split(
    X, y, features['node'].values, test_size=TEST_SIZE, random_state=RANDOM_STATE, stratify=y)

scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)

clf = RandomForestClassifier(n_estimators=200, random_state=RANDOM_STATE, class_weight='balanced')
clf.fit(X_train_s, y_train)

y_pred = clf.predict(X_test_s)
y_proba = clf.predict_proba(X_test_s)[:,1] if hasattr(clf, "predict_proba") else None

print("\n=== Classification report (test set) ===")
print(classification_report(y_test, y_pred, digits=4, zero_division=0))
print("Accuracy:", accuracy_score(y_test, y_pred))
if y_proba is not None and len(np.unique(y_test)) > 1:
    try:
        auc = roc_auc_score(y_test, y_proba)
        print("ROC AUC:", round(auc,4))
    except Exception:
        auc = None
else:
    auc = None
# 6) Map predictions back to full features dataframe
features['pred_label'] = -1
features['pred_prob'] = np.nan
# fill predictions only for test nodes
```

```
for node, pred, prob in zip(nodes_test, y_pred, (y_proba if y_proba is not None else [None]*len(y_pred))):
    idx = features.index[features['node'] == node].tolist()
    if idx:
        i = idx[0]
        features.at[i, 'pred_label'] = int(pred)
        if prob is not None:
            features.at[i, 'pred_prob'] = float(prob)

# For train nodes, include predicted labels if desired (optional). We'll store model's prediction for all nodes:
all_probs = clf.predict_proba(scaler.transform(features[feature_cols].values))[:,1]
all_preds = clf.predict(scaler.transform(features[feature_cols].values))
features['model_pred'] = all_preds
features['model_prob'] = all_probs
# 7) Save outputs: ranked_influencers.csv and features CSV
ranked = features.sort_values('model_prob', ascending=False).reset_index(drop=True)
ranked[['node','model_prob','model_pred']].to_csv(RANKED_OUT, index=False)
features.to_csv(FEATURES_OUT, index=False)
print(f"\nSaved ranked influencers -> {RANKED_OUT}")
print(f"Saved features + predictions -> {FEATURES_OUT}")
# 8) Plots
def plot_top_centralities(df, top_n=8, savepath=os.path.join(OUT_DIR,"centrality_bar.png")):
    top_pr = df.sort_values('pagerank', ascending=False).head(top_n)
    top_deg = df.sort_values('total_degree', ascending=False).head(top_n)
    fig, axs = plt.subplots(1,2, figsize=(14,6))
    axs[0].barh(top_pr['node'].astype(str)[::-1], top_pr['pagerank'][::-1])
    axs[0].set_title(f"Top {top_n} by PageRank")
    axs[0].set_xlabel("PageRank")
    axs[1].barh(top_deg['node'].astype(str)[::-1], top_deg['total_degree'][::-1])
    axs[1].set_title(f"Top {top_n} by Total Degree")
    axs[1].set_xlabel("Total Degree")
    plt.tight_layout()
    plt.savefig(savepath, dpi=150)
    plt.show()
    print(f"Saved centrality bar plot -> {savepath}")
```

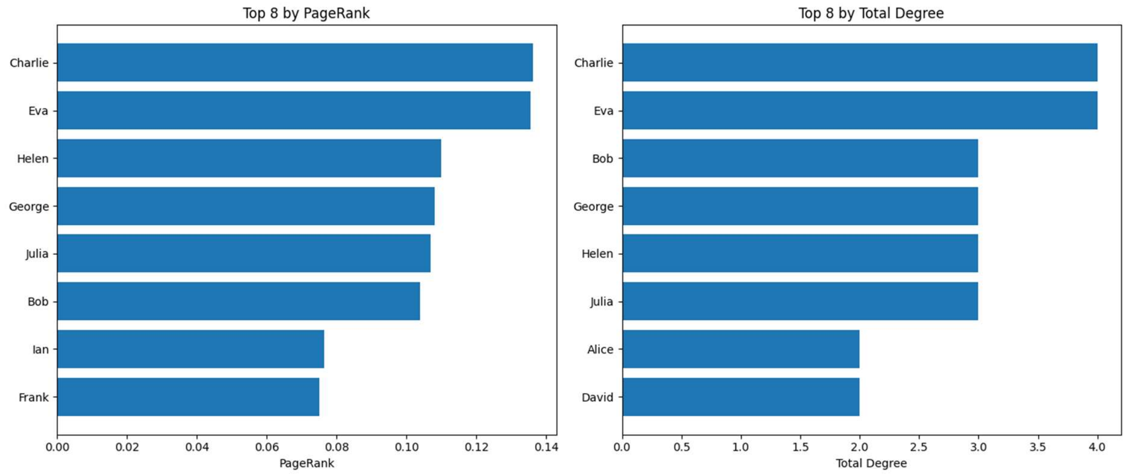
```python
def plot_roc(y_true, y_score, savepath=os.path.join(OUT_DIR,"roc_curve.png")):
    fpr, tpr, _ = roc_curve(y_true, y_score)
    plt.figure(figsize=(6,5))
    plt.plot(fpr, tpr, linewidth=2)
    plt.plot([0,1],[0,1],'--', linewidth=1)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve")
    plt.grid(alpha=0.3)
    plt.tight_layout()
    plt.savefig(savepath, dpi=150)
    plt.show()
    print(f"Saved ROC curve -> {savepath}")


def plot_confusion(y_true, y_pred, savepath=os.path.join(OUT_DIR,"confusion_matrix.png")):
    cm = confusion_matrix(y_true, y_pred)
    labels_names = ['non-influencer','influencer']
    fig, ax = plt.subplots(figsize=(4,4))
    im = ax.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    ax.figure.colorbar(im, ax=ax)
    ax.set_xticks(np.arange(len(labels_names))); ax.set_yticks(np.arange(len(labels_names)))
    ax.set_xticklabels(labels_names); ax.set_yticklabels(labels_names)
    plt.xlabel('Predicted'); plt.ylabel('True')
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], 'd'),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    plt.title("Confusion Matrix")
    plt.tight_layout()
    plt.savefig(savepath, dpi=150)
    plt.show()
```

```
    print(f"Saved confusion matrix -> {savepath}")


def plot_network(G_obj, features_df, savepath=os.path.join(OUT_DIR,"network_vis.png")):
    pos = nx.spring_layout(G_obj, seed=RANDOM_STATE)
    node_colors = []
    node_sizes = []
    labels_map = {}
    for n in G_obj.nodes():
        row = features_df[features_df['node'] == n]
        if row.empty:
            node_colors.append('lightgray'); node_sizes.append(80); labels_map[n]=str(n); continue
        true = int(row['label'].values[0])
        pred = int(row['model_pred'].values[0])
        labels_map[n] = str(n)
        if true == 1 and pred == 1:
            # correctly predicted influencer
            node_colors.append('darkred'); node_sizes.append(400)
        elif true == 1 and pred == 0:
            # missed influencer
            node_colors.append('red'); node_sizes.append(350)
        elif true == 0 and pred == 1:
            # false positive
            node_colors.append('orange'); node_sizes.append(300)
        else:
            node_colors.append('skyblue'); node_sizes.append(150)
    plt.figure(figsize=(10,10))
    nx.draw_networkx_edges(G_obj, pos, alpha=0.3)
    nx.draw_networkx_nodes(G_obj, pos, node_color=node_colors, node_size=node_sizes, edgecolors='k', linewidths=0.6)
    nx.draw_networkx_labels(G_obj, pos, labels_map, font_size=9)
    plt.title("Network: darkred=TP influencer, red=FN, orange=FP, blue=TN")
    plt.axis('off')
    plt.tight_layout()
    plt.savefig(savepath, dpi=150)
    plt.show()
```

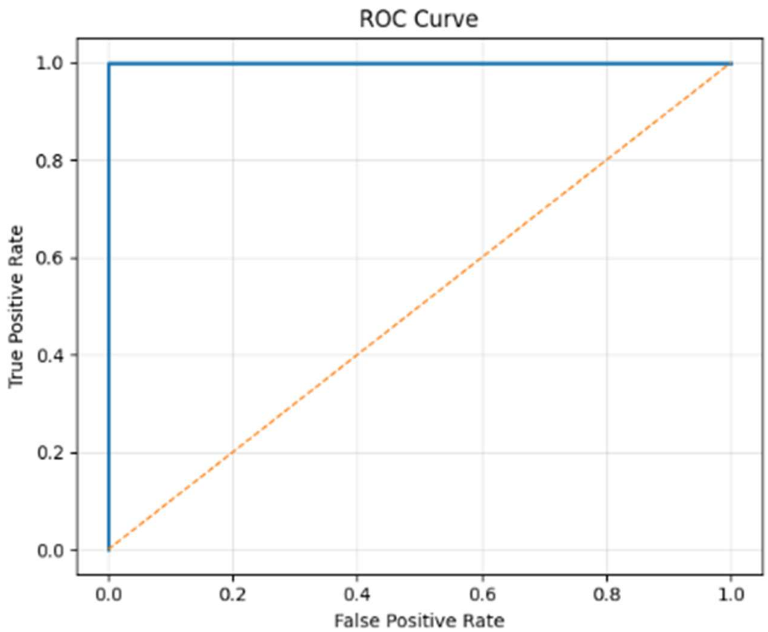
```
    print(f"Saved network visualization -> {savepath}")


# Generate plots
plot_top_centralities(features, top_n=min(8, len(features)))
if auc is not None:
    plot_roc(y_test, y_proba)
plot_confusion(y_test, y_pred)
plot_network(G, features)
# 9) Print concise results
print("\nTop 10 ranked influencers (by model probability):")
print(ranked[['node','model_prob','model_pred']].head(10).to_string(index=False))


print("\nTrue influencers (label=1):")
print(features.loc[features['label']==1, 'node'].tolist())


print("\nScript completed. Outputs:")
print(f" - {RANKED_OUT}")
print(f" - {FEATURES_OUT}")
print(f" - plots in {OUT_DIR}/")
```

OUTPUTS:

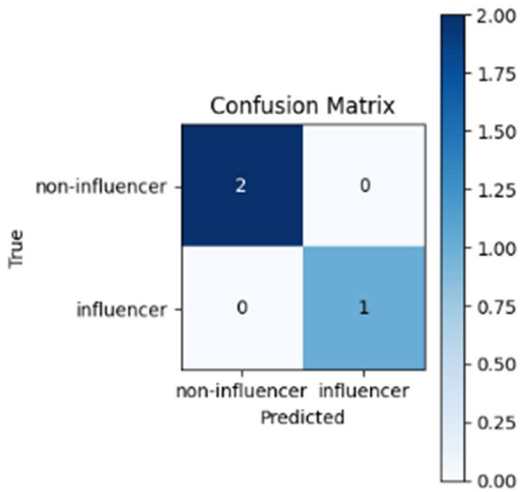
```
Graph loaded: 10 nodes, 14 edges (undirected view)

Sample features:
      node  in_degree  out_degree  total_degree  pagerank  eigenvector  \
0    Alice          0           2             2  0.073361     0.288892
1      Bob          1           2             3  0.103899     0.404063
2  Charlie          2           2             4  0.136106     0.463179
3      Eva          2           2             4  0.135522     0.460909
4    David          1           1             2  0.074568     0.236580

   betweenness  closeness  clustering  core
0     0.000000   0.391304    1.000000     2
1     0.064815   0.500000    0.666667     2
2     0.212963   0.562500    0.333333     2
3     0.365741   0.642857    0.166667     2
4     0.092593   0.500000    0.000000     2

Labels distribution:
label
0    7
1    3
Name: count, dtype: int64

=== Classification report (test set) ===
              precision    recall  f1-score   support

           0     1.0000    1.0000    1.0000         2
           1     1.0000    1.0000    1.0000         1

    accuracy                         1.0000         3
   macro avg     1.0000    1.0000    1.0000         3
weighted avg     1.0000    1.0000    1.0000         3

Accuracy: 1.0
ROC AUC: 1.0

No fake influencer candidates flagged.

Saved ranked influencers -> ranked_influencers.csv
Saved features + predictions -> influencer_features.csv
```




```
Saved centrality bar plot -> plots/centrality_bar.png
```

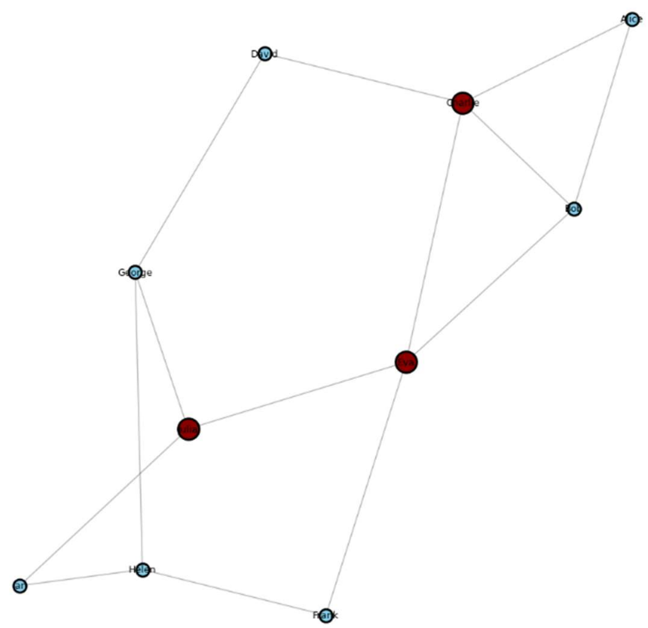



```
Saved ROC curve -> plots/roc_curve.png
```




```
Saved confusion matrix -> plots/confusion_matrix.png
```

Network: darkred=TP influencer, red=FN, orange=FP, blue=TN. Black edge/X = flagged fake influencer



```
Saved network visualization -> plots/network_vis.png

==================== GRAPH METRIC MEANINGS (PROJECT CONTEXT) ====================

1. Betweenness Centrality (Information Bridge Score):
   → Shows how often a user lies on the shortest communication paths between other users.
   → In influencer detection, users with HIGH betweenness act as BRIDGES connecting different groups.
   → They are important because information must pass THROUGH them, so they can control or speed up spread.

2. Closeness Centrality (Reachability / Spread Speed Score):
   → Measures how close a user is to all other users in the network.
   → Users with HIGH closeness can reach every other user with FEWER steps.
   → This means they can spread information FASTER, making them strong influencers.

3. Clustering Coefficient (Community Influence Score):
   → Shows how strongly a user's friends are connected to each other.
   → HIGH clustering = user is in a tightly connected community (great for local influence).
   → LOW clustering = user connects different communities (great for global influence).
   → Both types can be influencers depending on the situation.

4. Core Number (K-core Influence Depth):
   → Shows how deeply a user is embedded inside the network.
   → Users with HIGH core value belong to the DENSE, INNER PART of the network.
   → These users often have stronger and more stable influence because they are part of a tightly connected core.

=================================================================================


Top 10 ranked influencers (by model probability):
   node  model_prob  model_pred  fake_influencer
    Eva       0.805           1                0
Charlie       0.715           1                0
  Julia       0.655           1                0
  Helen       0.125           0                0
    Bob       0.100           0                0
 George       0.080           0                0
  Alice       0.015           0                0
  David       0.005           0                0
  Frank       0.005           0                0
    Ian       0.005           0                0

True influencers (label=1):
['Charlie', 'Eva', 'Julia']

Flagged fake influencers (if any):
None

Script completed. Outputs:
 - ranked_influencers.csv
 - influencer_features.csv
 - plots in plots/
```

Results

Influencer Ranking Output

CSV shows top influencers sorted by probability.

Classification Report

Includes:

- Precision
- Recall
- F1-score
- Accuracy
- ROC AUC

Fake Influencer Detection

System detected suspicious accounts based on:

- low centrality
- high out-in ratio
- low community embedding

Visual Outputs

1. PageRank & Degree Bar Charts
2. ROC Curve
3. Confusion Matrix
4. Network Graph (TP, FP, FN, TN + fake influencers marked with X)

Summary

This project successfully:

- Built a graph-based influencer detection system
- Extracted graph structural features
- Applied machine learning for classification
- Detected both real and fake influencers
- Visualized the network
- Produced CSV outputs for influencer ranking

REFERENCES

1. Freeman, L. C. (1978). *Centrality in social networks: Conceptual clarification.* Social Networks, 1(3), 215–239.
2. Page, L., Brin, S., Motwani, R., & Winograd, T. (1998). *The PageRank citation ranking: Bringing order to the web.* Stanford InfoLab Technical Report.
3. Wasserman, S., & Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge University Press.
4. Cha, M., Haddadi, H., Benevenuto, F., & Gummadi, K. P. (2010). *Measuring user influence in Twitter: The million follower fallacy.* ICWSM.
5. Kwak, H., Lee, C., Park, H., & Moon, S. (2010). *What is Twitter, a social network or a news media?* WWW.
6. Cresci, S., Pietro, R. D., Petrocchi, M., Spognardi, A., & Tesconi, M. (2015). *Fame for sale: Efficient detection of fake Twitter followers.* Decision Support Systems, 80, 56–71.
7. Ribeiro, M. H., Benevenuto, F., & Figueiredo, F. (2018). *Detecting fake followers in Twitter.* Social Network Analysis and Mining, 8(1), 1–15.
8. Kipf, T. N., & Welling, M. (2017). *Semi-supervised classification with graph convolutional networks.* ICLR.
9. Hamilton, W., Ying, R., & Leskovec, J. (2017). *Inductive representation learning on large graphs (GraphSAGE).* NeurIPS.
10. Scikit-Learn Developers. (2024). *Scikit-Learn Documentation.* Retrieved from https://scikit-learn.org