Reimagining Influence Detection in Social Networks via Graph Neural Networks

Group: 08

Harshavardana Reddy Kolan
Shikha Kumari

Contents:

Introduction

- Problem Statement
- Next Steps Integrated

Dataset

- Data Collection
- Data Cleaning & Preprocessing

Models

- Classical Models (XGBoost)
- Graph Neural Networks (GNN) Pipeline

Graph Structure

- Node Types: For Stack Overflow - Users, Questions, Answers; AskReddit – Posts, Comments
- Edge Types: For Stack Overflow - Asks, Answers, Has, Accepted Answer; AskReddit - Asks, Answers, Has

Heterogeneous GNN (HetGNN)

- GraphSAGE Pipeline
- Model Architecture Explanation

Model Training & Metrics

- Training Setup: Stratified Mini-Batching
- Metrics: Accuracy, Precision, Recall, F1-score, AUC

Conclusion & Future Work

- Summary of Findings
- Recommended Improvements

Introduction

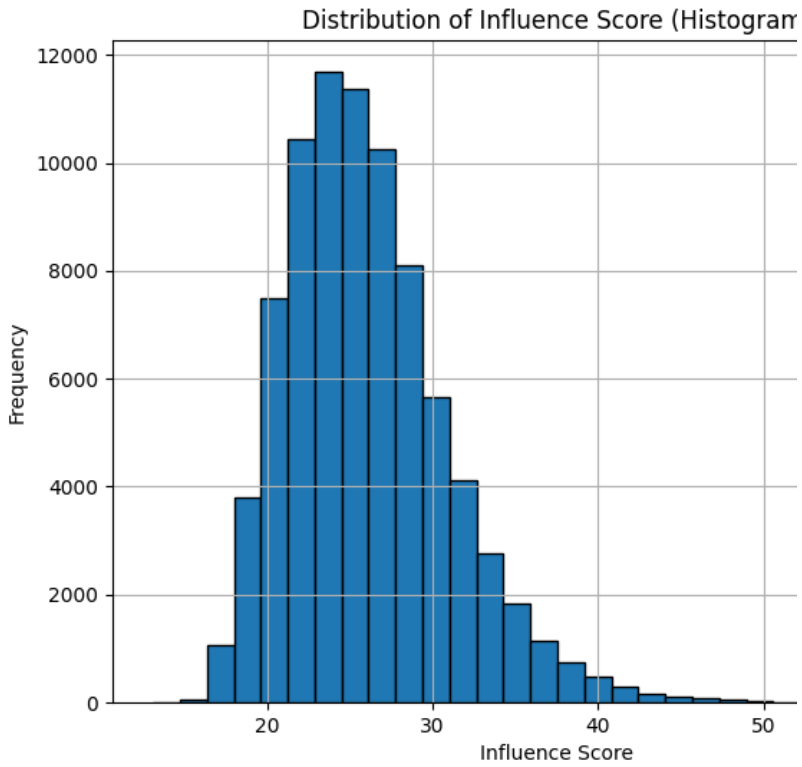
- **Problem Statement:**
  - Identifying influential users across social networks (Stack Overflow), Identifying active users acoss social network (Ask Reddit) for validation.

- **Objective:**
  - Reward influential users to enhance participation, engagement, and content quality.
  - Recommend top users to those seeking expertise in related domains.

Dataset Overview – Stack Overflow

- **What is the dataset?**
  - Stack Overflow interactions (Users, Questions, Answers)
  - AskReddit (Posts, Comments)
- **How did we extract it?**
  - Collected using API – Stack Overflow
  - Downloaded from Kaggle - AskReddit
- **How did we transformed in the tabular form for classical models?**
  - Formula to define influence:

    influence_score=**reputation+3*gold_badge_count+2*silver_badge_count+ bronze_badge_count**
  - The threshold for defining influential users is the users falling in the top 10% of influence_score.
  - Aggregated questions and answers by User ID to summarize user activity (total questions, average scores, accepted answers).
  - Merged the datasets(Users, Questions, Answers) to create a single, comprehensive dataset for classical models.



Dataset Overview – AskReddit

- **What is the dataset?**
  - AskReddit (Posts, Comments)
- **How did we extract it?**
  - Downloaded from Kaggle – A month of AskReddit
- **How did we transformed in the tabular form for classical models?**
  - Formula to define influence:
  - **Activity score= num_posts+num_comments**
  - The threshold for defining influential users is the users falling in the top 10% of activity_score.
  - Aggregated questions and answers by User ID to summarize user activity (avg_post_score, total_comments, avg_comment_score).
  - Merged the datasets(posts and comments) to create a single, comprehensive dataset for classical models.

```
influential
0    73620
1     8180
Name: count, dtype: int64
influential
0    0.9
1    0.1
Name: proportion, dtype: float64
```

```
active
0.0    51942
1.0    24780
Name: count, dtype: int64
active
0.0    0.677016
1.0    0.322984
Name: proportion, dtype: float64
```

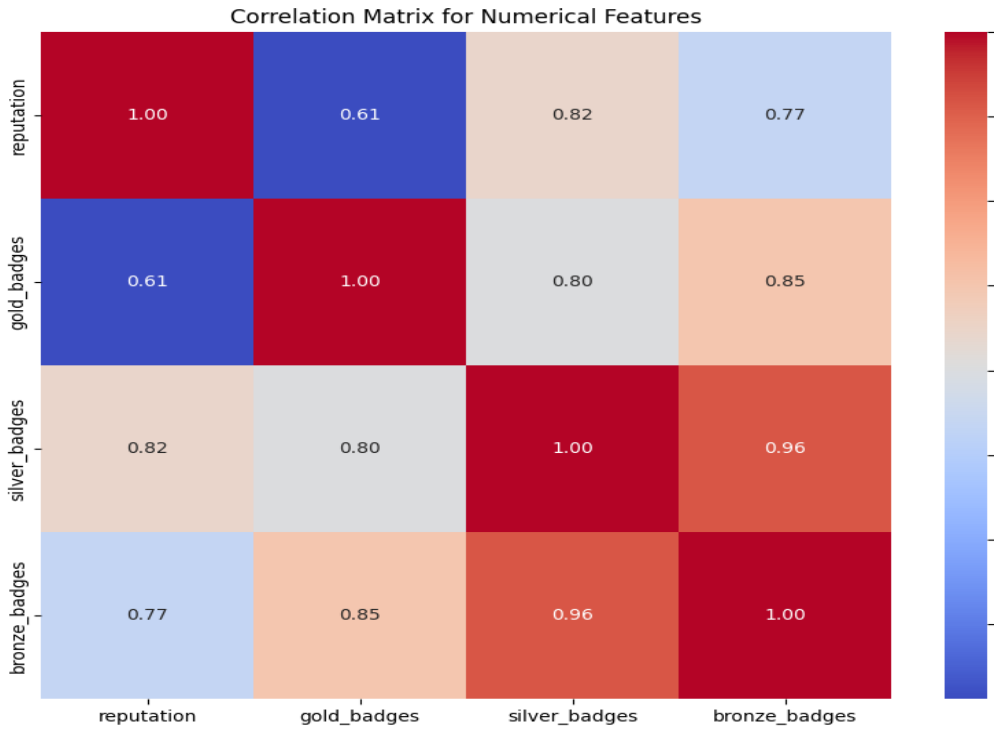
Preprocessing:

- **For Stackoverflow**

Removed reputation and badge variables because:

- High correlation between features
- Used for defining target variable

- **For Ask Reddit:**
- Removed unwanted columns such as timestamp, title and url of the post.



Models Used:

- **Classical Models:**
  - **XGBoost** for tabular learning.
- **Graph Neural Networks (GNN):**
  - Used **GarphSAGE**: **Heterogeneous Graph Representation** for node classification.

Classical Model: XGBoost

- **Applied on structured tabular data**
- **Features for Stack Overflow:**
  - total_questions, avg_question_score, avg_answer_score, accepted_answers
- **Features for AskReddit:**
  - avg_post_score, total_comments,avg_comment_score
- **Limitation:**
  - Ignores relationships between users and content (questions and answers)

Why GNNs?

- GNNs are **essential** for this problem because **Stack Overflow interactions are inherently a networked structure**.

Graphs Model Real-World Interactions:

- Unlike tabular models, **GNNs understand relational data**.
- This allows us to **predict user influence** based on their **position in the network**.

Message Passing & Information Propagation:

- GNNs **aggregate information** from connected nodes.
- **Classical models fail to model this** interaction effect.

Heterogeneous GNNs Adapt to Different Node Types:

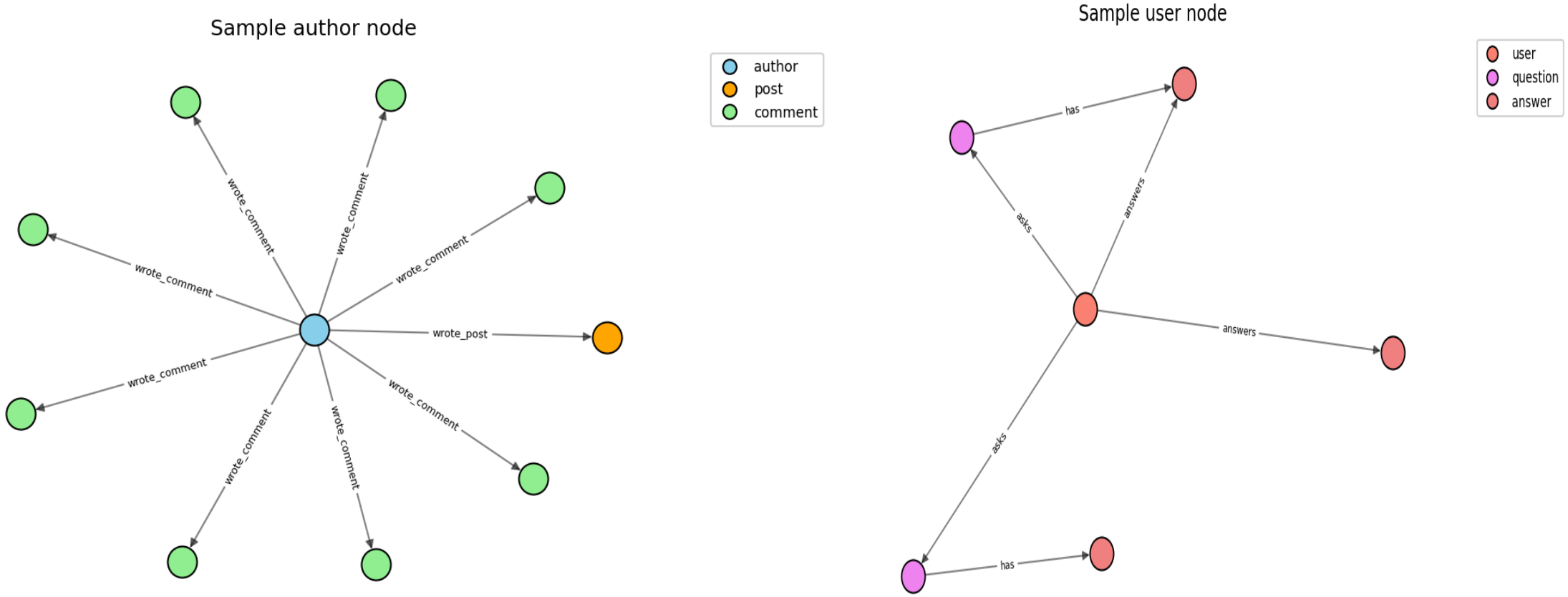
- Users, Questions, and Answers have **different roles**.
- GNNs **learn embeddings** specific to **each type of node** (e.g., an expert vs. a beginner has different engagement patterns).

GNN Structure

- **Node Type (StackOverflow)**
  - user, question, answer
- **Edge Type (StackOverflow)**
  - user → asks → question
  - user → answers → answer
  - question → has → answer
  - question → accepted_answer → answer
  - Reverse edges
  - Self-loops added

- **Node Type (**AskReddit**)**
  - author, post, comment
- **Edge Type (**AskReddit**)**
  - author→ wrote_post→ post
  - author → wrote_comment → comment
  - post → has→ comment
  - Reverse edges
  - Self-loopsadded

Visualization of Node and it's Neighbors



Training Pipeline

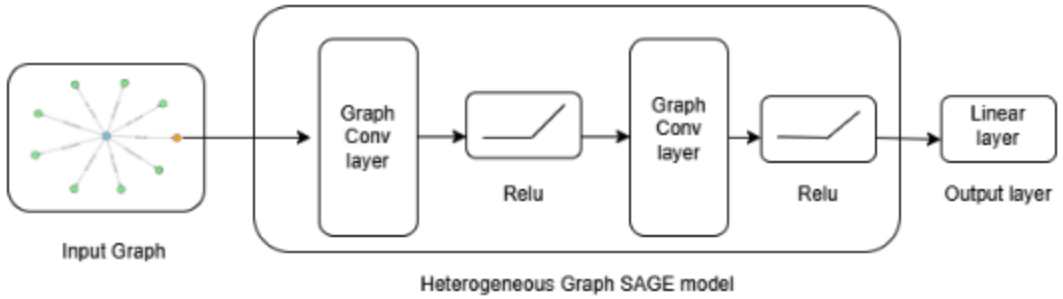


Fig. 6: GCN Model Architecture

- Takes an **input graph** with node features.
- Uses a **Heterogeneous GraphSAGE** model for message passing.
- Architecture consists of:
- **Graph Convolution Layer** → aggregates neighbor information.
- **ReLU Activation** → adds non-linearity.
- **Second Graph Convolution Layer** → further refines node embeddings.
- **ReLU Activation** → applied again for deeper representations.
- **Linear Layer (Output Layer)** → maps final embeddings to prediction scores.
- Suitable for tasks like **node classification** or **link prediction** in **heterogeneous graphs**.

Training Setup

- 🛠 **Training Setup for Heterogeneous GNN**
- **Graph Neural Network (GNN) Model:**
  - Implemented using **PyTorch Geometric**
  - Uses **Heterogeneous Graph Convolutions** for multiple node and edge types: GraphSAGE
- **Hyperparameters:**
  - **Hidden Layers:** 2-layer **SAGEConv** with 64 hidden units
  - **Output channels**: 2
  - **Aggregation:** Mean pooling to aggregate neighbor information
  - **Learning Rate: 0.01**
  - **Num_epochs: 50**
  - **Batch Size:** 64
  - **Number of neighbours**: [10,10]
  - **Split:** 'train_rest'
  - **Optimizer:** Adam(Weight Decay: **1e-4** for regularization)
  - **Loss Function**: Cross Entropy
- **Data Handling & Training:**
  - **Self-loops:** Helps model **user self-engagement patterns**
  - **Node Splitting: RandomNodeSplit** (80% train, 10% validation, 10% test)
  - **Stratified Mini-Batching** for balanced sampling (necessary due to class imbalance) - New Addition
  - **Mini-Batch Training: NeighborLoader** for efficiency

Stratified Sampling

Algorithm 1 Stratified Mini-Batching

Input: Target Variable y, train mask, batch size B
Output: Initialize a list of class-balanced mini-batches
$node\_indices \leftarrow$ Extract indices where train mask is true
Initialize $class\_indices \leftarrow$ empty dictionary
for each class c in unique($y[node\_indices]$) **do**
    $idx \leftarrow$ indices in $node\_indices$ where label $= c$
    Shuffle idx
    $class\_indices[c] \leftarrow idx$
end for
$num\_classes \leftarrow$ number of unique classes
$samples\_per\_class \leftarrow B \div num\_classes$
$min\_batches \leftarrow$ minimum complete batches possible across classes
Initialize $batches \leftarrow$ empty list
for $i = 0$ to $min\_batches - 1$ **do**
    $batch \leftarrow$ concatenate:
        $class\_indices[c][i \cdot samples\_per\_class : (i+1) \cdot samples\_per\_class]$ for all c
    Append $batch$ to $batches$
end for
return $batches$

- **Extract training indices**:
- **Group indices by class**:
- **Shuffle each class's indices**:
- **Compute batching parameters**:
  - samples_per_class: How many samples of **each class** should go into one batch.
  - min_batches: The number of **complete** batches we can form, based on the smallest class size.
- **Form batches**:
  - Take the right slice of samples from **each class's shuffled list**.
  - Concatenate them to form one batch.
- **Return all batches**.

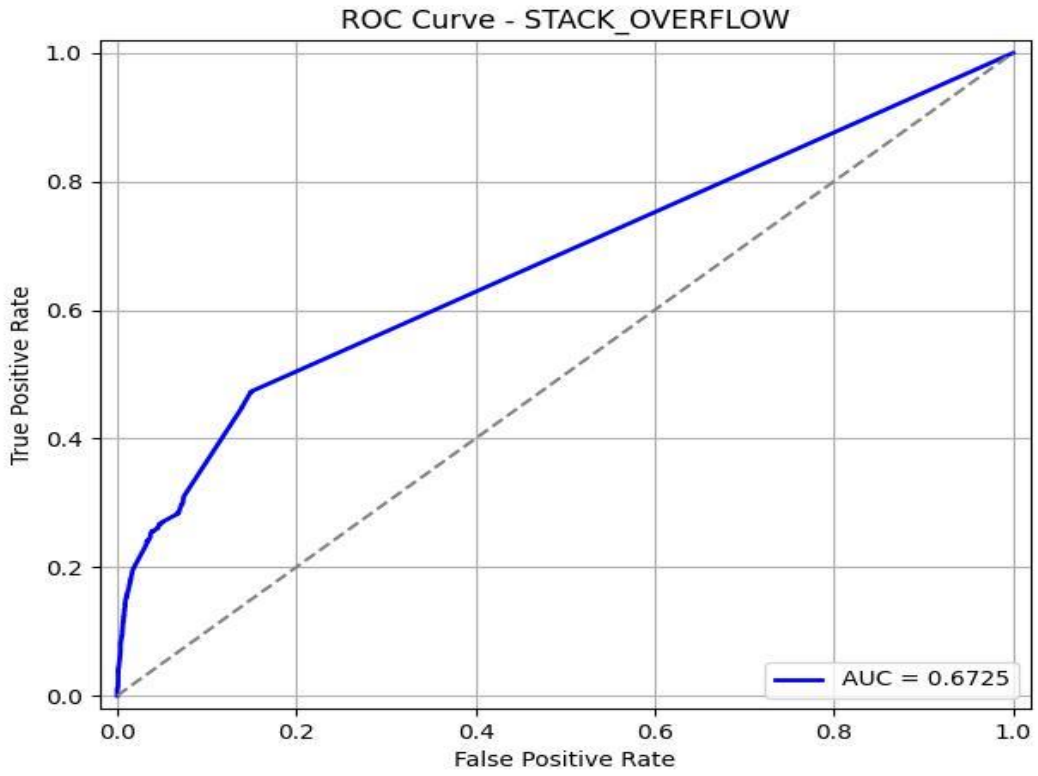
Evaluation Metrics

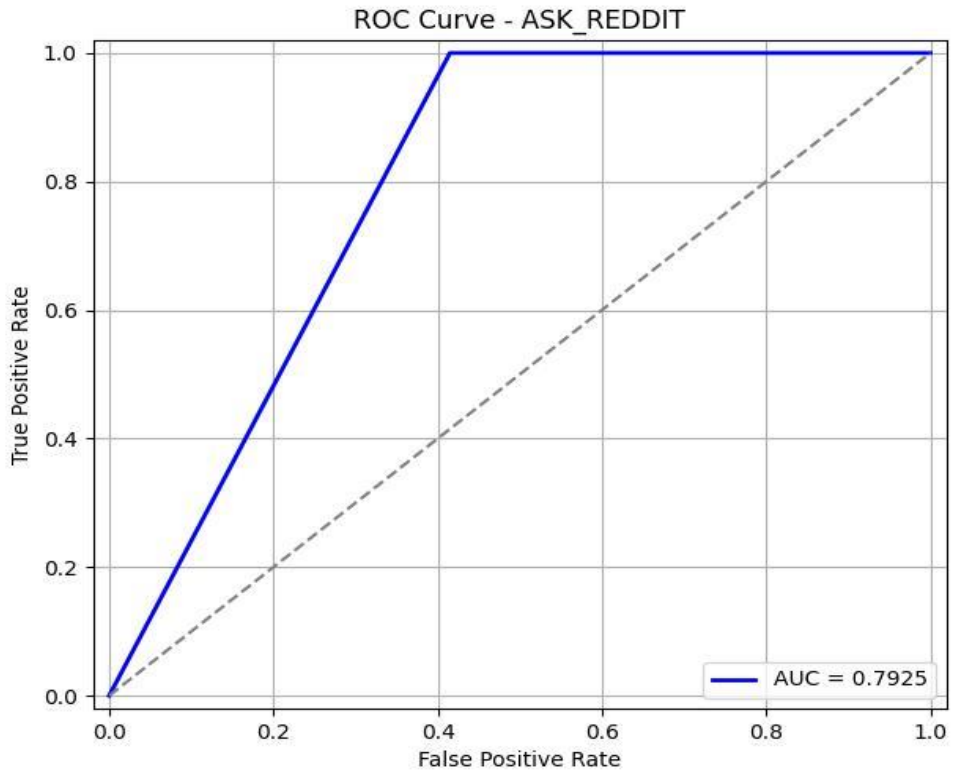
Table 1. Performance Comparision

	Stack Overflow					AskReddit				
Models	Accuracy	Precision	Recall	F1-score	AUC	Accuracy	Precision	Recall	F1-score	AUC
Xgboost	0.813	0.589	0.646	0.603	0.655	0.701	0.657	0.659	0.658	0.706
GCN	0.813	0.596	0.661	0.612	0.672	0.655	0.664	0.792	0.616	0.792

- Both models perform well across datasets, with GCN showing slightly better overall metrics.
- Despite a good AUC score for AskReddit, the ROC curve suggests underfitting, likely due to the model being too simple to capture complex patterns.

ROC Curve for GNN (Stack Overflow | AskReddit)





Future Work:

- Increasing complexity of the models by number of layers, changing configurations.
- Trying different models – like GAT
- Extending to other platforms in Social Networks.