# Mining The Specific Skill Demographics In Social Network

COMP 4710 (Winter 2023)

Group #13

S M Rukunujjaman
Aaron Joson
Gurkiran Grewal
Jiazhen Tian

University of Manitoba

# Introduction

In recent days, the need for fast and efficient data mining in social networks has become increasingly important to ensure users can quickly access their desired information. The field of social network research is continuously evolving, with new methods and approaches being developed to enhance the efficiency of social network mining. In this context, we will focus on mining data from a specific social network, GitHub, using an advanced method known as the Graph Convolutional Network (GCN) algorithm. Our aim is to extract information in a more productive manner, thereby improving the overall mining process.

# Project problem description

- For our project we will mine on github information, this is basically a graph-based problem involving nodes and edges .
- Our task is to look into github users repositories and predict whether the user fits in a specific category or not. In this case we will determine if a user is a web or machine learning developer
- Our primary dataset consists of social network of GitHub developers which was collected from the public API in June 2019.
- This problem related to binary node classification.
- Nodes are developers who have starred at least 10 repositories and edges are follower relationships between them. features are based on location, starred repositories, employer, and email.

# Motivation

- Most of the Algorithms we found are involved clustering, eccentricity and centrality measure of the nodes, skim-gram node embedding etc.
- These measures are not inherently designed for node classification tasks
- It increases complexity in processing features as more computation and normalization of data is required
- Redundancy occurs when two or more features provide almost the same information, leading to an unnecessary repetition of information in feature set.
- For larger data or graph scalability issue happens while calculating centrality and eccentricity measures as it has high computational complexity.
- These measures provide global information about graph structure but do not fully captures local information about immediate neighbourhood nodes.

# Potential real-life applications

This binary node classification used in several real-life applications such as

- Talent scouting and recruitment
- Personalized recommendation
- Targeted marketing and advertising
- Community building
- And many more

# Graph Convolutional Network(GCN)

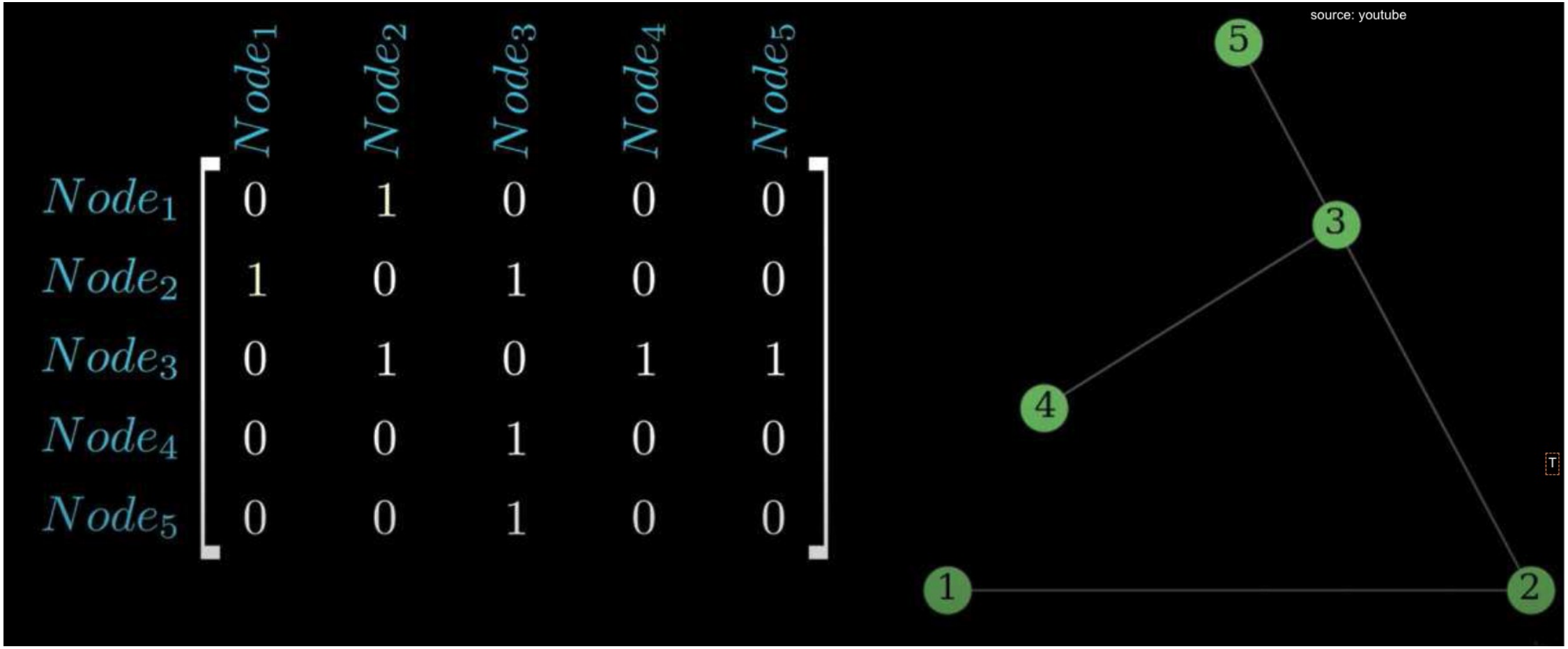
GCN works as follows

- Preprocessing  the data
  - Train  and validate
  - Message passing through by multiplying hidden state with adjacency matrix and normalize adjacency matrix
- Then calculate the loss function
- Test the data

# Preprocessing data (GCN)

- We create adjacency matrix out of the edge and feature list
- Normalize the feature matrix to ensure that features are on the same scale.
- by multiplying the hidden state (or node features in the first layer) by it, we are sort of applying a mask and aggregating only the information from neighbouring nodes.
- Train the normalized matrix. Optimize the model parameters to minimize the loss function on the training data.

# Adjacency matrix



# Loss calculation

- It is difference between the predicted output and the actual output
- The loss function is used in the backpropagation algorithm to compute the gradients of the loss with respect to the model parameters.
- minimizing the loss, your model will be able to make more accurate predictions on whether a GitHub user is a web or machine learning developer based on their features and follower relationships.
- Then we have to test the data. Test value closer to 1 gives more accurate prediction .

# How GCN is different?

- It is designed exclusively for node classification task .
- GCN learns meaningful features from the graph structure and node attributes, which allows it to capture both local and global information in the graph.
- GCN aggregates information from the local neighborhood of each node, which allows it to effectively capture the dependencies between nodes in the graph.
- GCN can generalize to unseen nodes and graphs, which is important for handling dynamic graphs or graphs with constantly changing node information.

# Our Group Contributions

- We would be Using Graph Convolutional Network (GCN) to mine for Interesting associations within the input graph.
  - Categorizes the nodes and determine their communities based on their interests.
  - We can create a  profile description for each communities based on the result of GCN.
- GCN offers high flexibility and versatility.
  - Each user node keeps a vector matrix containing all its features/labels.
  - Feature vectors are used to form communities with the same or similar features.
  - Feature vector allows GCN layers to mine for different rules and associations.

# Methodology

- The characteristics of each nodes are stored in a vector that will be sent to its neighboring nodes to mine for community associations and thus organize them into communities/clusters.
- The mined associations can be used to determine information about unknown nodes within the same community.
- Uses Multiple Layers to preprocess the input data to enhance the accuracy of our mined results.
- Predictions are generated by forming common interests within the cluster/community.

# Preliminary Results

We intend to identify Github developers into web or ML engineers.
So We need to pre-process the dataset.
After we processed the data, we first tested a portion of the data
First we need the GCN Model

we build a model

with 2 GCN dropout layers.

Each layer will have 32 nodes

Model: "model"

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | [(1, 37700, 4005)] | 0 | |
| dropout_2 (Dropout) | (1, 37700, 4005) | 0 | input_1[0][0] |
| input_3 (InputLayer) | [(1, 37700, 37700)] | 0 | |
| graph_convolution_2 (GraphConvo | (1, 37700, 32) | 128192 | dropout_2[0][0]<br>input_3[0][0] |
| dropout_3 (Dropout) | (1, 37700, 32) | 0 | graph_convolution_2[0][0] |
| graph_convolution_3 (GraphConvo | (1, 37700, 32) | 1056 | dropout_3[0][0]<br>input_3[0][0] |
| input_2 (InputLayer) | [(1, None)] | 0 | |
| gather_indices (GatherIndices) | (1, None, 32) | 0 | graph_convolution_3[0][0]<br>input_2[0][0] |
| dense (Dense) | (1, None, 1) | 33 | gather_indices[0][0] |

Total params: 129,281
Trainable params: 129,281
Non-trainable params: 0

# Preliminary Results

The training process is repeated for 200 epochs, using 32 samples for each training session and validation

Once we have trained the model, we will use the model to test the dataset

```
Epoch 1/200
1/1 [==============================] - 198s 198s/step - loss: 0.6941 - acc: 0.4400 - val_loss: 0.6409 - val_acc: 0.7750
Epoch 2/200
1/1 [==============================] - 95s 95s/step - loss: 0.6443 - acc: 0.7200 - val_loss: 0.5901 - val_acc: 0.7750
Epoch 3/200
1/1 [==============================] - 98s 98s/step - loss: 0.6011 - acc: 0.7200 - val_loss: 0.5447 - val_acc: 0.7750
Epoch 4/200
1/1 [==============================] - 98s 98s/step - loss: 0.5575 - acc: 0.7200 - val_loss: 0.5129 - val_acc: 0.7750
Epoch 5/200
1/1 [==============================] - 109s 109s/step - loss: 0.5320 - acc: 0.7200 - val_loss: 0.5031 - val_acc: 0.7750
Epoch 6/200
1/1 [==============================] - 105s 105s/step - loss: 0.5229 - acc: 0.7200 - val_loss: 0.5026 - val_acc: 0.7750
Epoch 7/200
1/1 [==============================] - 100s 100s/step - loss: 0.5115 - acc: 0.7200 - val_loss: 0.4909 - val_acc: 0.7750
Epoch 8/200
1/1 [==============================] - 107s 107s/step - loss: 0.4825 - acc: 0.7200 - val_loss: 0.4701 - val_acc: 0.7750
Epoch 9/200
1/1 [==============================] - 108s 108s/step - loss: 0.4673 - acc: 0.7200 - val_loss: 0.4464 - val_acc: 0.7750
Epoch 10/200
1/1 [==============================] - 100s 100s/step - loss: 0.4274 - acc: 0.7250 - val_loss: 0.4277 - val_acc: 0.7850
Epoch 11/200
1/1 [==============================] - 121s 121s/step - loss: 0.4138 - acc: 0.7650 - val_loss: 0.4151 - val_acc: 0.8150
Epoch 12/200
1/1 [==============================] - 107s 107s/step - loss: 0.3863 - acc: 0.8250 - val_loss: 0.4040 - val_acc: 0.8300
```

# Challenges & Limitations

- **Data preprocessing**: High dimensionality and sparsity of the social networking data. Contains missing data and not every node is connected to another node. This makes it difficult to effectively capture the graph structure and to make accurate predictions.
- **Graph structure:** Social network structure is complex and has large number and different types of nodes and edges. Larger the size of graph, higher the computational cost will be.
- **Dynamic nature of data**: Social networks are dynamic and can change over time, requiring additional preprocessing.

# Our Plan

- **Data Pre-processing**: Use feature technique –
  - Select features that are highly correlated with the target variable. Discard weakly correlated features. In our case, we are aggregating information from the neighbouring nodes and then normalizing it. Using the loss function brings it closer to accurate prediction.
  - Remove the least important feature iteratively from the subsets(layers) of data.

  This way we can find features that are most relevant to the analysis task.
- **Graph Structure**: Techniques aiming -
  - Graph coarsening – Aggregate nodes of similar attributes. It will reduce the size of the graph and the computational cost of the analysis.
  - Assign weights to different nodes and edges based on their importance to the analysis task. This allows, GCN model to focus on most relevant parts of the graph.

# Our Plan

- **Dynamic nature**: Use temporal GCN models to keep track of evolution of the social network over time. With changes in data, it allows the GCN model to adapt to changes in the graph structure. It helps to identify trends and patterns and improves predictive accuracy.

# Thank you!

Questions..?