Mining The Specific Skill Demographics In Social Network

S M Rukunujjaman
Dept. of Computer Science
University of Manitoba
Winnipeg, Canada
rukunusm@myumanitoba.ca

Aaron Joson
Dept. Of Computer Science
University of Manitoba
Winnipeg, Canada
josona@myumanitoba.ca

Gurkiran Grewal
Dept. of Computer Science
University of Manitoba
Winnipeg, Canada
grewalg2@myumanitoba.ca

Jiazhen Tian
Dept. Of Computer Science
University of Manitoba
Winnipeg, Manitoba
tianj3@myumanitoba.ca

***Abstract*— GitHub is a popular social coding platform with millions of users and repositories. The social interactions between these users and repositories form a complex network that can provide valuable insights into the programming community. In this paper, we propose a graph convolutional network (GCN) based approach to analyze the GitHub social network and extract meaningful information from it. We use GCN to learn the graph structure and represent the network as a set of high-dimensional node embeddings. We used GCN to distinguish between ML developers and web developers, and by running it we can get a very clear graph to distinguish them, and of course since the two groups have a lot in common, we can see a lot of partial overlap. Our experiments show that our GCN-based approach outperforms traditional methods in predicting user attributes on the GitHub social network. Furthermore, we demonstrate the interpretability of our model by visualizing the learned embeddings and showing their clustering patterns. Overall, our work highlights the potential of GCN-based methods for data mining on large-scale social networks like GitHub.**

Keywords— GitHub, GCN , node classification, loss function, graph, adjacency matrix, backpropagation , label propagation , model prediction.

I. Introduction

GitHub, a widely-used social coding platform, hosts millions of users and repositories, creating a complex network of social interactions that offer valuable perspectives on the programming community. Among these insights is the capacity to differentiate between various types of developers, such as machine learning (ML) and web developers. Nevertheless, extracting these insights can be a daunting endeavor due to the extensive scale and intricacy of the data.

To address this challenge, this is basically a node classification problem so we propose a graph convolutional network (GCN) based approach to analyze the GitHub social network and extract meaningful information from it. GCN is a type of graph neural network that can learn the graph structure and generate high-dimensional node embeddings, which capture important structural and semantic features of the network.

In our study, we use GCN to distinguish between ML developers and web developers on GitHub. By running GCN, we are able to generate a clear graph that distinguishes between the two groups, while also showing some partial overlap between them. Our experiments show that our GCN-based approach outperforms traditional methods in predicting user attributes on the GitHub social network.

Furthermore, we demonstrate the interpretability of our model by visualizing the learned embeddings and showing their clustering patterns. These visualizations provide insights into the community structure and characteristics of the different types of developers on GitHub.

Overall, our work highlights the potential of GCN-based methods for data mining on large-scale social networks like GitHub and provides a foundation for further research in this area.

In addition , these node classification have several real life applications . this applications are so wide spread that it can be found everywhere in our daily life. Following is some real life application of this

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

II. Background And Related work

A. Background

Social Media Networks (SMN) such as Twitter, Tiktok, GitHub, etc. have been massively popular with the public and became the mainstream source of information and sometimes misinformation. In recent years, they have become treasure troves containing a massive amount of information including some that have been studied and used in business ventures such as advertising and identifying target demographics, yet they still contain information that has not been subjected to processing such that there is still a lot of possible discoveries which can lead to furthering our knowledge and understanding about the interactions between people and technology.

For the purpose of our project, we have decided to mine for the specific skill demographics within the GitHub social network using a technique called *Graph Convolutional Network* (GCN) that uses machine learning to extrapolate data from a graph input consisting of nodes representing the users and edges representing the github repositories the two users share in common (eg. repositories that they both follow or liked). Graphs represent the dynamic structure of the GitHub social network where the nodes form intricate webs of connection with each other through multiple edges and thus forming different clusters containing similar properties. Considering that each node is able to form one or more connections in relation to other nodes, we are able to extract multiple different feature projections using the dataset by adjusting the search parameters we are using.

Graph Convolutional Networks operate using feature propagation in which it traverses through the edges between the nodes in order to form clusters or communities of nodes possessing the same features (eg. the two users both followed the same repositories), It will then use this knowledge to create a community profile that can be used to predict some information about nodes within the clusters. GCNs uses back propagation in order to improve the accuracy and reflect the dynamic changes within the dataset or input graph and update the information about the cluster of nodes thus updating the characteristics of the nodes belonging to the same cluster.

B. Related Works

We are aware of other variations of *Graph Convolutional Networ*k techniques that have been developed in recent years in order to address some of the issues that can arise when applying this technique such as the scalability, increased complexity or the intensive hardware resource required in order to run GCN when using a highly detailed or high-volume input graphs. These variations also suggest potential optimizations to consider such as changes to the formula to remove redundancy or reduce the complexity. For this section, we will look into some of the different variations for GCN and the changes they have incorporated to increase the accuracy and efficiency as well as optimizations they have done to reduce the complexity or resources it would require to run.

- *FastGCN* [9] proposes that by defining the manner of which the samples are collected, it will allow for uniformity within the samples and thus allow the precomputation of variance in batch loss and gradient. In this manner, FastGCN reduces the complexity of the computation and thus increases efficiency without sacrificing its accuracy. Unfortunately, the potential overhead calculations can overwhelm the benefits provided by FastGCN due to the complexity of the sample features and the number of layers that are used in calculating the variance.
- *LightGCN* [8] proposes that depending on the purpose we are trying to achieve, some parts or components of the original GCN equation can have minimal or negative implications on the results. They show that some of the components such as nonlinear activation and feature transformation can be replaced by simple functions. For our purpose, we decided to stick with the original GCN implementation and work on how we could extrapolate other information using the same set of input, leaving the possible optimization for the future.
- *Cluster-GCN* [7] suggests that the partitioning method used to define the clusters within the graph can have significant effects to the time complexities and resources required by the algorithm and thus have a profound effect on its ability to produce accurate predictions as well as the ability to train additional GCN layers without compromising accuracy and efficiency. This suggests that the scalability of a GCN algorithm is affected by how well the clusters within the dataset are defined and the method of which the calculations for entropy in each GCN layer.
- In most of the literature regarding GCNs such as [10], the authors have noted the issue of over-smoothing of the results with regards to additional GCN layers as a result of repetitive node embedding within each layer. This causes additional layer(s) of complexity within the calculations and has a negative effect on the computational speed and accuracy of a GCN algorithm.
- suggests the idea that a hierarchy might exist between the relations that takes the interest of each user nodes and adjusts the weights used traverse the edges

connecting them, this would imply that the distance between each neighborhood clusters might be closer and interconnected in more than one way. However, trying to explore this notion would include additional costs in terms of computational complexity and might not be accurate enough to reflect the actual relationship between clusters[13].

Each of these literature highlights different areas of which optimizations and fine tuning can be done in order to reduce the complexities of using GCN, We believe that this also opens up research opportunities to create a GCN variation that is able to provide multi-faceted results such that it is able to maintain or reduce the overhead calculations and time complexity while increasing the accuracy without sacrificing any functionality. It is also possible to extract the key ideas in each literature and aggregate them into a universally applicable version without losing any functionality or increasing any complexity and demands for resources. This will be for future research purposes as it requires further studies.

III. Main Body

A. Our Ideas

In this section we will describe our ideas behind using graph convolutional network preferably called GCN. The main goal of our project is to mining data from a social media network. We chose GitHub as a social media network, and we want to mine data from there. In this case, we are mining specific skill demographics from GitHub. More specifically, we are mining data to know whether a user from GitHub is a web developer or a machine learning developer. It is not limited to mine in just these two categories, we can also be able to mine any specific skills we would like to mine on.

We used data set from Kaggle which is collected from public API in June 2019. Data set contains nodes, edges, and features. Here nodes are developers who maintain at least 10 repositories in GitHub, there are around 40000 nodes. Edges are follower relationship between them. In addition, the vertex feature is extracted based on location, repositories, employer, and email address. This task related to graph binary node classification. We will look in GitHub user repositories to predict whether it fits into a specific category. in this case we are using 10 repositories for our purpose, but this number is not limited to it.

We will use graph convolutional network (GCN) model to mine the data. For that purpose, we must get frequent dataset through some steps. First, we will train and validate the dataset then, we are going to use adjacency matrix through which we will calculate the loss functions. Adjacency matrix is created from the datasets that we have. Loss function is crucial for predicting the model. So, we will use layer wise propagation rules which operate directly on graphs to show how it can be motivated from the first approximation of graph convolutions [1]. We also try to focus on doing fast and scalable node classifications in graph. Our model has been tested on our datasets, and the results indicate that it performs well in terms of both classification accuracy and efficiency when compared to others. Let us look at how we used graph convolutional network model in our project.

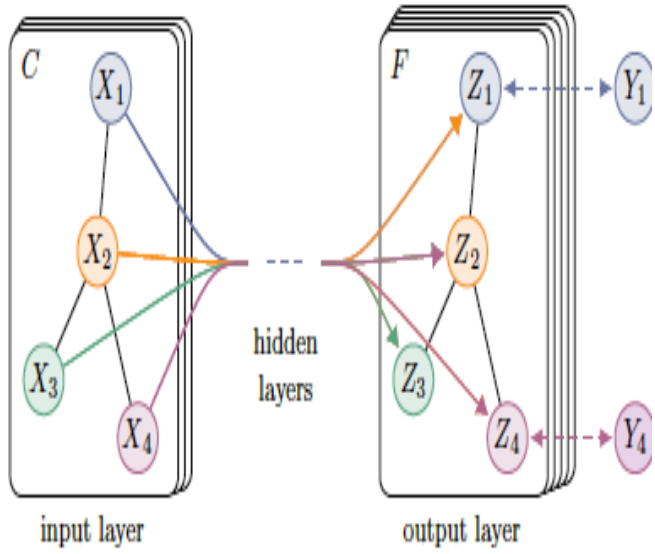
B. Details About Our Graph Convolutional Network Model

1) Train and validate data : we are going to train and validate data first. Because GCN needs less level than other supervised model. This is make our next process easy and reduce overheads for futher calculation. we will use 2% labeled data which is approximately 800 developers or nodes. so we are going to 400 developers for training and 400 developers for validation . Larger number gives more accurate prediction.

2) Data preprocessing : we need to preprocess the data . first we create adjacency matrix out of the node, edges and features. we are going to use multi-layer model for our GCN graph. The idea is , the information of neighboring nodes passed by multiplying multiplying hidden state by adjacency matrix through label propagation the adjacency matrix denotes connection between nodes. Consequently, when we multiply the hidden state (or the node features in the initial layer) by this, we are effectively applying a filter and accumulating information solely from the adjacent nodes. Our GCN layers can be explained by the following equation with undirected graph[2]

$$H^{(l+1)} = \sigma(\widetilde{D}^{-\frac{1}{2}}\tilde{A}\widetilde{D}^{-\frac{1}{2}}H^{l}W^{l})$$

H – hidden state or node attributes, when l=0
$\widetilde{D}$ – degree matrix or normalization of matrix
$\tilde{A}$ – Adjacency matrix with self-loop
W – trainable weights
σ – activation functions
l – layer number



There are two components to this equation: the non-trainable part (involving $\widetilde{D}$ and $\tilde{A}$) and the trainable part (involving H and $\tilde{A}$). The non-trainable section is referred to as the normalized adjacency matrix. It's worth noting that if we exclude the non-trainable component, the remaining part resembles a standard dense layer. By multiplying the hidden state with the normalized adjacency matrix, we effectively aggregate features from neighboring nodes.

Label propagation: our label propagation works similar to the following ways [3]

- Order nodes in a way that nodes coupled with known nodes
- Calculation of adjacency matrix
- Calculation of transition matrix
- Mark known nodes with 1 and unknown nodes 0
- Multiply known labels with transition matrix
- Repeat until label stops changing its behavior.

Pseudocode for the label propagation algorithm is following

Input: Graph G, level L, training set V_t, weight coefficients w_ℓ, convolutional coefficient r, threshold r_{max}, number of walks per node n_r, feature matrix $\mathbf{X}_{n\times F}$
Output: Embedding matrix $\mathbf{P}_{n\times F}$

$\mathbf{S}^{(\ell)} \leftarrow Sparse\left(\mathbf{0}_{n\times n}\right)$ for $\ell = 0,\ldots,L$;
for *each node* $s \in V_t$ **do**
  Generate n_r random walks from s, each of length L;
  if *The j-th random walk visits node u at the ℓ-th step,* $\ell = 0,\ldots,L, j = 1,\ldots,n_r$ **then**
    $\mathbf{S}^{(\ell)}(s,u) \mathrel{+}= \frac{1}{n_r}$;
$\mathbf{Q}^{(\ell)}, \mathbf{R}^{(\ell)} \leftarrow Sparse\left(\mathbf{0}_{n\times F}\right)$ for $\ell = 1,\ldots,L$;
$\mathbf{Q}^{(0)} \leftarrow \mathbf{0}_{n\times F}$ and $\mathbf{R}^{(0)} \leftarrow ColumnNormalized\left(\mathbf{D}^{-r}\mathbf{X}\right)$;
for ℓ *from* 0 *to* $L-1$ **do**
  for *each* $u \in V$ *and* $k \in \{0,\ldots,F-1\}$ *with* $\left|\mathbf{R}^{(\ell)}(u,k)\right| > r_{max}$ **do**
    for *each* $v \in \mathcal{N}(u)$ **do**
      $\mathbf{R}^{(\ell+1)}(v,k) \mathrel{+}= \frac{\mathbf{R}^{(\ell)}(u,k)}{d(v)}$;
    $\mathbf{Q}^{(\ell)}(u,k) \leftarrow \mathbf{R}^{(\ell)}(u,k)$ and $\mathbf{R}^{(\ell)}(u,k) \leftarrow 0$;
$\mathbf{Q}^{(L)} \leftarrow \mathbf{R}^{(L)}$ and $\mathbf{R}^{L} \leftarrow Sparse\left(\mathbf{0}_{n\times F}\right)$;
$\hat{\mathbf{P}} \leftarrow \sum_{\ell=0}^{L} w_\ell \cdot \mathbf{D}^r \cdot \left(\mathbf{Q}^{(\ell)} + \sum_{t=0}^{\ell} \mathbf{S}^{(\ell-t)}\mathbf{R}^{(t)}\right)$;
return Embedding matrix $\hat{\mathbf{P}}_{n\times F}$;

Algorithm 1. Label propagation for adjacency matrix and normalization of that adjacency matrix.

This algorithm reduced the time complexity and works very well with the adjacency matrix since it gives much more accurate approximation while training nodes and feature matrix.

3) Loss Function: In a Graph Convolutional Network (GCN), the calculation of the loss function is a crucial element that directs the training of the network. It measures the disparity between the model's predicted outputs and the true ground-truth values. The main objective of computing the loss function is to fine-tune the model's parameters throughout the learning process. To illutrate more we can explain it more matematically. Let Y represent the target space (for example, Y = {+1, -1} in binary classification), and let P|Y| represent the |Y|-dimensional probability simplex. In a standard classification problem, a loss function L : P × Y → R^+ maps a target-prediction pair to a non-negative real number. The objective is to minimize the loss function over a given training dataset D[4].

Within the scope of GCN, the loss function fulfills several roles such as

- Parameter optimization – The objective of training a GCN is to reduce the value of the loss function through iterative adjustments of the model's parameters, such as weights and biases. The gradient of the loss function concerning these parameters is employed to update them using an optimization algorithm.
- Measurement of performance – The loss function supplies a numerical measure to assess the model's performance during each training step. A smaller loss value signifies that the model is generating more accurate predictions, while a larger loss value implies that the model's performance is suboptimal.
- Regularization – The loss function may integrate regularization terms to avert overfitting. By adding constraints on the model parameters, these terms promote learning of more straightforward and easily generalizable data representations.
- Model selection - The loss function can facilitate comparison of various model architectures or hyperparameters during the model selection phase. A model yielding a lower loss value on a validation set is deemed more suitable for addressing the given problem.

Now, we know some of the characteristics of loss function. Let's move forward to find how that loss function can be determined for our purpose. We used a back propagation algorithm for calculating the loss function. This back propagation algorithm calculates the loss layer by layer, that is the layer-number we are using. It will calculate the function for each layer and compare it to the previous layer. Thus, find optimal value for our model. To give an illustrative overview here is a diagram of backpropagation [5].

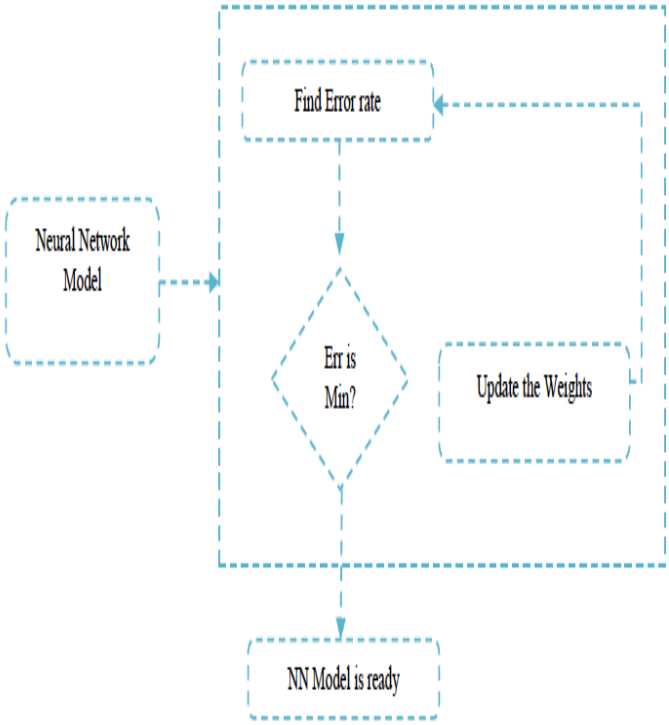


Fig. 1. Workflow of backpropagation algorithm

The diagram is explained in the following ways

- It calculates the model output with actual output.
- Verify whether the error is minimized or not.
- If the error exceeds the acceptable range, the weights and biases are updated. Subsequently, the error is re-evaluated. This process is iterated until the error is reduced to an acceptable level.
- When the error is within acceptable or more reliable range, the model is ready to get the forecasting data.

Generalized workflow and step by step computation of this backpropagation algorithm can be shown using the following pseudocode.[5]

Algorithm 2. Pseudocode for back propagation

```
1. Assign all network inputs and output
2. Initialize all weights with small random numbers, typically between -1 and
3. repeat
       for every pattern in the training set
       Present the pattern to the network
4. //     propagated the input forward through the network:
         for each layer in the network
           for every node in the layer
             1. Calculate the weight sum of the inputs to the node
             2. Add the threshold to the sum
             3. Calculate the activation for the node
           end
         end
5. //     Propagate the errors backward through the network
          for every node in the output layer
            calculate the error signal
          end
6. for all hidden layers
             for every node in the layer
               1. Calculate the node's signal error
               2. Update each node's weight in the network
             end
          end
7. //     Calculate Global Error
          Calculate the Error Function
8. end
9. while ((maximum number of iterations < than specified) AND
       (Error Function is > than specified))
```

We calculate our loss with iteration in every epoch. An epoch refers to a complete cycle of the training process where the entire dataset is fed into the neural network model, comprising of one iteration of forward propagation followed by backpropagation. We can set our epoch to a certain number, we did it for 200. Any number can be chosen, but larger number gives more accurate prediction on each epoch iteration.

```
Epoch 1/200
1/1 [==============================] - 198s 198s/step - loss: 0.6941
Epoch 2/200
1/1 [==============================] - 95s 95s/step - loss: 0.6443 -
Epoch 3/200
1/1 [==============================] - 98s 98s/step - loss: 0.6011 -
Epoch 4/200
1/1 [==============================] - 98s 98s/step - loss: 0.5575 -
Epoch 5/200
```

Fig. 2. Calculation of loss in every epoch

The above result shows that in each iteration of the epoch the loss function is getting reduced such our model is able to predict with more accuracy.

4) Model prediction : since we are done with all necessary steps such as data preprocessing, calculating loss function we are now able to predict the outset with our model. To do this we determined the following characteristics

- Receiver Operating Characteristic Area Under Curve (ROC AUC) – This measures the performance of binary classifier. Higher value indicates better classification. It ranges from 0 and 1.
- Precision-Recall Area Under Curve (PR AUC)- focuses on tradeoff between precision and recall. Higher value indicates better performance in precision.
- F1 score - This is the harmonic mean of precision and recall, with values ranging from 0 to 1. A higher F1 score indicates better performance.

Now with those values we can get our desired frequent mining sets from the dataset . in addition, model is now able to distinguish between who is web developer and who is machine learning developer.

A neural network model designed for graph-structured data should ideally be capable of learning node representations within a graph, considering both the graph's structure and the feature descriptions of the nodes[2]. The 1-dimensional Weisfeiler-Lehman (WL-1) algorithm[6] offers a well-researched framework for uniquely assigning node labels in a graph, given the graph itself and, optionally, initial discrete node labels.

Algorithm 3. Psedocode for WL-1 that gives coloring reprentation of our graph

Input: Initial node coloring $(h_1^{(0)}, h_2^{(0)}, ..., h_N^{(0)})$
Output: Final node coloring $(h_1^{(T)}, h_2^{(T)}, ..., h_N^{(T)})$
$t \leftarrow 0$;
repeat
  for $v_i \in \mathcal{V}$ **do**
    $h_i^{(t+1)} \leftarrow \text{hash}\left(\sum_{j \in \mathcal{N}_i} h_j^{(t)}\right)$;
  $t \leftarrow t+1$;
until *stable node coloring is reached*;

h_i^t represent the coloring of node v_i and N_i is its set of neighboring node indices here it uses has function. But for our purpose we will use layers like differentiable functions with that trained parameters. Therefore, we replaced, the hash () function of with the following equation [2]

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}} h_j^{(l)} W^{(l)}\right)$$

This function enables us to view our GCN model as a differentiable and parameterized extension of the 1-dimensional Weisfeiler-Lehman algorithm for graph-structured data.

C. GCN vs. Others

There are several other algorithms there to work with node classification problems such as clustering, eccentricity measure, centrality measure, skim-gram embedding and many more . But we choose to use GCN for our project since it has several advantages over those process . GCN works better when doing node classification. There are following advantages of using GCN that outweighs other algorithms benefits[11][12][13].

- 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.
- GCNs can efficiently handle large graphs by employing sparse matrix operations and parallelization strategies. This ability to scale allows them to excel in extensive node classification tasks where alternative methods may falter.

Overall, the unique capabilities of GCNs make them a powerful tool for solving node classification problems, as they can efficiently leverage the relational structure of graph data to produce meaningful and accurate node representations.

D. what we did differemntly

There has been large amount of works done on node classifications. There are many works we looked into and tried to analyze the result efficiently and differently by comparing and changing things such as in our GCN we have seen that using more node labels (e.g. 1000) give more accurate prediction of the model. Also, increasing layers and epochs iterations can lead to more feasible model. Also backpropagation and correct label propagation plays important roles too.

IV. Analytical Thoughts

Our project uses Graph Convolutional Networks (GCNs) to mine data from GitHub and classify users into specific skill demographics such as web developer or machine learning developer. The GCN model used in the project is trained and validated using 2% labeled data, which is approximately 800 developers or nodes. The data is preprocessed by creating an adjacency matrix out of the nodes, edges, and features, and a multi-layer model is used for the GCN graph. The GCN layers can be explained by an equation with an undirected graph, which involves a non-trainable part involving the normalized adjacency matrix and a trainable part involving the hidden state and the trainable weights. The label propagation algorithm is used to propagate labels to unknown nodes, and the loss function is calculated to fine-tune the model's parameters during the learning process.

To evaluate our model, we tested its accuracy. Here's the output:

```
ROC AUC: 0.8754371433466324
PR AUC: 0.7330231163326726
F1 score: 0.6731565901342934
[[0.94221426 0.03878774]
 [0.40207428 0.57973782]]
```

Fig. 3. Output of running test set on our model

From fig 3, we can see TP = 0.58, TN = 0.94, FP = 0.04, FN = 0.40. The GCN model achieved a ROC AUC score of 0.88 and a PR AUC score of 0.73 on the test set, which is quite impressive considering that the model was trained on just 400 labelled examples. The F1 score was 0.67, which is a good overall measure of the model's performance.

The matrix here summarizes the performance of the trained GCN model on the test set. The rows represent the actual class labels, and the columns represent the predicted class labels. In this case, the actual class labels and the predicted class labels are "ML developer" and "web developer". The matrix shows the number of true positives, false positives, true negatives, and false negatives, which are the number of correctly predicted positive samples, incorrectly predicted positive samples, correctly predicted negative samples, and incorrectly predicted negative samples, respectively. The matrix shows that the model has a relatively high true positive rate (TPR) of 0.58, which means that it correctly identifies the ML/web developers with a high degree of accuracy. The true negative rate (TNR) is also high at 0.94, indicating that the model correctly identifies non-ML/web developers with a high degree of accuracy as well. However, the false positive rate (FPR) is quite high at 0.04, which means that the model also classifies a relatively large number of non-ML/web developers as ML/web developers. The false negative rate (FNR) is also quite high at 0.40, indicating that the model misses a significant number of actual ML/web developers.

In addition to evaluating the performance of the Graph Convolutional Network (GCN) model, we also compared it to the performance of a Random Forest model that is trained and evaluated using the same data samples. This can help provide additional insights into the relative performance of the GCN model compared to other machine learning models. Comparing the results of the two models can help us identify any areas where the GCN model may need further improvement.

```
ROC AUC: 0.83201342653782
PR AUC: 0.6993878822672851
F1 score: 0.551906294830231
[[0.9422212 0.02013878]
 [0.57973737 0.40484252]]
```

Fig 4. Running test set on Random Forest model

From the figure above, we can see TP = 0.40, TN = 0.94, FP = 0.02, FN = 0.58 and that the GCN improves the ROC-AUC and PR-AUC by approximately 0.04, indicating that graph data contributes valuable information to the classification task.

To work with larger data sets and to run the experiment, we leveraged the full functionality of the StellarGraph API, which seamlessly integrates with Keras. Our experiment was conducted using 1000 labeled nodes, and we followed the same training procedure as before.

```
Epoch 1/200
1/1 - 4s - loss: 0.2291 -
Epoch 2/200
1/1 - 4s - loss: 0.2227 -
Epoch 3/200
1/1 - 4s - loss: 0.2170 -
Epoch 4/200
1/1 - 4s - loss: 0.2086 -
Epoch 5/200
1/1 - 3s - loss: 0.1944 -
```

Fig.5 Training and Validation Accuracy and Loss over epochs.

From the figure above, we see that training our model was a lot faster (in comparison to our earlier training)

Now, to evaluate our model, we tested its accuracy again. Here's the output:

```
ROC AUC: 0.8897047967718073
PR AUC: 0.7301466433429393
F1 score: 0.7346729390312858
[[0.93797299 0.07259011]
 [0.30207425 0.65231175]]
```

Fig. 6. Output of running test set on our improved model

We can see that the current matrix has TP = 0.65, FP = 0.07, TN = 0.94, FN = 0.30. Hence, it has a higher true positive rate (0.65) than the earlier matrix (0.58), which means that the current model is better at identifying positive instances. However, the current matrix also has a higher false positive rate (0.07) compared to the earlier matrix (0.04), which indicates that the current model has a slightly higher tendency to misclassify negative instances as positive.

Moving to the evaluation metrics, we can see that the ROC AUC and PR AUC scores have both improved in the current matrix compared to the earlier one, which indicates that the current model is better at distinguishing between positive and negative instances. The F1 score has also increased, which suggests that the overall performance of the model has improved.

Now we can see how the graph shows two different categories of skills (web developers or machine learning developer) with colors.

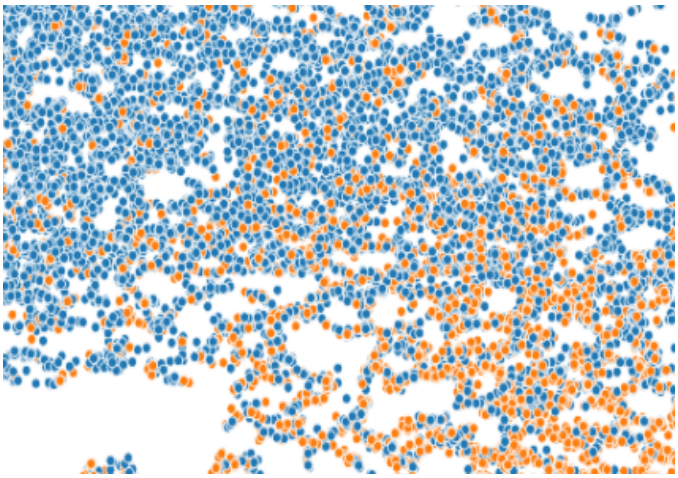


Fig 7. Embedded model showing ML and web developers.

We can gain insight into the features the model has learned by accessing the embeddings, which are representations of the input data in a lower-dimensional space. In this case, we are accessing the embeddings before the classification layer, which allows us to see how the model is grouping the data. We use the embedding model object to obtain the embeddings, and then visualize them using UMAP, a dimensionality reduction technique. The resulting plot shows that the two classes being predicted (web developers and machine learning developers) are largely separated, but there is some overlap in the center, which is to be expected as the two classes share some commonalities.

V. Conclusions, Limitations and Future Works

A. CONCLUSION

In conclusion our project has several important aspects that make it valuable. Firstly, it uses Graph Convolutional Network (GCN) models to mine data from social media, specifically GitHub, for specific skill demographics. This is a powerful technique that allows for efficient and scalable node classification in graphs, making it ideal for data mining tasks.

Secondly, the project uses a dataset collected from public APIs in June 2019, which contains nodes, edges, and features. This dataset has been preprocessed and used to train and validate the GCN model.

Thirdly, the project focuses on using a layer-wise propagation rule that operates directly on graphs, making it efficient and scalable. The label propagation algorithm is also used to reduce time complexity and provide a more accurate approximation while training nodes and feature matrices.

Finally, the project has demonstrated good performance in terms of classification accuracy and efficiency compared to other models. These aspects make the project valuable for data mining and classification tasks in social media networks.

B. LIMITATIONS

One of the limitations is that we only focused on web developers and machine learning developers.

Second limitation is that we only used data from June 2019, and the trends and patterns may have changed since then. In the future, we can update our dataset and incorporate more recent data.

Another limitation of our approach is that we have only used GitHub as our social media platform. There are many other social media platforms where developers may also have their profiles, and they may use different usernames on each platform. Therefore, it may not be easy to connect the profiles of the same developer across different platforms.

C. FUTURE WORKS

In future work, we can try the following:

We can try to identify more skills beyond web development and machine learning development.

We can explore using other social media platforms and see how our model performs on those platforms.

We can also try to improve the accuracy of our model by using more labeled data for training and validation.

Additionally, we can explore different techniques for feature extraction like selecting features that are highly correlated with the target variable. Discard weakly correlated features.
Removing the least important feature iteratively from the subsets(layers) of data will help to find features that are most relevant to the analysis task, and this can also help to reduce the false positive rate as well.

Acknowledgment

First and foremost, we would like to express our gratitude to our instructor, Dr. Carson Leung, for providing us with the opportunity to work on these projects. His teachings have imparted invaluable knowledge to us throughout this course, which will undoubtedly prove beneficial for all the students involved. We also extend our thanks to the teaching assistants, who have consistently offered helpful guidance and support throughout the course and in our project work. Additionally, we appreciate Musae for making the dataset available on Kaggle. Finally, we extend our appreciation to everyone else who has played a role in this course, making it a truly enriching experience for all of us.

References

[1] Hammond, D. K., Vandergheynst, P., & Gribonval, R. (2011). Wavelets on graphs via spectral graph theory. Applied and Computational Harmonic Analysis, 30(2), 129-150.

[2] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," https://doi.org/10.48550/arXiv.1609.02907, last revised 22 Feb 2017.

[3] Chen, M., Wei, Z., Ding, B., Li, Y., Yuan, Y., Du, X., & Wen, J.-R. (2021). Scalable Graph Neural Networks via Bidirectional Propagation. IEEE Transactions on Neural Networks and Learning Systems. [Online]. Available: https://doi.org/10.48550/arXiv.2010.15421

[4] Duarte, G. J. da S., Pereira, T. A., Nascimento, E. J. F., Mesquita, D., & de Souza Jr., A. H. (2021). How do loss functions impact the performance of graph neural networks? [online]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://sbic.org.br/wp-content/uploads/2021/09/pdf/CBIC_2021_paper_161.pdf

[5] Sekhar, Ch & Meghana, P. (2020). A Study on Backpropagation in Artificial Neural Networks. Asia-Pacific Journal of Neural Networks and Its Applications. 4. 21-28. 10.21742/AJNNIA.2020.4.1.03.

[6] Weisfeiler, B. and Lehmann, A. A., "A reduction of a graph to a canonical form and an algebra arising during this reduction," Nauchno-Technicheskaya Informatsia, vol. 2, no. 9, pp. 12-16, 1968.

[7] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Anchorage, AK, USA, Jul. 2019. Accessed: Apr. 18, 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3292500.3330925

[8] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation," in *SIGIR '20*, Virtual Event China, Jul. 2020. Accessed: Apr. 18, 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3397271.3401063

[9] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling," in *International Conference on Learning Representation*, Vancouver convention Center, Vancouver CANADA. Accessed: Apr. 18, 2023. [Online]. Available: https://arxiv.org/pdf/1801.10247.pdf

[10] I. Ullah, M. Manzo, M. Shah, and M. G. Madden, "Graph convolutional networks: analysis, improvements and results," in *Applied Intelligence (2022)*, Online, Nov. 2021. Accessed: Apr. 18, 2023. [Online]. Available: https://doi.org/10.1007/s10489-021-02973-4

[11] K. Sun, R. Zhang, S. Mensah, Y. Mao, and X. Liu, "Aspect-Level Sentiment Analysis Via Convolution over Dependency Tree," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, Hong Kong, China, Nov. 2019, pp. 5679–5688. Accessed: Apr. 18, 2023. [Online]. Available: https://aclanthology.org/D19-1569

[12] L. Yao, C. Mao, and Y. Luo, "Graph Convolutional Networks for Text Classification," in *AAAI-19 / IAAI-19 / EAAI-19 Proceedings*, Jul. 2019, vol. 33, no. 01, pp. 7370–7377. Accessed: Apr. 18, 2023. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.33017370

[13] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph Convolutional Neural Networks for Web-Scale Recommender Systems," in *24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, London, United Kingdom, Jul. 2018, pp. 974–983. Accessed: Apr. 19, 2023. [Online]. Available: https://doi.org/10.1145/3219819.3219890