Friend Recommendation with Graph Neural Networks

Team: 🐸

Gergely Ákos: EAWCFT

Kis Milán: API8EM

Szmida Patrik Péter: G0HPLP

Goal

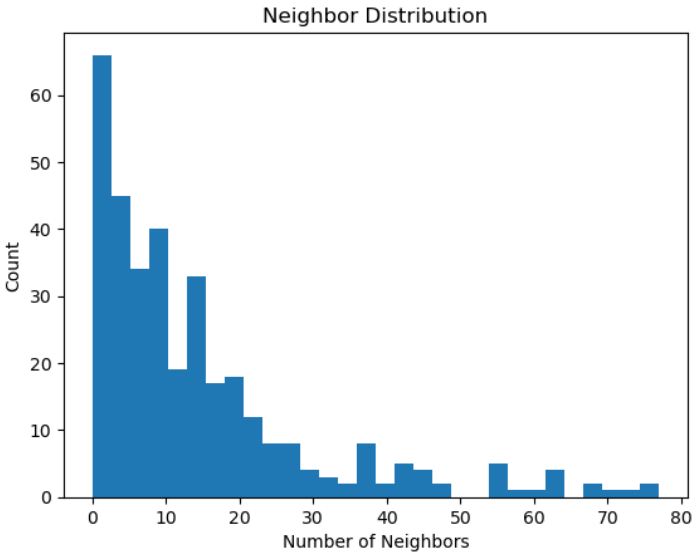
The goal of this project is to develop a personalized friend recommendation system by using Graph Neural Networks (GNNs). We aim to provide a friend recommendation service based on Graph Neural Networks. We will implement the node2vec paper (https://arxiv.org/abs/1607.00653), which is a graph embedding algorithm in order to recommend friends for each user.

Data Description

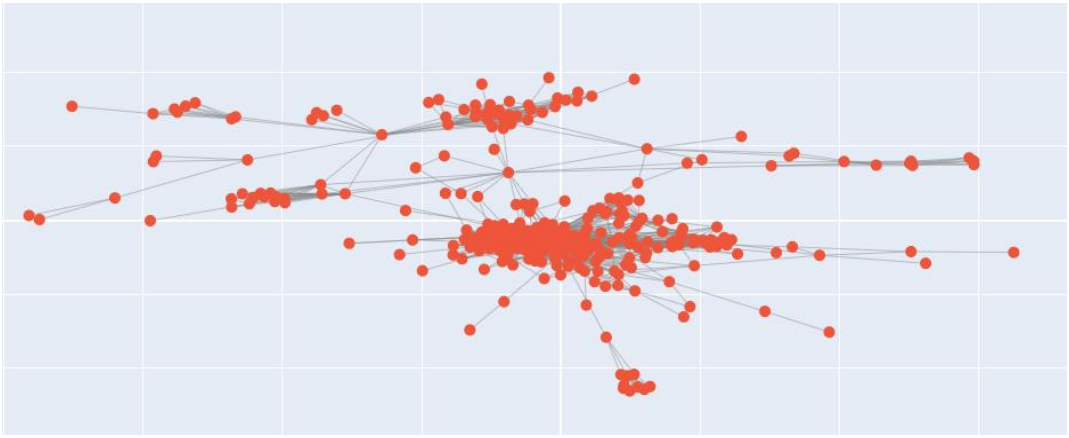
We utilized the Facebook Dataset available on the Stanford website.

The dataset comprises multiple graphs, where nodes represent individuals, and edges denote the friend connections between them.

Algorithm works for each graph in the dataset, but for exploratory data analysis, we investigated the 0.edges file:



Shape of Network:



This graph contains 333 nodes and 2519 edges.

Node2Vec Algorithm

We mapped the input node numbers to a continuous list that starts from zero and ends with the count of the nodes – 1.

After mapping, we created train and test splits by EdgeSplitter from Stellargraph.

We sample 10% of the edges for test and sample that many negatives as well. This will be used for evaluation.

From the remaining graph we sample 10% of the edges and that many negatives as well.

After selecting these edges, we split it into train and test in 75%-25% ratio. This will be used to find the best link classifier after projecting the nodes into embeddings.

Split	Number of Examples	Hidden from	Picked from	Use
Training Set	339	Train Graph	Test Graph	Train the Link Classifier
Model Selection	113	Train Graph	Test Graph	Select the best Link Classifier
Test Set	502	Test Graph	Full Graph	Evaulate the best Link Classifier

We generated random walks on the training graph and applied the Word2Vec algorithm to the resulting walking sequences. This process enables us to obtain embeddings for each node in the graph.

We created the input for the link classifier as follows:

- Get the embeddings of each edge nodes
- Used binary operator on these embeddings presented in the paper
- Train a Logistic Regression model whether it's a positive or negative edge

Operators can be found in the original paper:

Operator	Symbol	Definition
Average	$\boxplus$	$[f(u) \boxplus f(v)]_i = \frac{f_i(u)+f_i(v)}{2}$
Hadamard	$\boxdot$	$[f(u) \boxdot f(v)]_i = f_i(u) * f_i(v)$
Weighted-L1	$\|\cdot\|_{\tilde{1}}$	$\|f(u) \cdot f(v)\|_{\tilde{1}i} = \|f_i(u) - f_i(v)\|$
Weighted-L2	$\|\cdot\|_{\tilde{2}}$	$\|f(u) \cdot f(v)\|_{\tilde{2}i} = \|f_i(u) - f_i(v)\|^2$

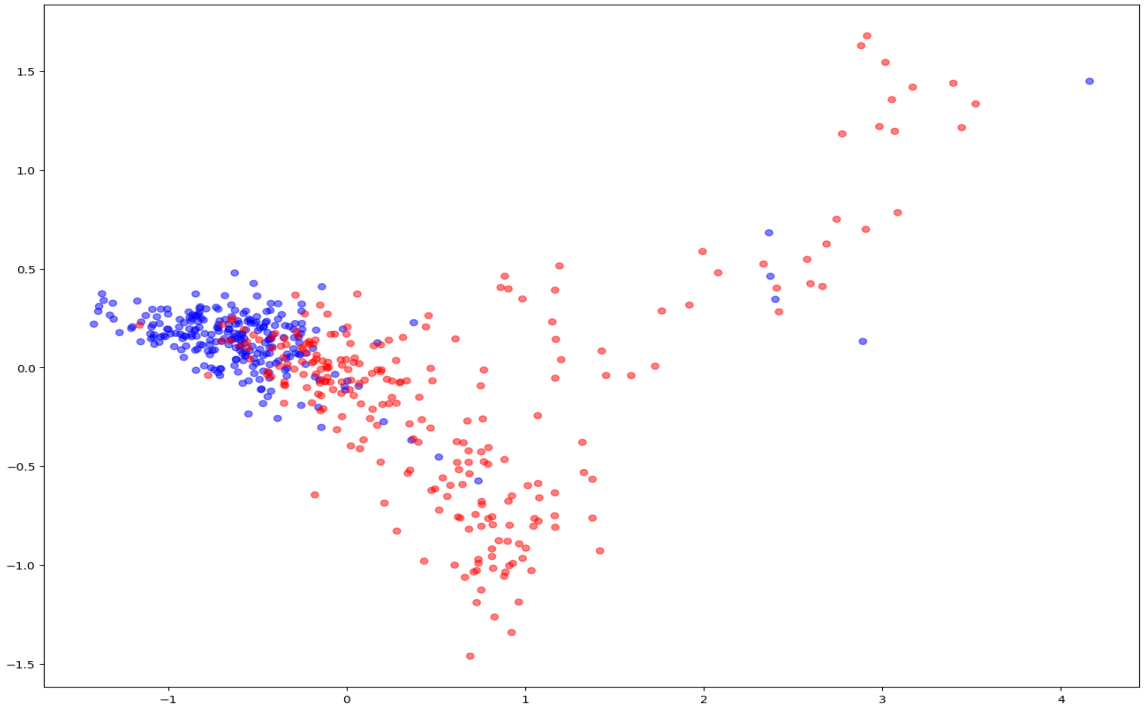
AUC scores for the following operators:

operator_hadamard	0.896
operator_l1	0.961
operator_l2	0.955
operator_avg	0.761

The best performing operator is the operator_l1.

Following the generation of embeddings on the entire graph, our Link Classifier Algorithm achieved an AUC score exceeding 0.92.

Visualization of positive and negative edges using a 2D PCA projection:

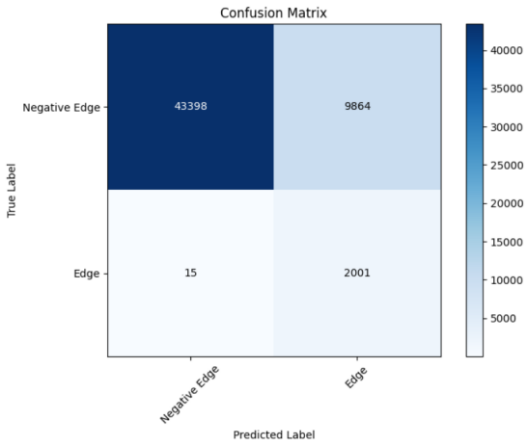


Hyperparameter Optimization

As we were dealing with a variety of graphs rather than a specific one, manual hyperparameter optimization was performed, focusing on the target graph, which was 0.edges. The primary emphasis during optimization was on refining the parameters related to random walks.

Evaluation

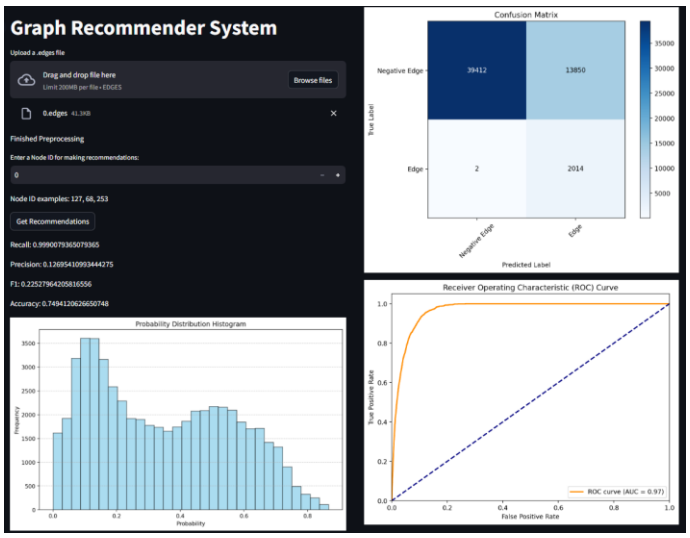
We used Accuracy, Precision, Recall and F1 score as evaluation metrics. We also plotted the ROC and the Confusion matrix (show in the figure below) after each model training.



The diagonal of the confusion matrix represents correctly identified positive and negative edges. The bottom-left section indicates positive edges misclassified as negative by the model, interpreted as errors. Conversely, the upper-right part reflects negative edges the model predicted as positive. This concept forms the basis of our friend recommendation solution: examining edges absent in the graph yet predicted by the model as present.

Streamlit GUI

We used the streamlit package to create a user interface for the trained model. First, we organized the codes related to the node2vec paper into functions and stored them in a python file named *common_functions.py*. Then we started creating the streamlit GUI by adding a file uploading interface accepting files with *.edge* type extension. Upon a successful upload, the UI used the main process function to split the graph, fit the model and predict the edges. The results are shown on the site when the process is completed: performance metrics like Recall, Precision, F1 score and Accuracy, along with the Confusion Matrix and ROC plots.



In this screen, an input for node selection is present as well, with the purpose of getting the top 5 recommendations for a specific node in the graph.

Containerization

We used Docker to containerize the created project. We created a *streamlit_demo.py* file with the same purpuse mentioned above: collecting and organizing GUI related code into a python file. We then continued by creating the Dockerfile which would allow us to build an image of the application. Since python 3.8 was used during the project, the official python docker image with the same version served as the base for us. In order to avoid version conflicts in the future, we collected the used python packages with version numbers into the *requirements.txt* file using the *pip freeze* command. Following this we constructed a docker file in which we installed the listed packages, exposed the port on which the streamlit application will be running, and launched the GUI. Using this dockerfile we constructed and run the docker image with the following commands:

- *docker build --rm -t dl_homework (Build image)*
- docker run --rm -p XXXX:8501 dl_homework (Run container)

In the latter, we set the port to which the container's exposed 8501 port was forwarded to. In the example code it's show as XXXX; it can be whatever the user configures (in our case we used 8501 here as well).

Conclusion

Our task was to develop a friend recommender system using graph neural networks. After having the necessary data, the initial step was to create train, validation and test sets from the graphs by sampling its edges and negative edges as well. Subsequently, we used the node2vec algorithm to get the node's embeddings from the training sets and trained a logistic regression model to predict edges. During the training process, we optimized the hyperparameter of the node2vec algorithm in order to achieve the best results. Moving forward, we evaluated the predictions and were able to achieve ROC exceeding 0.92. We finalized our project by creating a user interface for the model using the streamlit python package and containerizing it with Docker.

ChatGPT

ChatGPT was used for text formatting in some parts of this document.