Deep learning homework project - Friend recommendation

Anna Gergály (WWPD4V), Péter Mészáros (RBJNB7)

December 10, 2023

1 Introduction

The goal of this project is to develop a personalized friend recommendation system by using Graph Neural Networks (GNNs). We analyze data from Facebook, Google+, or Twitter to suggest meaningful connections based on user profiles and interactions.

This task is highly topical as recommender systems are all around us on the internet from targeted advertisements to friend and content recommendation; working with a larger and larger data pool every day.

1.1 Friend recommendation - link prediction

Graphs are incredibly versatile mathematical structures that are well-suited for representing a wide variety of real world data. This is also true for social networks: it is quite intuitive to represent data from a social media site as a graph.

The individual users of the site can be considered the nodes of our graph and the connections between them constitute the edges. The connections can be derived from user interactions, but most sites use some form of explicit signaling of user connections: adding people as friends, or following them. (These are undirected and directed forms of contact which is very relevant from a graph point of view.)

As such, our task of recommending friends when viewed from this standpoint becomes a task of predicting links (or edges) in a graph that are not currently present, but are likely to be present.

1.2 Deep learning on graph data

Graph data has a wide selection of applications, but it started being explored as an avenue for deep learning later than other types (e.g. images, text). This is because the complicated structure found in graphs poses unique challenges.

Earlier architectures often relied on known graph algorithms (e.g. random walk based methods), but today the most commonly used techniques are graph convolutional networks (which generalize convolutions often used in CNNs to graph data), and transformer-based architectures. The specific type of GNN layer we use in our model is usually abbreviated as GCN [2], it uses message passing, a fairly simple concept of aggregating neighbour information, but usually yields great results.

These solutions offer a very important advantage compared to more traditional graph algorithms, they are better suited for real world data, as we can add features to nodes, or even edges. In our case this means using information about the users and not just the current structure of the network.

The network is most often used to first create an internal representation of the individual nodes, usually referred to as node embeddings, which are then used to predict certain things about nodes, edges or the network as a whole.

The embeddings can be used at node level, to group nodes into certain categories, for node classification and clustering tasks; or even be aggregated over the entire graph for example for graph classification.

1.3 The link prediction task

In our case, where the main focus lies on the edges, the logical thing to do is to always consider pairs of nodes: based on the two node embeddings together we can use some sort of method to output some sort of rating about how likely we are to see an edge between them.

At this point our method can be quite simple as the graph structure is already encoded in the embedding and we only need to consider those. A commonly used method is to simply take the dot product of the embedding vectors, we explore using an MLP on the concatenated vectors.

2 Used datasets

The datasets we considered for this project come from the Stanford SNAP datasets[1] [3, 4]; all of them are publicly available, anonymized datasets. General information about the three datasets are shown on 1. The data was originally collected for research into predicting 'social circles', a sort-of node clustering task where nodes are allowed to belong to multiple clusters.

This is very different from our task, however the collected data about the network structure and node features is also useful for our case. Due to the original task of the datasets, they have a special feature: they are all EGO networks. This means it is a network formed around a single individual. The interesting things this means for our use-case is that the ego is already connected with every node present in the graph, and that every node has at least one common neighbour.

Table 1: General information about the graphs in all datasets.

		Number of	Number of nodes		Number of edges	
Dataset	Type	graphs	Average	Median	Average	Median
Facebook	Undirected	10	417.7	228	17850.8	5732
Google+	Directed	129	1991.5	1485	238383	78763
Twitter	Directed	973	138.6	141	2625.5	2046

2.1 Overview of the datasets

2.1.1 Facebook

The Facebook dataset is has quite a few differences from the other two. It is the most well-annotated and most complete dataset as it was not scraped from an existing social network, but rather created by the researchers for this study. This means users were urged to complete their profiles with more and more precise information and the answers were more closely monitored.

Node features were collected in 26 different categories including birthdays, collegaues and political affiliations. Users were also tasked with identifying circles in their friend group.

Due to the nature of Facebook friendships, the links in this dataset mean mutual acquaintance; the resulting graph is an undirected one, which means this dataset needs to be treated differently in certain aspects than the other two.

This dataset is on the smaller side with only 10 ego networks and 400 nodes per network on average, containing 4039 users overall.

2.1.2 Google+

This dataset was scraped from the actual Google+ website and as such it is a lot less supervised, but contains more data: 133 ego networks containing 106674 users. Google+ has a 'follow' mechanic and as such the relationships in this dataset are directed.

For node features certain parts of user profiles were scraped, the dataset contains information from six different categories (e.g. gender, job titles, universities).

[1] `https://snap.stanford.edu/data/`, last accessed on: 9th December 2023.

2.1.3 Twitter

The Twitter dataset contains 1000 ego networks also based on the site's follow mechanic; these are on average much smaller than the those of the Google+ dataset, containing only 81362 users.

As Twitter does not have de-facto user profiles with publicly available information, tweets from the previous two weeks were used to construct the user features. The person's used hashtags and mentions were recorded in two separate categories.

2.2 Data preparation

The data in each of the datasets was saved in a similar format, with five files per ego network. The "edges" file contains `nodeId` pairs, each of them constitute an edge; the "circles" file contains the circles of the ego node, where in each line there is a list of nodeIds corresponding to a circle; the "featnames" file contains the name of the features; the "feat" file contains the node features as multi-hot vectors, features are in the same order as in the "featnames" file; "egofeat" file contains the feature vector for the ego node. It is assumed that the ego node is connected to every other node.

For dataloading we originally planned to use the available SNAP dataset loading functions from the Pytorch Geometric[2] library, but this turned out to not be possible to use as is, because in the case of Google+ the node features are not read, and in the case of Twitter memory constraints were the limiting factor. The data has a very high dimensionality as is, due to how the features are formatted from the available data. This original loading code foregoes dealing with this problem, causing it to either crash the system due to lack of sufficient RAM or not load the features of nodes at all. In each dataset, each ego network has different feature names, and if all the graphs are prepared to node feature vectors of the same dimension, then it will became very high dimensional and sparse.

To deal with this problem we looked at the loading process from Pytorch Geometric and augmented the code with some techniques (see Section 2.3 After this, the data was in a Pytorch Geometric compatible dataloader.

We used composited Pytorch Geometric transforms to get the data ready to feed into a model, normalizing features and transferring the data to the correct Pytorch device. The RandomLinkSplit function was used to make the data compatible with the link prediction task.

This function is used to add negative edge samples into the graph with the correct labels to have negative examples when training, and also to change the edges used for message passing. This is useful to remove the edges that will be used as positive examples: having all the edges present then makes the task easy on the test set but does not help the model learn to generalize. It can also be used to separate the training validation and test sets within a graph. The number of sampled negative edges was equal to the number of positive edges.

We chose to not use it in that way, since we had access to multiple graphs in each of the datasets and so separating these at graph level made more sense.

2.3 Dimensionality reduction

The following techniques were used for dimensionality reduction.

2.3.1 PCA

PCA (Principal Component Analysis) is a common technique for dimensionality reduction, where the data is transformed into a new coordinate system and approximated by the first few principal components. We used this for the Facebook dataset, keeping 50% variance, which yielded a 28-dimensional feature vector instead of 1406. For the other datasets, the feature vectors could not fit into the memory, so PCA could not be utilized.

[2] `https://pytorch-geometric.readthedocs.io/en/latest/`, last accessed on: 9th December 2023

2.3.2 Ego difference

This method is based on [4]. In the paper, mainly edge features are contructed, but that is unsuitable for our case, because we predict edges, and would require node features. Difference vectors are constructed from the user profiles (which are the published node feature vectors), which are still high-dimensional. To reduce their size, the features are categorized into profile categories (e.g. high school, gender, work), and the value corresponding to each profile category is the sum of the difference values in the children features. In the case of the Twitter dataset, the two categories are hashtags (features starting with #) and mentions (features starting with @). In [4], both difference vectors between two users and the difference between a user and the ego user are used.

To construct node feature vectors, similar features to the ego are considered. For a given node, the corresponding feature vector would be the vector of similarities compared to the ego node. This has limitations, because the ego has high influence on the feature vectors. ϕ_x is the feature vector for node x; e is the ego node, l is an arbitrary feature, p is a profile category, $Category(p)$ is the set of features aggregated into p category.

$$\delta_{x,y}[l] = \begin{cases} 1 & \text{if } x \text{ and } y \text{ share the feature } l \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$\phi_x[p] = \sigma_{x,e}[p] = \sum_{l \in Category(p)} \delta_{x,e}[l] \tag{2}$$

As a result, Facebook, Google+ and Twitter datasets got a feature vector of size 26, 6, and 2, respectively. An advantage of this is that the number of features are constant in a dataset, and does not depend on the ego network itself.

2.3.3 Feature name clustering

The Twitter dataset has many similar features, which are often not hat different from each other (e.g. hashtags containing typos). The idea in this was to create an embedding using an LLM for each feature name and cluster them based on the embedding using KMeans clustering algorithm. The embeddings were created using the Hugging Face[3] library with the model named `nlptown/bert-base-multilingual-uncased-sentiment`, which can be found in the Hugging Face model zoo. 50 was chosen as the number of clusters, which highly reduces the number of dimensions. Expermenting with the number of clusters was not possible, due to the long runtime of the clustering algorithm. The node vectors were created such that the ith value is the number of features that node has which were put into the ith cluster.

2.4 Data split

Splitting the data into train, validation and test subsets could be done two ways, the main difference being whether the three sets are on different graphs or on the same graph [1]. We opted for the former one, because the practical application of the model would revolve around supplying the model with a new graph (friend ego network) and then recommending new friends. Putting different graphs into each subset emphasises this concept, because the test set has previously unseen graphs. We think that in this case, this is also the more beneficial approach, which involves exploiting the fact that the datasets consist of many graphs.

3 Used models

3.1 Model architecture

Our model uses the GCN graph convolutional layers to create node embeddings for each node [2]. Between each of the GCN layers we use ReLU activation and dropout. Both the number of layers, and the dimensionality of the hidden layers, and the resulting embedding are variable in

[3] `https://huggingface.co/`, last accessed on: 9th December 2023

size and have been tried in multiple configurations during parameter optimization. For predicting on individual edges, we take the created embeddings of two nodes and concatenate them before passing it to an MLP.

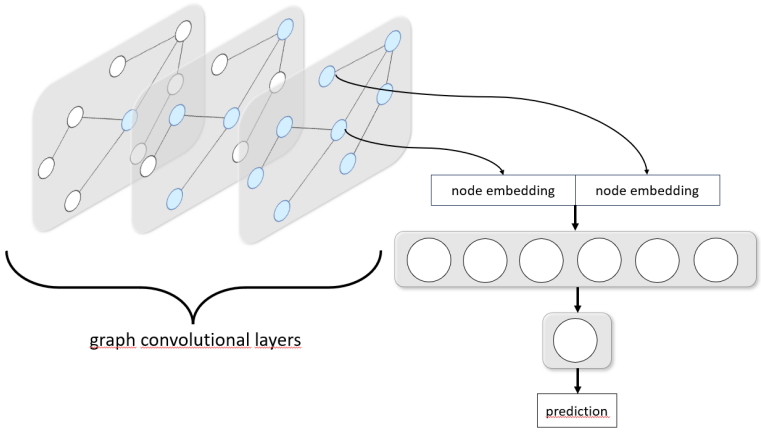


Figure 1: Overview of the model architecture.

This MLP contains two fully-connected layers, with ReLU activation and droput. The size of the hidden layer's dimension is also configurable. The output layer has a single output which has a sigmoid activation function for the binary classification task.

3.2 Training method

For each of the datasets / dataset variations we trained multiple models, while using WandB to perform a hyperparameter sweep.[4]. In our training function we perform the node encoding on each graph of the train set while running the classifier part on previously selected node pairs of the graph. As described in Section 2.2, these node pairs contain none of the edges present during message passing and have exactly the same amount of positive and negative examples. We used binary cross-entropy as our loss function between our prediction and the real edge label, and the ADAM optimizer during training.

Our hyperparameter sweep contained several important features: as previously mentioned, we tuned the dimension and number of the used GCN layers, the hidden dimension of the fully connected layers, as well as the size of the node embeddings. Other than this, we tried multiple configurations for the optimizer, both in terms of learning rate and weight-decay, and experimented with multiple values for dropout. The exact values for the sweep can be seen in Table 2. We used the bayesian sweep defined in WandB.

Table 2: Values used in the hyperparameter sweep.

Parameter name	Values
dropout	[0.3, 0.4, 0.5]
hidden_dim	[32, 64, 128, 256]
hidden_dim_linear	[32, 64, 128, 256]
learning_rate	(0; 0,001) uniform distribution
node_embedding_dim	[32, 64, 128, 256, 512]
num_layers	[2, 3, 4, 5, 6]
weight_decay	(0; 0,001) uniform distribution

[4] `https://docs.wandb.ai/guides/sweeps`, last accessed on: 9th December 2023

3.3 Baseline model

For the baseline model we wanted to select a particularly simple solution as advised. Since our model works by computing the probability of an edge existing, we wanted our baseline model to operate on a similar principle. The chosen baseline model is logistic regression, which predicts if an edge exists between a pair of nodes based on their feature vectors. This omits the other possible latent information about the graph's structure, which a GNN encodes in a node embedding, and also a logistic regression model is more simple than a neural network.

4 Evaluation

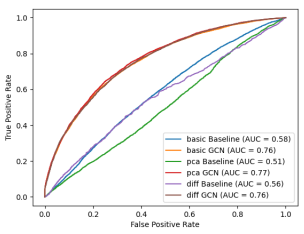
We measured the accuracy, precision and recall of our model compared to the baseline and created the ROC curves, precision-recall displays and the confusion matrixes. Table 3 contains the results on the test sets on each dataset and feature type for the baseline and the GCN models. In the table, "Basic" feature refers to when, no additional dimensionality reduction is done, "PCA", "Diff", and "Cluster refer to the methods mentioned in Section 2.3.1, Section 2.3.2, and Section 2.3.3, respectively. In every case, the GCN performed better then the baseline. In the case of the Facebook and Twitter dataset, where we experimented with multiple methods regarding the features, there were no difference in Accuracy, ROC and AP.

Table 3: Results on the test sets for each dataset and feature type.

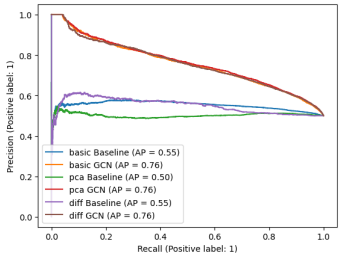
Dataset	Feature	Model	Accuracy	AUC	AP
Facebook	Basic	Baseline	0.55	0.58	0.55
		GCN	0.65	0.76	0.76
	Diff	Baseline	0.51	0.56	0.55
		GCN	0.58	0.76	0.76
	PCA	Baseline	0.51	0.51	0.50
		GCN	0.64	0.77	0.76
Google+	Diff	Baseline	0.55	0.56	0.54
		GCN	0.71	0.78	0.76
Twitter	Diff	Baseline	0.58	0.60	0.57
		GCN	0.66	0.77	0.78
	Cluster	Baseline	0.58	0.60	0.55
		GCN	0.66	0.77	0.78

4.1 Facebook

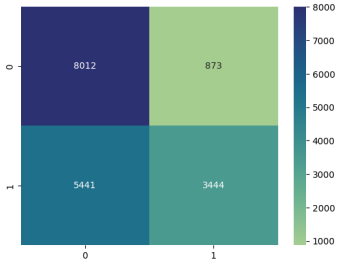
The following figures depict the ROC curves and Precision-Recall curves for the approaches on the Facebook dataset. The confusion matrix corresponds to the "Diff" method. Based on the first two curves, the GCN methods perform almost the same regardless of the used dimensionality reduction method. From the confusion matrix, it can be observed that the higher uncertainty is in the case of the negative examples.



(a) ROC curve.



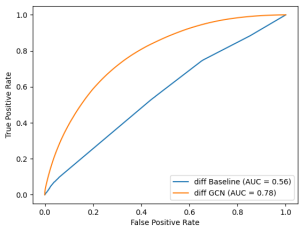
(b) Precision-recall.



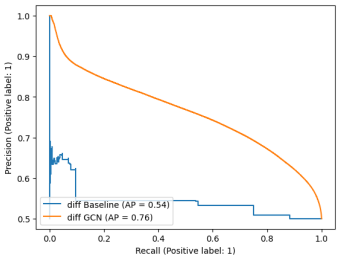
(c) Confusion matrix.

4.2 Google+

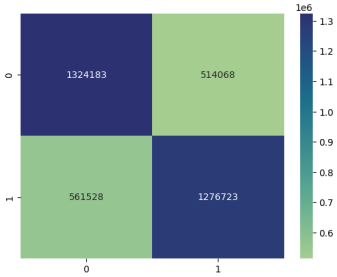
The following figures depict the ROC curves and Precision-Recall curves for the approaches on the Google+ dataset. The confusion matrix corresponds to the "Diff" method. As opposed to the other two cases, here, the confusion matrix is balanced.



(a) ROC curve.



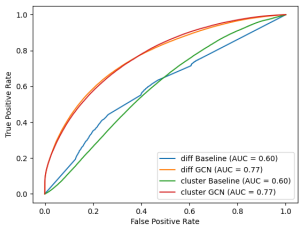
(b) Precision-recall.



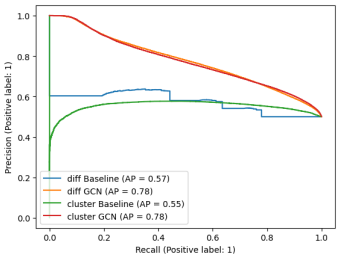
(c) Confusion matrix.

4.3 Twitter

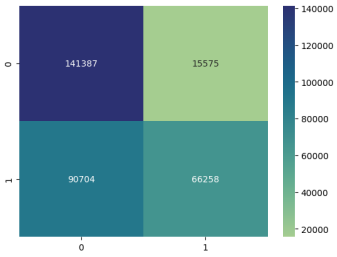
The following figures depict the ROC curves and Precision-Recall curves for the approaches on the Twitter dataset. The confusion matrix corresponds to the "Diff" method. Based on the first two curves, the GCN methods perform almost the same regardless of the used dimensionality reduction method. From the confusion matrix, it can be observed that the higher uncertainty is in the case of the negative examples.



(a) ROC curve.



(b) Precision-recall.



(c) Confusion matrix.

5 User interface

We used Gradio[5] to create an easily usable user interface for our project. It can display a graph uploaded in .json format and suggest 3 friends to any selected nodes, that they did not know beforehand. We achieve this by calculating the score of this node with each of the non-neighbouring ones and choosing the top 3. The interface can be seen on Figure 5. We created json files in the correct format from one graph in each of the datasets, which are available on the interface as examples.

6 Conclusions

Our solution outperforms the baseline solution in every metric. While the achieved accuracy is still quite low, the nature of the task makes this not as important: to provide a few recommendations to a user, we only need the nodes with the highest scores to be good recommendations.

Generally, this is a pretty difficult task for a model, especially with the limitations we chose: not using positive edges reserved for supervision during training, and making the training validation

[5] `https://www.gradio.app/`, last accessed on: 9th December 2023

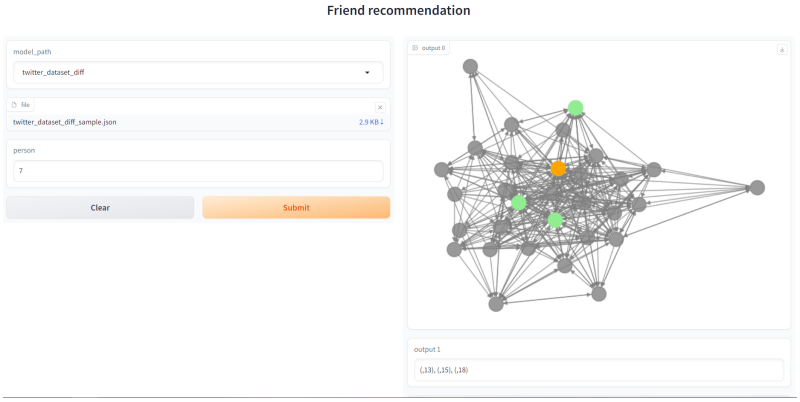


Figure 5: Our user interface. The orange dot is the selected user, the green ones are the proposed friends.

and test sets contain entirely separate graphs. But we believe these are important choices, as they are needed to help the model generalize better and model its use-case more closely.

Maybe the hardest part of the project was working with the dataset: the data and file structure are hard to understand and parse correctly, especially as the high dimensionality of the data needs to be addressed. We tried a few methods, largely inspired by authors of the original paper, but we did not feel completely satisfied with any of them: they are obviously much better suited for the social circle identifying problem the authors of the paper were trying to solve.

It is hard, however, to try to approach the data in other ways, as it seems the data collection was already excecuted with this data structure in mind, and it is very hard to parse in any other meaningful way that would also allow new datapoints to be added if needed.

All in all we believe the pipeline could be best improved by changing the approach of the data preparation / dimension reduction to a structure that is better suited to the task at hand.

References

[1] *Graph: Train, valid, and test dataset split for link prediction*. Aug. 12, 2021. URL: `https://zqfang.github.io/2021-08-12-graph-linkpredict/` (visited on 12/09/2023).

[2] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: `1609.02907 [cs.LG]`.

[3] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. URL: `http://snap.stanford.edu/data` (visited on 12/09/2023).

[4] Jure Leskovec and Julian Mcauley. "Learning to Discover Social Circles in Ego Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: `https://proceedings.neurips.cc/paper_files/paper/2012/file/7a614fd06c325499f1680b9896beedeb-Paper.pdf`.