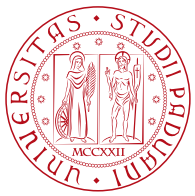

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Learning from Networks: Final Report

Predicting Global Clustering Coefficient with Graph Neural Network

Federico Pivotto, Fabrizio Genilotti, Francesco Boscolo Meneguolo

1 Introduction

In graph analytics, one of the most important graph-level metrics is the global clustering coefficient. This statistic measures the tendency of the graph to 'cluster' and can provide valuable insights in many real-world applications. The literature offers many methods with different definitions of clustering coefficient and propose different algorithms for its computation, which can either yield exact results or approximated estimates. In particular, one of the most widely studied problem is the computation of graph analytics for large graphs. In this report is proposed an approach based on a Graph Neural Network (GNN) model to compute the global clustering coefficient. The research aims to contribute to the development of a fast and reliable model for predicting the global clustering coefficient. The experimental results highlight the difficulty of this regression task, particularly the high risk of underfitting in small models when working with datasets of graphs of different sizes and domains under limited computational resources. However, this work may enable future advancements in methods for scaling GNN models for large graphs.

1.1 Questions

1. Is it possible to guarantee a good accuracy in average with a better inference time with respect to the exact/approximate algorithms?
2. Is it better to predict directly the global clustering coefficient or the number of triangles in a graph with a GNN?
3. *Optional*: Is it possible to train a GNN on small graphs and obtain a scalable model efficiently working also on large graphs?

2 Problem Formulation

This project aims to design and train a Graph Neural Network (GNN) for predicting the global clustering coefficient of an undirected graph, using the graph's structure and its node feature vectors as input. This work focuses on predicting the clustering coefficient rather than the number of triangles, as predicting a bounded value is often simpler than predicting an unbounded one.

2.1 Datasets

The datasets selected for this experiment come from different application domains for the training and testing phase of the GNN. The datasets are the following:

- ENZYMES: it is a dataset of protein tertiary structures consisting of 600 enzymes from the BRENDA enzyme database.
  `https://networkrepository.com/ENZYMES.php`
- D&D: it is a dataset of 1178 protein structures; each protein is represented by a graph, in which the nodes are amino acids and two nodes are connected by an edge if they are less than 6 Angstroms apart.
  `https://networkrepository.com/DD.php`
- COLLAB: it is a dataset describing scientific collaborations defined by the edges.
  `https://networkrepository.com/COLLAB.php`

NOTE: For computational reasons, the `REDDIT-MULTI-12K` dataset was discarded because of the excessively large number of graph samples.

	Nodes	Edges	Graphs
ENZYMES	19.5K	74.6K	600
D&D	334.9K	1.7M	1178
COLLAB	372.5K	49.1M	5000

TABLE 1: Networks dataset statistics

2.2 Libraries

This project leverages the `PyTorch Geometric` [Fey & Lenssen, 2019] library to train, validate and test the GNN model, and the `graph-tool` [Peixoto, 2014] library for computing the global clustering coefficients of the graphs.

3 Graph Representation

This section presents how each graph is represented and stored for training the GNN.

3.1 Ground Truth Computation

Initially, to assess this regression task, the true clustering coefficients must be computed for all the graphs in the three considered datasets. The exact clustering coefficients are calculated using the `graph-tool` library. Once all the labels are computed, they are stored in a file `.global_cc`.

3.2 Node Features Computation

To compute the node feature vectors, the `Node2Vec` [Team, 2024] approach provided by `PyTorch Geometric` was used. This is a shallow embedding method employed to generate node feature vectors for all the nodes in a graph. Due to the large number of samples in the datasets, small feature vectors were computed for each node in each graph using a limited number of epochs and optimized parameters that prioritize faster embedding evaluation. Specifically, each node embedding consists of five features, each computed using a small number of random walks of short and constant length. Despite this, the node embeddings remain informative, as clustering coefficient information is often captured by the local neighborhood. Furthermore, to enhance the node embeddings, the normalized node degree relative to the total number of nodes is concatenated to the embeddings. This additional feature can be useful during the training phase of the GNN, as the model could exploit the information regarding the density of each node's neighborhood.

3.3 Graph Tensor Representation

The graphs in the datasets are stored as `PyTorch Geometric` tensors. Each tensor contains the adjacency matrices, the true clustering coefficients and the node feature vectors.

Graph adjacency matrix To store the adjacency matrix in a space-efficient manner, a sparse COO tensor is used to store the edge indices. This data structure is designed to efficiently exploit memory when dealing with sparse graphs.

Node feature vectors Each tensor contains the corresponding node feature vectors. Each feature vector is composed of six elements: the first five are obtained by computing the node embedding using the `Node2Vec` approach, while the last feature represents the normalized degree of the node relative to the total number of nodes.

4 Clustering Coefficient Prediction Approach

This section presents how the graph neural network model was defined to solve the clustering coefficient prediction task.

4.1 GNN Model Architecture

The design of the graph neural network is inspired by the expressive power of the Graph Isomorphism Network (GIN) layer [Xu et al., 2018]. The architecture consists of an sequence of five GIN layers, followed by a global mean pooling operation over the node feature vectors and final fully connected layer (MLP) to predict the global clustering coefficient. The design leverages the power of the GIN layer to extract informative representations of the node features, enabling the model to distinguish between feature vectors of nodes in dense local neighborhoods and those in sparsely connected regions. Batch normalization is applied at the end of each layer to prevent numerical instability in the features and to reduce dependence on the initialization of the model parameters. In Table 2 are presented the model architecture details.

Layer(s)	Description	Output Channels
1	GIN (BatchNorm + Dropout(0.2))	128
2–4	GIN (BatchNorm + Dropout(0.2))	64
6	MLP	1

TABLE 2: Description of layers and their respective output channels.

Graph Attention Layer Furthermore, the Graph Attention Network (GAT) layers [Veličković et al., 2017] were excluded from the model design because the model's ability to assign weights to information propagated from neighboring nodes did not effectively enhance training. This may be because all the neighbors provide equally important information when addressing topological features.

5 Experiments and Results

This section describes how the graph neural network model was trained, validated and tested for the prediction of the global clustering coefficient.

5.1 Model Training and Test Procedure

The graphs in the datasets were converted into `PyTorch Geometric` Data objects, which were then divided into 80% for training, 10% for validation and the 10% for testing. The model training phase is conducted over a maximum of 1000 epochs, with an early stopping patience of 100 epochs (i.e. the number of consecutive epochs without validation performance improvements). The learning rate was initialized to 0.001 and reduced by a factor of 0.1 every quarter of the maximum epochs using a step scheduler. The `SparseAdam` optimizer from `PyTorch Geometric` was employed to optimize the model. The mean squared error (MSE) was used to measure the performance.

5.2 Final Results

The final model obtained from training was observed to underfit the data, achieving a test score of 0.1473. In Figure 1 is shown the model's difficulty in fitting the data. While the test score alone is an initial indicator of the model accuracy, it does not fully capture the model's performance. To gain deeper insights, in Figure 2 are compared the model's predictions with the ground truths over the test samples. It is evident that the model tends to predict nearly constant values, which are close to the ground truths for most samples. Indeed the model struggles in predicting values for samples with ground truth that differ significantly, in particular those greater than or equal to 0.5.
The model underfitting can be attributed to several factors. First, the problem is complex, requiring the identification of densely connected regions and distinguishing them from sparsely connected ones. Secondly, the graphs in the datasets are from various domains and can have really different sizes, ranging from just few nodes to few thousands of nodes. Lastly, the provided node feature vectors are limited in size and may lack sufficient information to effectively perform the regression task. Therefore, designing a small model may fail to generalize over all graph samples, resulting in inaccurate predictions of the clustering coefficient.

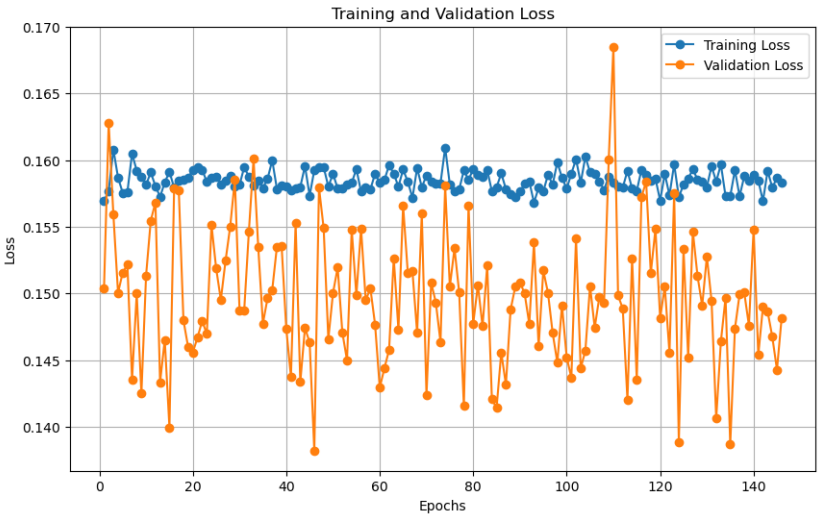

FIGURE 1: Plot of model training epochs, with both training and validation loss.

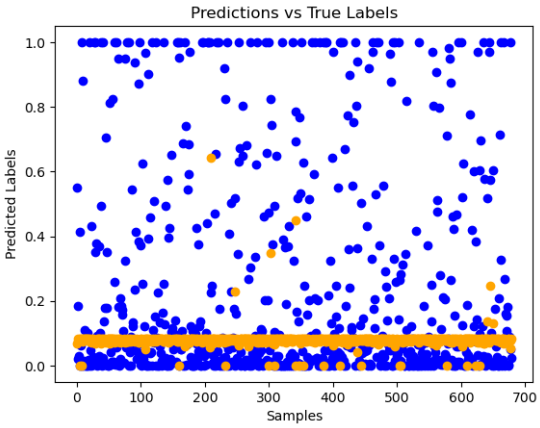

FIGURE 2: Plot of model prediction (in orange) vs ground truth (in blue) over the test samples.

6 Conclusions

In conclusion, training a small model to predict the global clustering coefficient of a graph resulted in underfitting the data. This phenomenon can be attributed to several factors, including the complexity of the problem, the diverse nature and varying size of the graphs in the datasets and the limited dimension of the node feature vectors.
To address these challenges and to improve performance in this regression task, it could be interesting to compute larger node and more informative feature vectors, such as including additional node-level analytics. Additionally, limiting the training, validation and test phase to smaller graphs (in terms of number of nodes) could help to simplify the task. Finally, sampling graphs from the datasets to create a more uniform distribution of ground truth values may further enhance the model's ability to generalize.
In the future, this study may be expanded to design a model that could scale efficiently on large graphs and to compare the computation time and precision of various approximation algorithms.

6.1 Machine for experiments

The machine used to train and test the proposed GNN model has the following hardware specifications:

- *CPU*: Intel i5-9600K
- *GPU*: NVIDIA RTX 2060 SUPER
- *RAM*: 32GB DDR4

7 Github Repository

One can view the current state of the project at `https://github.com/Fabrifio/graph-gcc-net`

8 Contributions

Federico Pivotto Implementation of graph label generation for a given dataset, using `graph-tool` to compute the exact global clustering coefficients. Node2Vec implementation in `PyTorch Geometric` for node feature vector computation.
Contribution time: 32%

Fabrizio Genilotti Implementation of the `Pytorch` datasets loading and split and the pipeline for training, validate and test the GNN model. Design specification for the GNN model. Implementation with plots of model training and test, along with a test-only Jupyter notebook.
Contribution time: 35%

Francesco Boscolo Meneguolo Implementation of graphs adjacency matrices conversion to sparse tensor for a given graph. Definition of Node2Vec implementation flow and final node feature vector structure. Definition in `Pytorch` of the GNN model architecture (i.e. the class).
Contribution time: 33%

References

[Fey & Lenssen, 2019] Fey, M. & Lenssen, J. E. (2019). Fast graph representation learning with pytorch geometric. http://arxiv.org/abs/1903.02428.

[Peixoto, 2014] Peixoto, T. P. (2014). The graph-tool python library. http://figshare.com/articles/graph_tool/1164194.

[Team, 2024] Team, P. (2024). Node2vec. https://pytorch-geometric.readthedocs.io/en/2.5.1/generated/torch_geometric.nn.models.Node2Vec.html.

[Veličković et al., 2017] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2017). Graph attention networks. https://arxiv.org/abs/1710.10903.

[Xu et al., 2018] Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2018). How powerful are graph neural networks? https://arxiv.org/abs/1810.00826.