Assignment2 and Project of
CS 410/510: Mining & Learning on Graphs

Instructor: Yu Wang, **Website**: https://ml-graph.github.io/fall-2024/,

Due: 12/10/2024 23:59 PST

Assignment2 has two parts: the homework and the project. For the homework, it is meant only for you to complete. In other words, you should not work with others in the course or seek help from others besides the instructor. For the project, you need to either find the partners to form the team(at most 4 people in a team) or complete the project by yourself. If you form the team, you need to explicitly mention which team members you are working with and each team could submit just one project report across all the team members. Additionally, if you use AI assistants, this is fine, but please acknowledge this somewhere in your submission (and know that you're taking full responsibility for your submission).

Please submit your solutions on Canvas. For homework 1-4, please submit your results in the pdf form. For the code, please submit a Python Source File or Jupyter Notebook. Do not include any downloaded datasets you used for your project in the submission file.

There will be a deduction of 10 points if submitting other formats besides the ones mentioned above (e.g., submitting .docx instead of a pdf or .zip combining multiple files).

1. [10 points] Package Setup

   (a) [2 points] Basic Package Setup: Install Python and Pip on your computer, and then install basic Python package, including Numpy, Matplotlib, Seaborn, Pandas.

   (b) [3 points] PyTorch Setup: Install PyTorch following instruction here. Choose the appropriate version based on your OS. I suggest using pip install instead of conda.

   (c) [2 points] PyTorch-Geometric Setup: Install PyTorch-Geometric following instruction here. Choose the appropriate version based on your OS and Pytorch Version. I suggest using pip install instead of conda.

   (d) [3 points] Test your Pytorch and Pytorch-Geometric by running examples here to ensure the installation is all correct here.

1. [30 points] **Node Classification under Different Homophily and Heterophily.** In this question, you will write code to perform node classification on graphs with different levels of homophily and heterophily using different models.

(a) [5 points] Dataset collection and analysis: Please download two data sets, Cora and Citeseer, according to here and two datasets, Texas and Cornell, according to here.

Calculate the number of nodes, edges, and average degrees. In addition, describe the physical meaning of each of these networks. What do the nodes, edges, and features represent?

(b) [10 points] Implementing the Multi-Layer Perceptron, Label Propagation, and Graph Neural Networks based on what you have learned from the class and training the above three models, respectively. You will need to check the validation accuracy to save the model and finally report the testing performance of the model that achieves the highest validation accuracy.

- (1) Please report by drawing the Figure showing the change in training accuracy, validation accuracy, and testing accuracy relative to the training epoch for each dataset for each model (the y-axis is the performance, and the x-axis is the training epoch). **Note that the goal here is not to achieve the best performance, so you don't need to heavily hyper-parameter tuning your model but just achieve a reasonable performance to show your capability of coding Graph Machine Learning Models**.
- (2) Compare the performance of different models on the same dataset/network. What conclusions/insights can you draw?

(c) [15 points] Can you use the Stochastic Block Model (SBM) to simulate networks with different homophily, conduct node classification using Multi-Layer Perceptron and Graph Neural Networks, and report the node classification performance?

- (1) Assuming we only have three classes/communities in SBM and each community we have 100 nodes.
- (2) You can specify the intra-edge probability and inter-edge probability to control how many edges there are between nodes within the same class/community and how many edges there are between nodes across different classes/communities. Please try to synthesize networks with homophily from very low to very high and then visualize the generated network.
- (3) Sample node features from Gaussian Distribution (assume we have node feature of dimension 2, then you would need to sample your data points from a 2D

Gaussian distribution by specifying the mean and covariance, check this command: np.random.multivariate_normal). In addition, you need to specify three different means for your three classes. For covariance, we assume they are all the same. Please use TSNE visualize to visualize your generated node features in the 2D plot (python matplotlib.pyplot will help you with visualizing that).

- (4) Calculate the homophily value for each of your simulated networks.
- (5) Train your MLP and GNN models, respectively, on each of your simulated networks and obtain the training, validation, and testing performance. Visualize the training, validation, and testing performance of MLP and GNN on each dataset.
- (6) Compare the testing performance of MLP and GNN models with the network homophily. What kind of conclusion can you draw?

2. [30 points] **Recommender System** Please use Python and parts of the RecBole library. Here, we will run some recommender system algorithms, including some from very recent years that are near state-of-the-art methods. These methods will vary in terms of their model architecture, loss functions, and additional information they might leverage beyond just the network structure (e.g., knowledge graph).

- (1) [5 points] Installing RecBole and getting started As the RecBole library has some compatibility issues (e.g., with recent Numpy versions), please create a fresh virtual environment before attempting to use the library to better ensure there will not be issues. Note that I had used Python 3.10.10, but I believe other Python 3 versions will also work (assuming PyTorch is already installed). Then, install the following:

  pip install recbole
  pip install "ray[tune]"
  pip install ray
  pip install numpy==1.23.3
  pip install kmeans-pytorch

  For the dataset, we will use their example/default "ml-100k" dataset, which consists of users' interactions with movies.
  Dataset Link MovieLens https://grouplens.org/datasets/movielens/.

  Some basic dataset stats: The number of users: 944
  Average interactions per users: 106.05
  The number of items: 1683
  Average interactions per item: 59.45
  The total number of interactions: 100,000

  Please run a basic example to ensure that things are installed correctly. You can create a file that includes the following two lines of code:

  from recbole.quick_start import run_recbole
  run_recbole(model="NeuMF", dataset="ml-100k")

  You should see a lot of output mentioning stats of the dataset, model, optimization procedure, etc. After this, the training process will start to output, including progress for each epoch, including training loss, validation performance, etc.

  Eventually, you will have a final result of NDCG on the testing data of 0.2725 after 42 epochs (due to using their default early stopping of 10 epochs).

- (2) [10 points] Default parameter investigation on model efficiency/utility trade-off. Here, you will run code similar to the above example, but rather than only using the model "NeuMF", you will also benchmark other models. Additionally, you will want to track the training time needed. This can be done programmatically in Python (e.g., import time and then use a start and end time to calculate the time it took to run the given block of code) or via command line if using a .py file (e.g., time python mycode.py).

  The methods to be compared are the following (and note that these are just requiring to change the "model" parameter in the "run_recbole()" function):

  - Pop - https://recbole.io/docs/user_guide/model/general/pop.html: this method just provides the recommendation for all users as the top popular items in the dataset
  - ItemKDD - https://recbole.io/docs/user_guide/model/general/itemknn.html: This method is the very basic item-based collaborative filtering we discussed in class.
  - NeuMF - https://recbole.io/docs/user_guide/model/general/neumf.html: This is the matrix factorization (MF) method we discussed in class, but then slightly adjusting how the embeddings are used along with an MLP beyond the basic MF using dot product of the user and item embedding (we also briefly commented on this NeurMF model after discussing MF in detail).
  - LightGCN - https://recbole.io/docs/user_guide/model/general/lightgcn.html: This is based on the Graph Convolutional Network model, but it removes the non-linearity, since in this setting we are not needing a neural network to transform learned features, but directly learning the user/item latent features and only leveraging the message passing mechanism of the GCN. In other words, if having a single-layer model, then users aggregate information from the items they interacted with, and items aggregate information from the users that interacted with them; if having two layers, then a user would consider not only the information of the items they interacted with but also the information/features of all the users who also interested with the same items that they did (and similarly for the two-layer items aggregation).
  - KTUP - https://recbole.io/docs/user_guide/model/knowledge/ktup.html: This method leverages additional information for context in making predictions and learning representations of the users and items based on a knowledge graph. Specifically, movies in this dataset are linked to a knowledge graph about the

movies, which provides connections between the movies known to us beforehand, such as if movies have links to the same director, genre, etc.

- SGL - https://recbole.io/docs/user_guide/model/general/sgl.html: This is a graph neural network (GNN) model that includes self-supervised learning (SSL). We will discuss SSL in GNNs later in the course, but at a high level, what they do here is use the neighborhood aggregation around a node (i.e., 1-hop neighbors of the items they interacted with, the 2-hop neighbors of the users who also interacted with the items that they did, etc.) and they get a resulting embedding via the message passing. Then, they make some augmentations to that graph by performing random operations such as dropping some of the edges/nodes in the graph, resulting in different graphs, so when again performing the message passing, the resulting embeddings of nodes will be different than when using the original graph. Ultimately, they seek to have a node's original embedding and the augmented version be similar, while the embedding between two different nodes should be dissimilar. They use this self-consistency loss in addition to traditional training loss on the user-item pairs.

Based on this, you will have 6 data points, one for each of the methods above. Please plot a basic matplotlib scatter plot to plot these 6 points where the x-axis represents the performance (in NDCG on the test, with the higher the better) and the y-axis represents the running time (in seconds for the total running time of that method based on your timer and not the timer that recbole uses associated with each epoch, with the lower the better).

- (3) [15 points] Hyperparameter Search on LightGCN. Here, we will do some basic hyperparameter analysis on LightGCN using the following code. Note that you can also explore their hyperparameter code. Still, as it is more complicated and has further installation/compatibility issues, we can just do this manually here with nested for loops over the parameter settings.

```
from recbole.quick_start import run_recbole
emb_sizes = [16, 256]
layers = [1,2,3]
for es in emb_sizes:
    for l in layers:
        parameter_dict = {
        'embedding_size': es,
        'n_layers': l,
        }
        run_recbole(model='LightGCN', dataset='ml-100k', config_dict=parameter_dict)
```

Figure 1: LightGCN model

Then, based on the output of these three runs, extract the NDCG performance on the validation set and report them in a nice readable/concise table. For example, something like the below with filling in the validation performance: emb size layers validation

- embedding size layer validation performance
- 16 1 ?
- 16 2 ?
- 16 3 ?
- 256 1 ?
- 256 2 ?
- 256 3 ?

3. [30 points] **Graph Classification for Molecule Classification Discovery** Will release this part after our class on this topic.

5. [100 points] **Project**.

For the course Project, you can choose two ways to finish it. The first one is you choose a given project from here [1]. The other one is to finish a project that is of your own interest or aligned with your own research.

Both of the above two ways fall into the same evaluation protocol where you will write a project report at the end of this semester, including:

- **Project Introduction and Background**
- **Research objective**
- **Research method**
- **Delivered experiment**
- **Research conclusion**

The report should ideally be at least 2 pages. However, shorter submissions may be accepted if the project outcomes are thoroughly and effectively conveyed within that length.

[1] https://ml-graph.github.io/fall-2024/project/