COL761 – HW3

Instructions

- 📅 Deadline: **April 26, 2025.**
- 🚫 Anti-Plagiarism Policy: Plag detection equals F-grade in the course.
- ✍️ Cite your sources in the report.
- 📊 Datasets: link.
- 🏆 This assignment is competitive.
- 🐍 Python 3.9

Submission related instructions:

- 💻 Add to GitHub.
- 1️⃣ Only one submission (per team).
- ➕ More in the document.

🧩 Graph Neural Networks in Action (ft. PyTorch)

This is a toy problem to prepare you for the next problem. However, just because this is a toy problem, it doesn't mean that this will be easy. We will give you a graph G and labels $\mathbf{y}$ for some nodes. Your task will be to first create a learnable function that will be modeled as a GNN, and then learn the function to make predictions.

Remark. Creating a learnable function means creating a function $f_{\boldsymbol{\theta}}$ parameterized by some tunable parameters $\boldsymbol{\theta}$. Learning the function then amounts to tuning these parameters based on some criteria. In our case, $f_{\boldsymbol{\theta}}$ is going to be a GNN. An equivalent (and more precise) way of saying "learning a function" is to say "learning the parameters of a parameterized function".

For this exercise, you must use `PyTorch v2.2.1` as the framework for learning the parameters of learnable functions. You will use `PyTorch Geometric v2.3.1` to build a GNN based learnable function.

Note. You will be using python 3.9 (no other version will be allowed) to run the experiments.

The quality of a learned function can be judged using the error between the predictions that it makes and the actual values. The metrics depend on our end goal and the dataset. Therefore, these metrics are defined later in another section. For now we will use the abstract terminology of "minimizing the error".

Another important thing to keep in mind is that the metric needs to be computed on actual predictions, and not predictions of the values used to learn since learning is done using optimization and optimization inherently minimizes the errors on the data that was used for optimization. But we want the learned function to *generalize* beyond the dataset used to train it (or tune its parameters). Therefore, the measure of goodness of the function then is the error between the predictions on an unseen subset of the full data and the actual values.

Remark. The above is general deep learning practice. In the context of our problem of node classification on a graph, the seen set will be a set of nodes, and the unseen set will also be a set of nodes. But I think you know that by now. What I should specify is that we keep the labels of the unseen set hidden from you, which means some entries of $\mathbf{y}$ are invalid (we represent this by `nan`).

With these, we can now define the problem to be solved.

Problem 1. Let $\{n\}$ represent the set $\{1, 2, \ldots, n\}$ (will be used as the set of n nodes numbered from 1 to n). Then, given

1. a graph $G = (\mathbf{X}, E)$, where $\mathbf{X} \in \mathbf{R}^{n \times d}$ represents d-dimensional features and the edge set $E \subseteq \{n\} \times \{n\}$, and
2. corresponding labels $\mathbf{y} \in \mathbf{R}^n$,

design a PyTorch based GNN and learn its parameters $\boldsymbol{\theta}$ such that the predictions on an unseen set give the minimum error.

Task 1: Design a GNN and learn its parameters to solve problem 1. [50 marks]

The Personality Quest

In E-commerce applications, an important task is to predict who is who, or the personality of users which can then be used as additional information when suggesting them products. The personality of users is a multi-dimensional thing (this is only the part of the personality we are interested in as part of our E-commerce application), a user A may be a fashion enthusiast and a budget shopper; while the user B might be a fashion enthusiast and an extravagant shopper. This may be represented as personality$(A) = \{$`fashion-enthusiast`, `budget-shopper`$\}$, and personality$(B) = \{$`fashion-enthusiast`, `extravagant-shopper`$\}$. These personalities will become the labels we want to predict.

To simplify things, we will do the following, we will represent the personality of each user by a vector $\mathbf{y} \in \mathbf{R}^\ell$ where ℓ represents the number of different "axes" of the user personality we might be interested in (e.g. fashion-enthusiast-ness, budget-shopper-or-not, loves-music-or-not, etc.). We will assign (once and then fix it across our experiments) an arbitrary order to these "axes", and arrange the label values in the vector. In our example, $\ell = 2$, and let's say that the first axis represents fashion-enthusiast-ness. Then,

$$\mathbf{y}_A := \text{personality}(A) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \text{and} \quad \mathbf{y}_B := \text{personality}(B) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Our E-commerce application will have users, products, and interactions between users and products. It will also have user features, product features. Finally, it will have edges between user nodes and product nodes representing interactions between users and products. With these, we can state the problem to be solved.

Problem 2. Let $\{k\}_{k=p}^{q}$ represent the set $\{p, p+1, \ldots, q\}$ (assuming $p \leq q$). Without loss of generality, assume that there are m users and $n-m$ products, and that the user nodes are numbered before product nodes. Therefore $U := \{k\}_{k=1}^{m}$ is the set of user nodes and $P := \{k\}_{k=m+1}^{n}$ is the set of product nodes. And the set of nodes is $V = U \cup P$. Then, given

1. a graph $G = (\mathbf{X}_U, \mathbf{X}_P, E)$, where $\mathbf{X}_U \in \mathbf{R}^{m \times d_1}$ represents d_1-dimensional features of the user nodes, $\mathbf{X}_P \in \mathbf{R}^{(n-m) \times d_2}$ represents d_2-dimensional features of the product nodes, and $E \subseteq U \times P$ is the set of edges between the user nodes and product nodes,
2. $\mathbf{Y} \in \mathbf{R}^{m \times \ell} := [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m]$ is a matrix made by stacking the user labels in order,

design a PyTorch based GNN and learn its parameters $\boldsymbol{\theta}$ such that the personality predictions on an unseen set gives the minimum error.

Task 2: Design a GNN and learn its parameters to solve problem 2. [50 marks]

📋⚙️ Submission Instructions

To ensure smooth evaluation of assignments, please ensure that you adhere to the following guidelines.

1. For task-1, create scripts `train1.sh` and `test1.sh`. Similarly for task-2 create `train2.sh` and `test2.sh`.

   ```
   bash train1.sh <path_to_train_graph> <output_model_file_path>
   bash test1.sh <path_to_test_graph> <path_to_model> <output_file_path>
   ```

   The interface for `train2.sh` and `test2.sh` is exactly the same. **The paths will be absolute.** For task-1 the output of `test1.sh` should be a csv file with one single column which is the prediction for each node. For task-2, the output should be a csv file which when read using `pandas` has the shape (m_t, ℓ) where ℓ has the usual meaning of the number of personality axes and m_t is the number of test nodes. More precisely, we will use the following code to load your csv file

   ```
   df = pd.read_csv('csv_file.csv', header=None)
   ```

2. Add `https://github.com/2023col761` as a collaborator to your GitHub repository. We'll count your latest git push as your submission time. (Do not create a new repository for this submission, use your repository from HW1.)
3. Upload your assignment under the `hw3` directory on *GitHub*.
4. The directory structure on *GitHub* should be as follows:

   ```
   hw3
   ├── src/
   ├── train1.sh
   ├── test1.sh
   ├── train2.sh
   └── test2.sh
   ```

5. On **Moodle**, submit a text file containing the link to your GitHub repository. It should be named `github_repo.txt`.
6. Do not submit data files.

⚖️ Evaluation metrics

Task	Dataset	Metric
Task-1	d1	ROC-AUC
Task-1	d2	Accuracy
Task-2	d1	Weighted F1

All these metrics are available in the scikit-learn library. For weighted F1, pass the parameter `average='weighted'`. For ROC-AUC, make sure to save class probabilities. For datasets with accuracy and F1, make sure to save actual class predictions.

Remark. In task-2, if there are m_t samples, then the predictions and ground truth both will be a $m_t \times \ell$ matrix, say $\widehat{\mathbf{Y}}$ and $\mathbf{Y}$ respectively. However, in scikit-learn's implementation of the F1-score, you can only use a 1-dimensional vector. Therefore, for computing F1-score, we will flatten $\widehat{\mathbf{Y}}$ to a vector $\widehat{\mathbf{y}} \in \mathbf{R}^\xi$ and $\mathbf{Y}$ to a vector $\mathbf{y} \in \mathbf{R}^\xi$ where $\xi = m_t\ell$. A (m_t, ℓ) shaped NumPy object can be flattened using

```
flattened_vector=two_D_vec.reshape(-1)
```

However, your output file should still be organized as (m_t, ℓ). We will do the flattening on our end before computing F1.

Competitive evaluation

The evaluation of both the tasks is competitive. However, their are some task-1 is only partially competitive.

Task-1:

1. 25 marks – Baseline performance: We have a few baseline numbers that you must hit. For `d1` the baseline number is 34%. For `d2` the baseline number is 64%. Your GNN must achieve performance greater than this, only then you will be awarded these 25 marks. Note that 12.5 marks for `d1` and 12.5 for `d2`.
2. 25 marks – Competitive component: For the competitive marking, we will follow a linear marking scheme (with round off). Let x be the maximum accuracy achieved by anyone doing the assignment. Let y be the accuracy achieved by you. Then, the marks allotted to you will be $\lfloor 25 \cdot 10y/x \rceil/10$. The notation $\lfloor a \rceil$ means rounding to the nearest integer. Therefore $\lfloor 25 \cdot 10a \rceil/10$ with $0 \leq a \leq 1$ will give a number between 0 and 25, rounded to the nearest tenth. For example, if $a := y/x = \sqrt{2}/2 \approx 0.70711$ then $25 \cdot 10a \approx 176.7775$ and $\lfloor 25 \cdot 10a \rceil/10 = 17.6$. Note that 12.5 marks for `d1` and 12.5 for `d2`, so in reality your scores will be computed using the equation mentioned before and then $\div 2$ for each dataset.

Task-2: This task is fully competitive. Let x be the max accuracy achieved. Let y be the accuracy achieved by your code. The marks allotted will be

$$\left\lfloor 50 \cdot \frac{10y}{x} \right\rceil \div 10$$

Time limit

‼️ Your code will be given **60 minutes** to run per dataset. Please periodically write the model your algorithm produces in the output file.