EN.553.744 Data Science Methods for Large-Scale Graphs

Homework 1

Luana Ruiz, Caio F. D. Netto, Ruijia Zhang

February 12, 2025

You must use Google Colab as your development environment—your `.ipynb` notebook will be your deliverable.

1 Introduction

The primary goal of this homework is to learn to implement machine learning models, specifically graph machine learning models, in PyTorch from scratch (check the quick PyTorch tutorial on Canvas if you have not already).

The setting is a community detection problem similar to the one we discussed several times in class. We will compare three approaches to community detection: (unsupervised) spectral clustering, spectral embeddings, and graph neural networks. By the end of the assignment, you will have a working understanding of each approach; grasp their advantages and limitations; and review important theoretical properties of graph convolutions (and so also GNNs). Enjoy!

2 Community detection, the stochastic block model and spectral clustering

Let $G = (\mathcal{V}, \mathcal{E})$ be an unweighted and undirected graph with node set $\mathcal{V} = \{1, \ldots, n\}$ and edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Community detection consists of assigning nodes $i \in \mathcal{V}$ to communities $c \in \{1, \ldots, C\}$. Specifically, we want to obtain the graph signal $y \in \{1, \ldots, C\}^n$ where $y_i = c$ implies node i belongs to community C.

2.1 The stochastic block model

The canonical model for graphs with communities is the stochastic block model (SBM). A SBM graph with n nodes and C communities has **symmetric** adja-

cency matrix $A \in \{0,1\}^{n\times n}$ given by

$$[A]_{ij} = [A]_{ji} \sim \text{Ber}([P]_{ij}), \quad P = YBY^\top \tag{1}$$

where $Y \in \{0,1\}^{n\times C}$ is the community assignment matrix $Y_{i\cdot} = \texttt{one-hot}(c)$, and $B \in [0,1]^{C\times C}$ is a full-rank matrix representing the block connection probability. In this assignment, we will consider a simplified model in which intra-community edges have probability p for all communities and inter-community edges probability q for all community pairs. Explicitly, B is given by

$$B = q\mathbf{1}\mathbf{1}^T - (q-p)I \tag{2}$$

where $\mathbf{1}$ is the all-ones vector and I the identity. We will also consider the SBM to be balanced, i.e., each community has n/C nodes.

Exercise 1 (Noisy SBM)**.** *Write a function that generates a noisy SBM graph given n, p and q and an edge deletion probability η (the noise parameter). I.e., your function should output the adjacency matrix (a `torch.tensor`) of an n-node random graph sampled from the SBM with intra-community probability p, inter-community probability q, and balanced communities, and where additionally each edge is dropped with probability η.*

Exercise 2. *Fix $\eta = 0.05$. Generate three 60-node graphs with the following parameters:*

- $C = 3, p = 0.8, q = 0.2$
- $C = 4, p = 0.8, q = 0.6$
- $C = 3, p = 0.55, q = 0.45$

and plot them using NetworkX.

2.2 Spectral clustering

Spectral methods for community detection are inspired by the spectral decomposition of the SBM. Consider for example the case in which $C = 2$, and B is given by

$$B = \begin{bmatrix} p & q \\ q & p \end{bmatrix}$$

with $p \neq q$. Further, the communities are balanced, i.e., n is even and both communities have size $n/2$. Relabeling $\mathcal{V}$ so that the first $n/2$ nodes belong to the first community and the remaining $n/2$ to the second, we see that the eigenvectors of $\mathbb{E}A \equiv P$, the expected adjacency, are given by

$$[v_1(\mathbb{E}A)]_i = \frac{1}{\sqrt{n}} \qquad [v_2(\mathbb{E}A)]_i = \begin{cases} -1/\sqrt{n}, & i \leq n/2 \\ +1/\sqrt{n}, & i > n/2. \end{cases} \tag{3}$$

For a graph G sampled from this model, given sufficiently large n and mild assumptions on p, q, we can expect the eigenvector $v_2(A)$ to provide a good estimate of the graph's community structure, i.e., $v_c(A) \approx v_c(\mathbb{E}A), c \in \{1, 2\}$.

More generally, in graphs with $C > 2$ balanced communities, the community information is "embedded" in the top C eigenvectors $v_1, v_2, \ldots, v_C$ ordered by descending eigenvalue magnitude $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_C|$. Explicitly, let

$$V_C = [v_1 \; v_2 \; \ldots \; v_{C-1} \; v_C] = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} \tag{4}$$

i.e., $V_C \in \mathbb{R}^{n \times C}$ is a matrix whose columns are the first C eigenvectors of A. Then, we can view the rows of V_C as the nodes' embeddings in C-dimensional space. Spectral clustering consists of applying clustering techniques to the nodes' embeddings in this space, the most common clustering technique being K-means clustering (in our case, $K = C$).

C-means clustering works as follows. At initialization, the cluster centers are vectors $\bar{u}_1, \ldots \bar{u}_C \in \mathbb{R}^C$ corresponding to embeddings $u_{i_1}, \ldots u_{i_C}$ of C nodes picked at random from $\mathcal{V}$. The algorithm then alternates between two steps:

1. Assign: for each node i, assign i to

$$y_i = \text{argmin}_c \|u_i - \bar{u}_c\|$$

2. Update: for each cluster c, recompute cluster center as the mean of

$$\mathcal{C}_i = \{u_i \text{ s.t. } y_i = c\}.$$

Steps 1 and 2 are repeated until the cluster assignments cease to change.

Exercise 3 (Assign)**.** *Write a function that implements the assign step of C-means clustering. I.e., your function should take in the cluster centers $\bar{u}_1, \ldots, \bar{u}_C$ and node embeddings $u_1, \ldots, u_n$, and output the assignments $y_1, \ldots, y_n$.*

Exercise 4 (Update)**.** *Write a function that implements the update step of C-means clustering. I.e., your function should take in the node embeddings $u_1, \ldots, u_n$ and the cluster assignments $y_1, \ldots, y_n$, and output the updated cluster centers $\bar{u}_1, \ldots, \bar{u}_C$.*

Exercise 5 (C-means clustering)**.** *Write a function that implements the full C-means clustering stack. I.e., your function should take in a graph adjacency matrix and the number of clusters C, and output the cluster assignments $y_1, \ldots, y_n$ and the cluster centers $\bar{u}_1, \ldots, \bar{u}_C$.*

Exercise 6. *Test this function on the graphs from Exercise 2 and compute the classification accuracy. Report the average accuracy, the standard deviation, and the best accuracy over 20 cluster center initializations. What do you observe?*

3 Semi-supervised learning with spectral embeddings

The spectral clustering method discussed in the previous section is fully unsupervised: no community labels are known beforehand, and all assignments are predicted using only graph structural information (the adjacency matrix). Yet, in some cases we might have knowledge of some of the nodes' labels, and these can be taken into account to improve predictions when labels are unknown.

3.1 Spectral embeddings

In situations where we have partial knowledge of community labels, an alternative is to use semi-supervision, i.e., to learn a classifier from partial label information.

Explicitly, given an n-node graph $G = (\mathcal{V}, \mathcal{E})$ and a true community assignment matrix $Y \in \{0,1\}^{n\times C}$, we fix a training set consisting of a subset $\mathcal{T} = \{i_1, \dots, i_m\} \subset \mathcal{V}$ of the graph nodes. This training set is used to define a node selection matrix $M_{\mathcal{T}} \in \{0,1\}^{m\times n}$ where $[M_{\mathcal{T}}]_{ij} = 1$ iff $i = m$ and $j = i_m$. We then use $\mathcal{T}$ to solve the following optimization problem

$$\min_f \ell(M_{\mathcal{T}} Y, M_{\mathcal{T}} f(A)) \tag{5}$$

where $\ell : \mathbb{R}^{m\times C} \times \mathbb{R}^{m\times C} \to \mathbb{R}$ is the cross-entropy loss and f is a function of A.

In semi-supervised spectral embedding methods, the function f is parametrized as

$$f(A) = \text{softmax}(V_C W) \tag{6}$$

where V_C is the eigenvector matrix in (4) and $W \in \mathbb{R}^{C\times C}$ are learnable parameters.

Exercise 7 (Spectral embeddings)**.** *Implement model f in (6) as a PyTorch model.*

Exercise 8. *Instantiate this model and train it on the graphs from Exercise 2 using 50% of the nodes as training samples. Compute the classification accuracy on the test set $\mathcal{V} \setminus \mathcal{T}$. Report the average accuracy, the standard deviation, and the best accuracy over 20 runs. What do you observe?*

When an n-node graph G has features $X \in \mathbb{R}^{n\times d}$ associated with its nodes, these can also be taken into account to inform the predictions made by model f. Before discussing how this is done in practice, we introduce the contextual stochastic block (CSBM) model, which can be thought of as an SBM augmented with node features.

3.2 The contextual SBM

A CBSM graph with n-nodes and C communities is defined as a tuple (G, X) where

- The adjacency matrix A is sampled as in (1);
- The node features $X \in \mathbb{R}^{n \times d}$ are sampled as

$$[X]_{i\cdot} \sim \mathcal{N}(\mu_c, \Sigma_c) \quad \text{for} \quad [Y]_{ic} = 1 \tag{7}$$

  where $\mu_c \in \mathbb{R}^d$ and $\Sigma_c \in \mathbb{R}^{d \times d}$ for $1 \leq c \leq C$.

Here again we consider the simplified model in which all communities have the same size, and intra-community edges have probability p for all communities and inter-community edges probability q for all community pairs. We further assume $\Sigma_c = \Sigma$ for all c.

Exercise 9 (Noisy CSBM)**.** *Write a function that generates a noisy CSBM graph-signal pair given n, p and q; an edge deletion probability η; and means $\mu_1, \ldots, \mu_C$ and covariance Σ. I.e., your function should output the adjacency matrix and node features (both in* `torch.tensor` *form).*

Exercise 10. *Fix $\eta = 0.05$, $C = 3$, $p = 0.55$, $q = 0.45$ and $\mu_1 = -1, \mu_2 = 0, \mu_3 = 1$. Generate two 60-node graphs varying the parameter Σ:*

(a) $\Sigma = 0.1$ *(b)* $\Sigma = 1$

(note that $d = 1$).

3.3 Feature-aware spectral embeddings

Given this additional node feature information, we can incorporate it into the spectral embeddings from (4) by concatenating to V_C the top κ eigenvectors of the feature correlation matrix XX^T. Explicitly, the function f in (6) becomes:

$$f(A, X) = \text{softmax}([V_C \mid V'_\kappa]W) \tag{8}$$

where $V'_\kappa \in \mathbb{R}^{n \times \kappa}$ are the top-κ eigenvectors of XX^T, $\mid$ denotes column-wise concatenation and $W \in \mathbb{R}^{C+\kappa \times C}$.

To avoid information "leakage" from the test labels to the training features, we modify problem (5) slightly and solve:

$$\min_f \ell(M_{\mathcal{T}} Y, M_{\mathcal{T}} f(A, X_{\mathcal{T}})) \tag{9}$$

where $X_{\mathcal{T}} \in \mathbb{R}^{n \times d}$ are the masked input features, satisfying $[X_{\mathcal{T}}]_{i\cdot} = [X]_i$ for $i \in \mathcal{T}$ and 0 otherwise.

Exercise 11 (Feature-aware spectral embeddings)**.** *Implement model f in (8) as a PyTorch model.*

Exercise 12. *Instantiate this model and train it on the graphs from Exercise 10 using $\kappa = C$ and 50% of the nodes as training samples. Compute the classification accuracy on the test set $\mathcal{V} \setminus \mathcal{T}$. Report the average accuracy, the standard deviation, and the best accuracy over 20 runs. What do you observe?*

Exercise 13. *Compute the Fourier transform $\hat{Y}'$ of $Y' = f(A, X)$. Plot $[\hat{Y}']_{i1}$ versus i, the eigenvalue indices ordered by decreasing value of $|\lambda_i|$. What do you observe?*

4 Graph neural networks for community detection

In the presence of node features, another method we can use to detect communities are graph neural networks (GNNs). In this case, $f(A, X)$ will be given by the output of L layers of the form:

$$X_l = \sigma_l\left(\sum_{k=0}^{K-1} A^k X_{l-1} H_{lk}\right) \tag{10}$$

with $X_0 = X$ and $f(A, X) = X_L$. Recall that σ_l are pointwise nonlinearities; in this particular problem, we have $\sigma_L = \text{softmax}$ and $\sigma_l = \text{ReLU}$ for $1 \leq l \leq L-1$. Given a parametrization $H_{lk} \in \mathbb{R}^{d_{l-1} \times d_l}$, the goal is to solve (9) for H_{lk}.

Exercise 14 (Graph convolutions)**.** *Implement the MIMO graph convolution in (10) as a PyTorch model. You **may not** use existing implementations of graph convolutions from other Python libraries for this part.*

Hint: At initialization, this class should take in the number of filter taps K, the input dimension d_{in}, and the output dimension d_{out}. The forward method should take in A and X, and output $U = \sum_{k=0}^{K-1} A^k X H_k$.

Exercise 15 (GNNs)**.** *Using the graph convolution class from the previous exercise, implement the full GNN stack (consisting of L layers like (10)) as a PyTorch model. You **may not** use existing implementations of graph convolutions from other Python libraries for this part.*

Hint: For a more versatile GNN model, you can feed the number of features per layer as a Python list $[d_0, d_1, \ldots, d_L]$.

Exercise 16. *Instantiate the GNN and train it on the graphs from Exercise 2 using 50% of the nodes as training samples. Compute the classification accuracy on the test set $\mathcal{V} \setminus \mathcal{T}$. Report the average accuracy, the standard deviation, and the best accuracy over 20 runs. What do you observe?*

Exercise 17. *Compute the Fourier transform $\hat{Y}'$ of $Y' = f(A, X)$. Plot $[\hat{Y}']_{i1}$ versus i, the eigenvalue indices ordered by decreasing value of $|\lambda_i|$. What do you observe, particularly compared with Exercise 13?*

Exercise 18. *Among spectral clustering, spectral embeddings, feature-aware spectral embeddings, and GNNs, which methods are local? Which are equivariant to node relabelings? Explain your answers.*

Exercise 19. *Based on your numerical results and on your answer to the previous exercise, compare the different community detection methods used in this assignment (you can, for instance, create a table listing their advantages and disadvantages).*