ADVANCED LEARNING FOR TEXT AND GRAPH DATA

Lab session 4: Machine Learning for Graphs

Lecture: Prof. Michalis Vazirgiannis
Lab: Michalis Chatzianastasis, Xiao Fei & Johannes Lutzeyer

Tuesday, November 12, 2024

We want to acknowledge significant contributions of Prof. Giannis Nikolentzos to the content of this lab.

This handout includes theoretical introductions, coding tasks and questions. Before the deadline, you should submit on moodle **or** here a **.zip** file named `Lab<x>_lastname_firstname.pdf` containing a `/code/` folder (itself containing your scripts with the gaps filled) and an answer sheet, following the template available here, and containing your answers to the questions. Your answers should be well constructed and well justified. They should not repeat the question or generalities in the handout. When relevant, you are welcome to include figures, equations and tables derived from your own computations, theoretical proofs or qualitative explanations. **One submission is required for each student. The deadline for this lab is November 26, 2024 11:59 PM**. No extension will be granted. Late policy is as follows: $]0, 24]$ hours late → -5 pts; $]24, 48]$ hours late → -10 pts; > 48 hours late → not graded (zero).

---

1 Setting Up the Environment

Please run the following commands in order in a terminal on your computer in a folder containing the provided "lab4-requirements.txt" file. This will allow you to work on a Python environment in which the code of this lab should run without issues and no further package installations shall be necessary.

```
conda create -n altegrad-lab4 Python=3.9
conda activate altegrad-lab4
pip install -r lab4-requirements.txt
```

Please note that we provide different requirements files for different cuda versions, which include the install of the required PyTorch Geometric. If the installation of Pytorch Geomertic (PyG) fails please follow the below steps in sequence.

1. Please install a compatible PyTorch 2.4.0 version from here: `https://pytorch.org/get-started/locally/`.
2. Please install a compatible PyG and optional dependencies from here: `https://pytorch-geometric.readthedocs.io/en/latest/install/installation.html`
3. Please pip install `networkx matplotlib numpy scipy scikit-learn nltk grakel jupyter`.

2 Learning objective

In this lab, you will implement some basic techniques for dealing with different graph mining problems. Specifically, the lab is divided into three parts. In the first part, you will study the dynamics of a real-world graph. Then, you will use some clustering algorithms to reveal its community structure. Finally, you will use graph kernels to measure the similarity between graphs and to perform graph classification. We will use Python, and the NetworkX 3.1 library (`http://networkx.github.io/`).

3 Analyzing a Real-World Graph

In this part of the lab, we will analyze the `CA-HepTh` collaboration network, examining several structural properties. The Arxiv HEP-TH (High Energy Physics - Theory) collaboration network comes from the e-print arXiv and covers scientific collaborations between authors of papers submitted to the High Energy Physics - Theory category. If an author i co-authored a paper with author j, the graph contains an undirected edge from i to j.

3.1 Load Graph and Simple Statistics

The graph is stored in the `CA-HepTh.txt` file[1], as an edge list:

```
# Directed graph (each unordered pair of nodes is saved once): CA-HepTh.txt
# Collaboration network of Arxiv High Energy Physics Theory category
# (there is an edge if authors coauthored at least one paper)
# Nodes: 9877 Edges: 51971
# FromNodeId    ToNodeId
24325   24394
24325   40517
24325   58507
24394   3737
24394   3905
24394   7237
...
```

Let's first create an undirected NetworkX graph based on the data contained in this file.

Task 1
Load the network data into an undirected graph G, using the `read_edgelist()` function of NetworkX. Note that, the delimeter used to separate values is the tab character $\backslash t$ and additionaly, that lines that start with the $\#$ character are comments. Furthermore, compute and print the following network characteristics: (1) number of nodes, (2) number of edges.

3.2 Connected Components

We will next compute the number of connected components of the graph, and extract the largest connected component. A connected component is defined as a subset of the nodes in the graph such that 1) any two nodes in the subset are connected to each other by a path and 2) there exists no path between a node in the subset and a node not in the subset.

[1]The graph can be downloaded from the following link: `https://snap.stanford.edu/data/ca-HepTh.txt.gz`.

Task 2
Print the number of connected components. If the graph is not connected, retrieve the largest connected component subgraph (also known as *giant connected component*) (Hint: you can use the `connected_components()` function of NetworkX). Find the number of nodes and edges of the largest connected component and examine to what fraction of the whole graph they correspond.

Question 1 (5 points)
What is the number of edges in graph G consisting of two connected components, where one of the connected components is a complete graph on 100 vertices and the other connected component is a complete bipartite graph with 50 vertices in each partition set? What is the number of triangles in G?

4 Community Detection

In the second part of the lab, we will focus on the community detection (or clustering) problem in graphs. Typically, a community corresponds to a set of nodes that highly interact among each other, compared to the intensity of interactions (as expressed by the number of edges) with the rest nodes of the graph. The experiments for this part will also be performed on the `CA-HepTh` collaboration network.

4.1 Spectral Clustering

We will first implement and apply a very popular graph clustering algorithm, called *Spectral Clustering* [5]. The basic idea of the algorithm is to utilize information encoded in the eigenvalues and eigenvectors of the normalised graph Laplacian (or another matrix associated with the graph) in order to identify well-separated clusters. Algorithm 1 illustrates the pseudocode of Spectral Clustering.

Algorithm 1 Spectral Clustering

```
Input: Graph $G = (V, E)$ and parameter $k$
Output: Clusters $\mathbf{C}_1, \mathbf{C}_2, \dots, \mathbf{C}_k$ (i.e., cluster assignments of each node of the graph)
1: Let $\mathbf{A}$ be the adjacency matrix of the graph
2: Compute the Laplacian matrix $\mathbf{L_{rw}} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$. Matrix $\mathbf{D}$ corresponds to the diagonal degree
   matrix of graph $G$ (i.e., degree of each node $v$ (= number of neighbors) in the main diagonal)
3: Apply eigenvalue decomposition to the Laplacian matrix $\mathbf{L_{rw}}$ and compute the eigenvectors that
   correspond to $d$ smallest eigenvalues. Let $\mathbf{U} = [\mathbf{u}_1|\mathbf{u}_2|\dots|\mathbf{u}_d] \in \mathbb{R}^{m \times d}$ be the matrix containing
   these eigenvectors as columns
4: For $i = 1, \dots, m$, let $y_i \in \mathbb{R}^d$ be the vector corresponding to the $i$-th row of $\mathbf{U}$. Apply $k$-means to the
   points $(y_i)_{i=1,\dots,m}$ (i.e., the rows of $\mathbf{U}$) and find clusters $\mathbf{C}_1, \mathbf{C}_2, \dots, \mathbf{C}_k$
```

We will now implement the Spectral Clustering algorithm.

Task 3
Fill in the body of the `spectral_clustering()` function which implements the Spectral Clustering algorithm. The algorithm must return a dictionary keyed by node to the cluster to which the node belongs (Hint: to perform k-means, you can use scikit-learn's implementation of the algorithm).

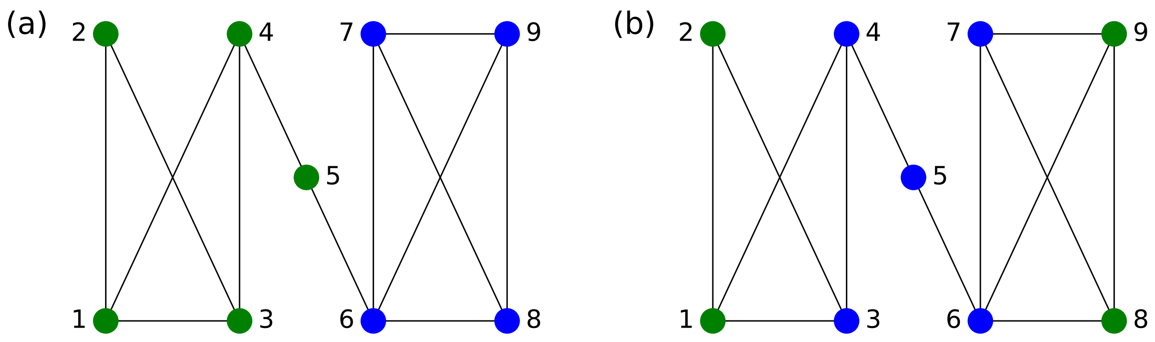

Figure 1: Two graphs where nodes have been assigned to 2 clusters. Cluster membership is indicated by node colour.

Task 4
Apply the Spectral Clustering algorithm to the giant connected component of the `CA-HepTh` dataset, trying to identify 50 clusters.

4.2 Modularity

To assess the quality of a clustering algorithm, several metrics have been proposed. *Modularity* is one of the most popular and widely used metrics to evaluate the quality of a network's partition into communities [3]. Considering a specific partition of the network into clusters, modularity measures the number of edges that lie within a cluster compared to the expected number of edges of a null graph (or configuration model), i.e., a random graph with the same degree distribution. In other words, the measure of modularity is built upon the idea that random graphs are not expected to present inherent community structure; thus, comparing the observed density of a subgraph with the expected density of the same subgraph in case where edges are placed randomly, leads to a community evaluation metric. Modularity is given by the following formula:

$$Q = \sum^{n_c} \left[\frac{l_c}{m} - \left(\frac{d_c}{2m} \right)^2 \right]$$

where, $m = |E|$ is the total number of edges in the graph, n_c is the number of communities in the graph, l_c is the number of edges within the community c and d_c is the sum of the degrees of the nodes that belong to community c. Modularity takes values in the range $[-1, 1]$, with higher values indicating better community structure.

Question 2 (5 points)
Compute (showing your calculations) the modularity of the clustering results shown in Figure 1. Note that different colors correspond to different clusters.

Task 5
Fill in the body of the `modularity()` function that computes the modularity of a clustering result.

Next, we will use modularity to evaluate two clustering results of the nodes of the giant connected component of the `CA-HepTh` dataset.

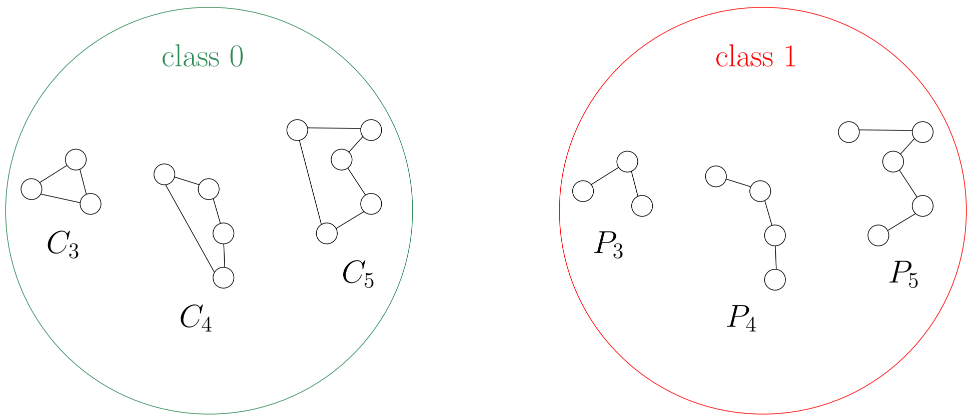

Figure 2: Dataset consisting of two sets of graphs: cycle graphs (left) and path graphs (right).

Task 6
Compute the modularity of the following two clustering results: (i) the one obtained by the Spectral Clustering algorithm using $k = 50$, and (ii) the one obtained if we randomly partition the nodes into 50 clusters (Hint: to assign each node to a cluster, use the `randint(a,b)` function which returns a random integer n such that $a \leq n \leq b$).

5 Graph Classification using Graph Kernels

In the last part of the lab, we will focus on the problem of graph classification. Graph classification arises in the context of a number of classical domains such as chemical data, biological data, and the web. In order to perform graph classification, we will employ graph kernels, a powerful framework for graph comparison.

Kernels can be intuitively understood as functions measuring the similarity of pairs of objects. More formally, for a function $k(x, x')$ to be a kernel, it has to be (1) symmetric: $k(x, x') = k(x', x)$, and (2) positive semi-definite. If a function satisfies the above two conditions on a set $\mathcal{X}$, it is known that there exists a map $\phi : \mathcal{X} \rightarrow \mathcal{H}$ into a Hilbert space $\mathcal{H}$, such that $k(x, x') = \langle \phi(x), \phi(x') \rangle$ for all $(x, x') \in \mathcal{X}^2$ where $\langle \cdot, \cdot \rangle$ is the inner product in $\mathcal{H}$. Kernel functions thus compute the inner product between examples that are mapped in a higher-dimensional feature space. However, they do not necessarily explicitly compute the feature map ϕ for each example. One advantage of kernel methods is that they can operate on very general types of data such as images and graphs. Kernels defined on graphs are known as *graph kernels*. Most graph kernels decompose graphs into their substructures and then to measure their similarity, they count the number of common substructures. Graph kernels typically focus on some structural aspect of graphs such as random walks, shortest paths, subtrees, cycles, and graphlets.

5.1 Loading Dataset

We will work with the MUTAG dataset [2] which is a collection of nitroaromatic molecules and the goal is to predict their mutagenicity on Salmonella typhimurium. Input graphs are used to represent chemical compounds, where vertices stand for atoms, while edges between vertices represent bonds between the corresponding atoms. We will load the dataset from pytorch geometric and then we will transform the the graphs to networkx using the function `to_networkx` from pytorch geometric.

Task 7
Fill in the body of the `load_dataset()` function to load the dataset as described above.

Before computing the kernels, it is necessary to split the dataset into a training and a test set. We can use the `train_test_split()` function of scikit-learn as follows:

```
from sklearn.model_selection import train_test_split

G_train, G_test, y_train, y_test = train_test_split(Gs, y, test_size=0.2)
```

5.2 Implementation of Graphlet Kernel

We will next investigate if graph kernels can distinguish cycle graphs from path graphs. We will use the following two graph kernels: (1) shortest path kernel, and (2) graphlet kernel. The shortest path kernel has already been implemented for you (i.e., `shortest_path_kernel()` function). The shortest path kernel counts the number of shortest paths of equal length in two graphs [1]. It can be shown that in the case of unlabeled graphs, the kernel maps the graphs into a feature space where each feature corresponds to a shortest path distance and the value is equal to the frequency of that distance in the graph (see Figure 3 for an illustration).

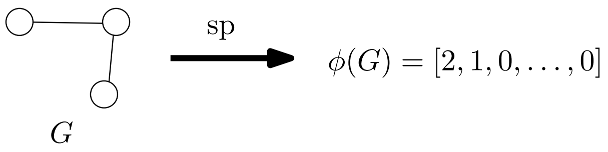

Figure 3: Example of feature map of the shortest path kernel. There are 2 shortest paths of distance 1 and 1 shortest path of distance 2 in this graph.

Question 3 (5 points)
Let P_n denote a path graph on n vertices and C_n denote a cycle graph on n vertices. Calculate the shortest path kernel for the pairs (C_4, C_4), (C_4, P_4) and (P_4, P_4).

The graphlet kernel decomposes graphs into graphlets (i.e., small subgraphs with k nodes where $k \in \{3, 4, 5\}$) and counts matching graphlets in the input graphs [4]. For example, the set of graphlets of size 3 is shown in Figure 4 below.

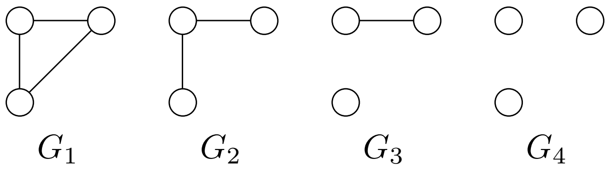

Figure 4: Set of the graphlets of size 3.

The graphlet kernel samples a number of small subgraphs from a graph, and computes their distribution. Here, we will focus on graphlets of size 3. Let $\{\text{graphlet}_1, \text{graphlet}_2, \text{graphlet}_3, \text{graphlet}_4\}$ be the set of size-3 graphlets (i.e., those shown in Figure 4). The graphlet kernel maps each graph G into a vector $f_G \in \mathbb{N}^4$ whose i-th entry is equal to the number of sampled subgraphs from G that are isomorphic to graphlet_i. Then, the graphlet kernel is defined as follows:

$$k(G, G') = f_G^\top f_{G'} \qquad (1)$$

Given a set of training graphs (with cardinality N_1), a set of test graphs (with cardinality N_2) and a graph kernel, we are interested in generating two matrices. A symmetric matrix $\mathbf{K}_{train} \in \mathbb{R}^{N_1 \times N_1}$ which contains the kernel values for all pairs of training graphs, and a second matrix $\mathbf{K}_{test} \in \mathbb{R}^{N_2 \times N_1}$ which stores the kernel values between the graphs of the test set and those of the training set. For the shortest path kernel, we can produce these two matrices as follows:

```
K_train_sp, K_test_sp = shortest_path_kernel(G_train, G_test)
```

We will next implement the graphlet kernel.

Task 8
Fill in the body of the `graphlet_kernel()` function. The function generates the feature maps of equation 1 by sampling `n_samples` size-3 graphlets from each graph. Then, it generates the $\mathbf{K}_{train}$ and $\mathbf{K}_{test}$ matrices by computing the inner products between the feature maps (Hint: you can use the `random.choice()` function of NumPy to sample 3 nodes from the set of nodes of a graph. Given a set of nodes `s`, use the `G.subgraph(s)` function of NetworkX to obtain the subgraph induced by set `s`. To test if a subgraph is isomorphic to a graphlet, use the `is_isomorphic()` function of NetworkX).

Task 9
Use the `graphlet_kernel()` function that you implemented to compute the kernel matrices associated with the graphlet kernel.

Question 4 (5 points)
Let k denote the graphlet kernel that decomposes graphs into graphlets of size 3. Let also G, G' denote two graphs and suppose that $k(G, G') = f_G^\top f_{G'} = 0$. What does a kernel value equal to 0 mean? Give an example of two graphs G, G' for which $k(G, G') = 0$ holds.

5.3 Graph Classification using SVM

After generating the $\mathbf{K}_{train}$ and $\mathbf{K}_{test}$ matrices, we can use the SVM classifier to perform graph classification. More specifically, as shown below, we can directly feed the kernel matrices to the classifier to perform training and make predictions:

```
from sklearn.svm import SVC

# Initialize SVM and train
clf = SVC(kernel='precomputed')
clf.fit(K_train, y_train)

# Predict
y_pred = clf.predict(K_test)
```

Task 10
Train two SVM classifiers (i.e., one using the kernel matrix generated by the shortest path kernel, and the other using the kernel matrix generated by the graphlet kernel). Then, use the two classifiers to make predictions. Evaluate the two kernels (i.e., shortest path and graphlet) by computing the classification accuracies of the corresponding models (Hint: use the `accuracy_score()` function of scikit-learn).

5.4 Graph Classification with GraKeL

In this section, we will use the GraKeL library, which offers a collection of pre-implemented graph kernels, to perform graph classification on a real-world dataset. GraKeL simplifies the process of applying various graph kernels, providing an easy-to-use framework for comparing and classifying graphs. GraKeL includes several well-known graph kernels such as the shortest path kernel, Weisfeiler-Lehman subtree kernel, random walk kernel, and more. Instead of manually implementing these kernels, we can leverage GraKeL's efficient implementations.

We will look into how graph kernels and graph classification methods can be applied to improve natural language processing tasks, specifically text categorization. Text categorization involves automatically assigning category labels to documents, which is widely used in applications like news classification and sentiment analysis in product reviews. We'll approach text categorization as a graph classification task and use graph kernels along with an SVM classifier to address it. The following code reads the data from disk, performs preprocessing steps (such as stemming), and extracts the vocabulary.

Task 11
Transform the documents into graphs and perform graph classification. Complete the function create_graphs_of_words() that transforms a list of documents into a list of graphs.

We will next utilize the graph_from_networkx() function of GraKeL to convert the NetworkX graphs to objects that can be handled by GraKeL.

Task 12
Create the training and test graphs using the graph_from_networkx() function. Initialize a WL subtree kernel and use it to construct the two kernel matrices (i.e., train and test).

Task 13
Train an SVM classifier and use it to make predictions.

In the final task, you have to experiment with different kernels from the GraKeL library and try to achieve better performance.

Task 14
Experiment with different kernels and evaluate the performance.

References

[1] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Proceedings of the 5th IEEE International Conference on Data Mining*, pages 74–81, 2005.

[2] Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991.

[3] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.

[4] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M Borgwardt. Efficient Graphlet Kernels for Large Graph Comparison. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 488–495, 2009.

[5] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.