INF581A: Advanced Deep Learning

Lab session: Graph Mining

Lecture: Prof. Michalis Vazirgiannis
Lab: Michail Chatzianastasis & Iakovos Evdaimon & Giannis Nikolentzos & Johannes Lutzeyer

January 14, 2025

1 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 3.8, and the NetworkX library (`http://networkx.github.io/`).

2 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.

2.1 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 i
# 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.

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

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.

2.2 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.

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
Let G be a cycle graph that consists of n nodes, i.e., a graph containing a single cycle through all n nodes. We randomly remove two edges from G (with uniform probability). What is the number of connected components of the emerging graph?

2.3 Analysis of the Degree Distribution of the Graph

Extract the degree sequence of the graph using the following code:

```
degree_sequence = [G.degree(node) for node in G.nodes()]
```

The above code returns a list that contains the degree of all the nodes of the graph.

Task 3
Find and print the minimum, maximum, median and mean degree of the nodes of the graph (Hint: you can use the built-in functions `min()`, `max()`, `mean()` of the NumPy library).

Let's now compute and plot the degree distribution of the graph.

Task 4
Plot the degree histogram using the `matplotlib` library of Python (Hint: use the `degree_histogram()` function that returns a list of the frequency of each degree value). Produce again the plot using log-log axis.

Question 2
Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic, $G_1 \cong G_2$, if there is a bijective mapping $f : V_1 \to V_2$ such that $(v_i, v_j) \in E_1$ iff $(f(v_1), f(v_2)) \in E_2$. Let two graphs have identical degree distributions. Does this imply that the two graphs are isomorphic to each other? If not, give a counterexample.

2.4 Clustering Coefficient

Next, we will compute the *global clustering coefficient*, also called the transitivity, of the graph, which is a measure of the degree to which nodes in a graph tend to cluster together, i.e., to create tightly knit groups characterized by a relatively high density of ties.

The global clustering coefficient is defined based on triplets of nodes. We call a triplet of nodes, which is connected by two edges, an *open triplet*. A triplet of nodes, which is connected by three edges is called a *closed triplet* (or triangle). Then, the global clustering coefficient is defined as the number of closed triplets divided by the number of both open and closed triplets.

Task 5
Calculate the global clustering coefficent of the HepTh graph. (Hint: Use the `transitivity` function implemented in NetworkX.)

Question 3
Let G be an undirected graph (without self-loops) consisting of n nodes and $\frac{n(n-1)}{2} - 1$ edges. What is the global clustering coefficient of that graph?

3 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.

3.1 Spectral Clustering

We will first implement and apply a very popular graph clustering algorithm, called *Spectral Clustering* [?]. 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 $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_k$ (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 $\mathbf{U} = [\mathbf{u}_1|\mathbf{u}_2|\ldots|\mathbf{u}_d] \in \mathbb{R}^{m \times d}$ be the matrix containing these eigenvectors as columns
4: For $i = 1, \ldots, m$, 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 $(y_i)_{i=1,\ldots,m}$ (i.e., the rows of $\mathbf{U}$) and find clusters $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_k$
```

We will now implement the Spectral Clustering algorithm.

Task 6
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).

Question 4
How does the mulitiplicity of the smallest eigenvalue of $\mathbf{L}_{\mathbf{rw}}$ relate to the graph structure represented in $\mathbf{L}_{\mathbf{rw}}$? And what structure is obeyed by the eigenvectors corresponding to the smallest eigenvalue of $\mathbf{L}_{\mathbf{rw}}$? Please feel free to cite any part of [**?**] in your answer.

Question 5
Given a graph G is the output of the spectral clustering deterministic or stochastic?

Task 7
Apply the Spectral Clustering algorithm to the giant connected component of the `CA-HepTh` dataset, trying to identify 50 clusters.

3.2 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 [**?**]. 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 6
Compute (showing your calculations) the modularity of the clustering results shown in Figure 1. Note that different colors correspond to different clusters.

Task 8
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.

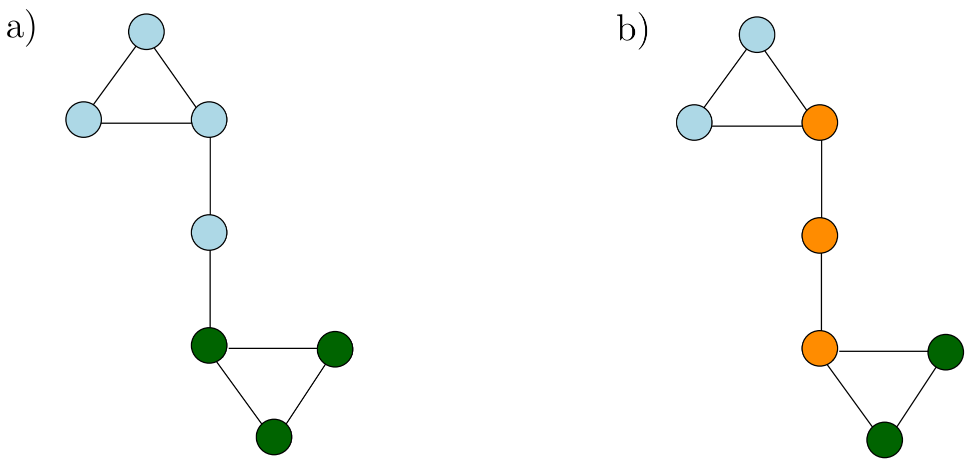

Figure 1: Two graphs where nodes have been assigned to 2 and 3 clusters, respectively. Cluster membership is indicated by node colour.

Task 9
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 $a \leq n \leq b$).

4 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.

4.1 Dataset Generation

We will first create a very simple graph classification dataset. The dataset will contain two types of graphs: (1) cycle graphs, and (2) path graphs. A cycle graph C_n is a graph on n nodes containing a single cycle through all nodes, while a path graph P_n is a tree with two nodes of degree 1, and all the remaining $n - 2$ nodes of degree 2. Each graph is assigned a class label: label 0 if it is a cycle or label 1 if it is a path. Figure 2 illustrates such a dataset consisting of three cycle graphs and three path graphs.

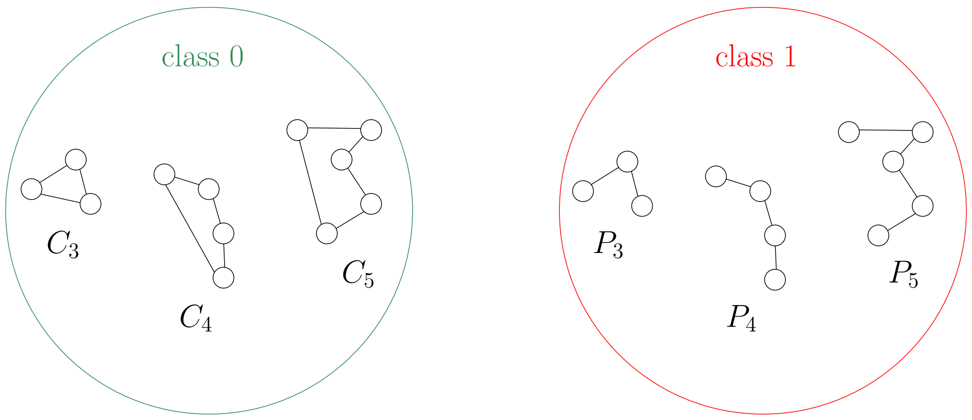

Figure 2: Dataset consisting of two sets of graphs: cycle graphs (left) and path graphs (right).

Use the `cycle_graph()` and `path_graph()` functions of NetworkX to generate 100 cycle graphs and 100 path graphs of size $n = 3, \ldots, 102$, respectively. Store the 200 graphs in a list and their class labels in another list.

Task 10
Fill in the body of the `create_dataset()` function to generate 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=0.1)
```

4.2 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. This 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 [?]. 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).

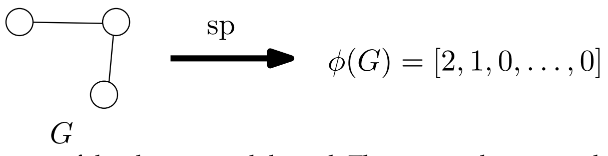

Figure 3: Example of feature map of the shortest path kernel. There are 2 shortest paths of distance 1 and 1 shortest path of distance 2 in this graph.

Question 7
Let P_n denote a path graph on n vertices and K_n denote a complete graph on n vertices. Calculate the shortest path kernel for the pairs (P_4, P_4), (P_4, K_4) and (K_4, K_4).

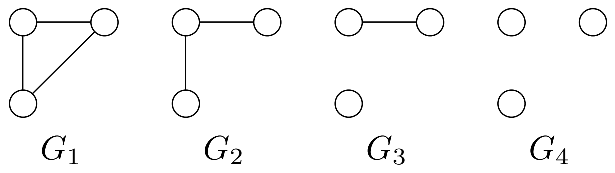
Figure 4: Set of the graphlets of size 3.

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 [?]. For example, the set of graphlets of size 3 is shown in Figure 4 below.
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:

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

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 11
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 12
Use the `graphlet_kernel()` function that you implemented to compute the kernel matrices associated with the graphlet kernel.

4.3 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 13
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).