Exercice5 : Forage de données de type Graphe

- Recherche de motifs fréquents
- Classification de graphes
- Segmentation (clustering)
- Compression de graphes

Utilise Networkx et scikit-learn. Tutoriel pour Networkx : https://networkx.org/documentation/stable/reference/index.html Exemple:

- `G = nx.erdos_renyi_graph(20, 0.2)` : instanie un graphe `G` aléatoire à 20 sommets, et 20% de probabilité de création d'arête)
- `combinations(G.nodes(), 4)` : instaure un liste des quadruplets de sommets de `G`)
- `sub = G.subgraph(nodes)` : instaure un motif de G contenant les sommets de l'ensemble `nodes`)
- `str(nx.to_numpy_array(G))` : instaure une forme canonique du graph `G` sous forme de `string` en utilisant les matrices d'adjacence)

In [1]:
```
import sys
!{sys.executable} -m pip install networkx
```
```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: networkx in /Users/oaoa2001/Library/Python/3.9/lib/python/site-packages (3.2.1)
WARNING: You are using pip version 21.2.4; however, version 24.0.1 is available.
You should consider upgrading via the '/Library/Developer/CommandLineTools/usr/bin/python3 -m pip install --upgrade pip' command.
```

In [2]:
```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
```

Example de graphes

In [3]:
```
G = nx.karate_club_graph()
nx.draw(G, with_labels=True)
plt.show()
```

In [4]:
```
print(str(nx.to_numpy_array(G)))
```
```
[[0. 4. 5. ... 2. 0. 0.]
 [4. 0. 6. ... 0. 0. 0.]
 [5. 6. 0. ... 0. 2. 0.]
 ...
 [2. 0. 0. ... 0. 4. 4.]
 [0. 0. 2. ... 4. 0. 5.]
 [0. 0. 0. ... 4. 5. 0.]]
```

In [5]:
```
#Generation d'une collection de graphes
graphs = []
labels = []
for i in range(25):
    graphs.append(nx.erdos_renyi_graph(10, 0.3))
    labels.append(1)
for i in range(25):
    graphs.append(nx.erdos_renyi_graph(10, 0.5))
    #graphs.append(nx.barabasi_albert_graph(20, 2))
    labels.append(2)
```

Recherche de motifs fréquents

1. Ecrire une fonction permettant de trouver tous les motifs fréquents de taille `k` (Example: min_support = 70%) d'une collection de graphes `graphs`

In [6]:
```
from itertools import combinations

def motifs_frequents(graphs, k, min_support):
    compte_motifs = {}
    nb_graphs = len(graphs)

    for G in graphs:
        motifs = set()
        sousgraphes = []
        for ens_noeuds in combinations(G.nodes(), k):
            sousgraphes.append(G.subgraph(ens_noeuds).copy())

        for H in sousgraphes:
            motif = str(nx.to_numpy_array(H).astype(int).tolist())
            ens_motifs.add(motif)

        for motif in ens_motifs:
            if(motif in compte_motifs.keys()):
                compte_motifs[motif] += 1
            else:
                compte_motifs[motif] = 1


    motifs_frequents = {}
    for motif, compte in compte_motifs.items():
        if compte / nb_graphs > min_support
    }

    return motifs_frequents
```

In [7]:
```
#Application à graphs

motifs = motifs_frequents(graphs, k=4, min_support=0.7)

print("Frequent motifs (>70%):")
for m, freq in motifs.items():
    print(m,freq)
```
```
Frequent motifs (>70%):
[[0, 0, 0, 1], [0, 0, 1, 0], [0, 1, 0, 1], [1, 0, 1, 0]] 0.8
[[0, 0, 0, 0], [0, 0, 1, 1], [0, 1, 0, 0], [0, 1, 0, 1]] 0.92
[[0, 0, 1, 1], [0, 0, 0, 0], [1, 0, 0, 1], [1, 0, 1, 0]] 0.72
[[0, 0, 0, 1], [0, 0, 0, 0], [0, 0, 0, 1], [1, 1, 1, 0]] 0.86
[[0, 1, 1, 0], [1, 0, 0, 1], [1, 0, 0, 0], [0, 1, 0, 0]] 0.82
[[0, 0, 0, 1], [0, 0, 0, 1], [0, 0, 0, 1], [1, 1, 1, 0]] 0.86
[[0, 1, 0, 0], [1, 0, 0, 0], [0, 0, 0, 1], [0, 0, 1, 0]] 0.88
[[0, 1, 0, 0], [1, 0, 0, 1], [0, 0, 0, 1], [0, 1, 1, 0]] 0.8
[[0, 0, 1, 1], [0, 0, 0, 1], [1, 0, 0, 0], [1, 1, 0, 0]] 0.94
[[0, 0, 0, 0], [0, 0, 1, 0], [0, 1, 0, 1], [0, 0, 1, 0]] 0.86
[[0, 0, 0, 1], [0, 0, 0, 0], [0, 0, 0, 1], [1, 1, 1, 0]] 0.86
[[0, 0, 0, 1], [0, 0, 0, 1], [0, 0, 0, 1], [1, 1, 1, 0]] 0.74
[[0, 0, 0, 0], [0, 0, 1, 1], [0, 1, 0, 0], [0, 1, 0, 1]] 0.8
[[0, 0, 0, 1], [0, 0, 0, 1], [0, 0, 0, 1], [1, 1, 1, 0]] 0.82
[[0, 1, 0, 0], [1, 0, 0, 1], [0, 0, 0, 1], [0, 1, 1, 0]] 0.8
[[0, 0, 0, 0], [0, 0, 1, 1], [0, 1, 0, 0], [0, 1, 0, 1]] 0.88
[[0, 1, 0, 1], [1, 0, 0, 0], [0, 0, 0, 1], [1, 0, 1, 0]] 0.86
[[0, 0, 0, 1], [0, 0, 0, 1], [0, 0, 0, 1], [1, 1, 1, 0]] 0.8
[[0, 0, 1, 1], [0, 0, 0, 1], [1, 0, 0, 0], [1, 1, 0, 0]] 0.9
[[0, 1, 0, 1], [1, 0, 0, 0], [0, 0, 0, 1], [1, 0, 1, 0]] 0.74
[[0, 0, 0, 0], [0, 0, 1, 1], [0, 1, 0, 0], [0, 1, 0, 1]] 0.74
[[0, 0, 0, 1], [0, 0, 0, 1], [0, 0, 0, 1], [1, 1, 1, 0]] 0.84
[[0, 0, 1, 1], [0, 0, 0, 0], [1, 0, 0, 0], [1, 1, 0, 0]] 0.84
[[0, 0, 1, 0], [0, 0, 0, 0], [1, 0, 0, 0], [0, 0, 0, 0]] 0.98
[[0, 1, 1, 0], [1, 0, 0, 0], [1, 0, 0, 0], [0, 0, 0, 0]] 0.9
[[0, 0, 1, 0], [0, 0, 1, 1], [1, 1, 0, 0], [0, 1, 0, 0]] 0.88
[[0, 0, 0, 1], [0, 0, 0, 1], [0, 0, 0, 1], [1, 1, 1, 0]] 0.86
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]] 0.94
[[0, 0, 1, 0], [0, 0, 0, 0], [1, 0, 0, 1], [0, 0, 1, 0]] 0.94
[[0, 0, 0, 1], [0, 0, 0, 1], [0, 0, 0, 1], [1, 1, 1, 0]] 0.84
[[0, 0, 1, 0], [0, 0, 0, 0], [1, 0, 0, 1], [0, 0, 1, 0]] 0.82
[[0, 0, 0, 1], [0, 0, 0, 1], [0, 0, 0, 1], [1, 1, 1, 0]] 0.78
```

Extraction de caractéristiques pour classification

2. Generer une matrice X1 dont les lignes correspondent aux graphes de `graphs` , et les colonnes au nombre de sommets et au nombre d'arêtes de chaque graphe.
3. Generer une matrice binaire X2 dont les lignes correspondent aux graphes de `graphs` , et les colonnes aux motifs fréquents de taille 4 (freq > 70%).

In [8]:
```
features = []
for i in range(50):
    G = graphs[i]
    features.append([G.number_of_nodes(), G.number_of_edges()])

X1 = np.array(features)
```

In [9]:
```
features = []
for i in range(50):
    G = graphs[i]
    features.append([])
    sousgraphes = []
    for ens_noeuds in combinations(G.nodes(), 4):
        sousgraphes.append(G.subgraph(ens_noeuds).copy())
    for H in motifs.keys():
        if H in sousgraphes:
            features[-1].append(1)
        else:
            features[-1].append(0)
X2 = np.array(features)
```

In [10]:
```
## Evaluation d'un modèle de classification simple (RdF)
from sklearn.ensemble import RandomForestClassifier
y = np.array(labels)
clf = RandomForestClassifier()
clf.fit(X1, y)
print("Accuracy X1:", clf.score(X1, y))
clf.fit(X2, y)
print("Accuracy X2:", clf.score(X2, y))
```
```
Accuracy X1: 0.9
Accuracy X2: 0.5
```

Clustering

4. Evaluer la qualité des clusterings (ARI) obtenus avec KMeans (k=2) pour les representations X1, X2

In [11]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
kmeans = KMeans(n_clusters=2, n_init=10)
kmeans.fit(X1)
print("Clusters X1:", kmeans.labels_)
print("ARI X1:", adjusted_rand_score(y, kmeans.labels_))
kmeans.fit(X2)
print("Clusters X2:", kmeans.labels_)
print("ARI X2:", adjusted_rand_score(y, kmeans.labels_))
```
```
Clusters X1: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
ARI X1: 0.5696290504702858
Clusters X2: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
ARI X2: 0.0
```
```
/Users/oaoa2001/Library/Python/3.9/lib/python/site-packages/sklearn/base.py:1473: ConvergenceWarning: Number of distinct clusters (1) found smaller than n_clusters (2). Possibly due to duplicate points in X.
  return fit_method(estimator, *args, **kwargs)
```

Compression de graphe

On vous donne deux fonctions:

- find_motif_occurrences(G, motif) : retourne toutes les occurences d'un `motif` dans un graph `G`
- compress_graph(G, motif, non_motif) : remplace toutes les occurences d'un `motif` par un seul noeud

In [13]:
```
def find_motif_occurrences(G, motif):
    matcher = nx.algorithms.isomorphism.GraphMatcher(G, motif)
    return (mapping for mapping in matcher.subgraph_isomorphisms_iter())

def compress_graph(G, motif, non_motif):
    G = G.copy()
    occurrences = find_motif_occurrences(G, motif)

    used_nodes = set()
    new_node_id = max(G.nodes()) + 1 if G.nodes() else 0

    for occ in occurrences:
        nodes = set(occ.keys())

        if nodes & used_nodes:
            continue

        used_nodes.update(nodes)

        # Créer un super-noeud
        super_node = new_node_id
        new_node_id += 1
        G.add_node(super_node, label=non_motif)

        # Reconnecter les voisins
        neighbors = set()
        for n in nodes:
            neighbors.update(G.neighbors(n))

        neighbors -= nodes  # retirer les arêtes internes

        for neigh in neighbors:
            G.add_edge(super_node, neigh)

        # Retirer les noeuds originaux
        G.remove_nodes_from(nodes)

    return G
```

5. Generer une nouvelle collection de 50 graphes à partir de `graphs` où tous les motifs de type `triangle` dans les graphes de `graphs` sont remplacés par un noeud.

In [14]:
```
motif = nx.Graph()
motif.add_edges_from([(0,1), (1,2), (2,0)])  # triangle

graphs_compresses = []
for i in range(50):
    graphs_compresses.append(compress_graph(graphs[i], motif, non_motif="triangle"))
    print("Original nodes:", graphs[i].number_of_nodes())
    print("Compressed nodes:", graphs_compresses[i].number_of_nodes())
```
```
Original nodes: 10
Compressed nodes: 6
Original nodes: 10
Compressed nodes: 8
Original nodes: 10
Compressed nodes: 6
Original nodes: 10
Compressed nodes: 6
Original nodes: 10
Compressed nodes: 7
Original nodes: 10
Compressed nodes: 8
Original nodes: 10
Compressed nodes: 6
Original nodes: 10
Compressed nodes: 8
Original nodes: 10
Compressed nodes: 6
Original nodes: 10
Compressed nodes: 8
Original nodes: 10
Compressed nodes: 6
Original nodes: 10
Compressed nodes: 8
Original nodes: 10
Compressed nodes: 6
Original nodes: 10
Compressed nodes: 8
Original nodes: 10
Compressed nodes: 6
Original nodes: 10
Compressed nodes: 8
Original nodes: 10
Compressed nodes: 6
Original nodes: 10
Compressed nodes: 6
Original nodes: 10
Compressed nodes: 6
```