République Tunisienne
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique

Université de Carthage

Faculté des Sciences Economiques
et de Gestion de Nabeul

Etudiante en :
Master Business Computing

Spécialité :
Business Computing
N° d'ordre :

MRBC-2024-05-F-E

Analyse des Réseaux Sociaux

présenté à

Faculté des Sciences Economiques et de Gestion de Nabeul

en vue de l'obtention du

Diplôme National de Master de Recherche en : Business Computing

Spécialité : Business Computing

Réalisé par :

Ferchichi Eya

Astro Physics collaboration network

Mme. Fériel BEN FRAJ

Table des matières

i. Introduction

L'analyse des réseaux sociaux est une discipline pluridisciplinaire qui se penche sur les interactions entre différentes entités sociales telles que les individus, les groupes, les organisations et même les concepts.

Dans notre cas, nous nous concentrons sur les réseaux de collaborations scientifiques, où les chercheurs sont les entités et les liens entre eux représentent leurs collaborations dans le domaine de l'astrophysique.

Ces réseaux offrent un aperçu unique de la manière dont la connaissance scientifique est générée, partagée et propagée.

Ces réseaux sont souvent représentés sous forme de graphes, où les nœuds représentent les entités et les arêtes leurs relations.

En analysant la structure de ces réseaux, nous pouvons découvrir des informations cruciales sur les dynamiques de collaboration, l'influence des acteurs clés, les communautés de recherche et les tendances émergentes.

Nous nous penchons notamment sur la centralité des acteurs, qui mesure leur importance dans le réseau en fonction de leur position et de leur rôle dans les interactions.

Cette mesure nous aide à identifier les chercheurs centraux, les principaux contributeurs à la production et à la diffusion des idées scientifiques.

Nous examinons également les communautés de recherche, qui regroupent des entités fortement interconnectées au sein du réseau.

Identifier ces communautés nous permet de cartographier les différents domaines de recherche, de détecter les collaborations interdisciplinaires et de comprendre comment les connaissances sont organisées et partagées dans le domaine de l'astrophysique.

ii. Collecte des données

1. Le source de données en ligne

Le réseau de collaboration Arxiv ASTRO-PH (Astrophysique) provient des e-prints arXiv et couvre les collaborations scientifiques entre les auteurs des articles soumis à la catégorie Astrophysique. Si un auteur i a co-écrit un article avec l'auteur j, le graphe contient une arête non orientée de i à j. Si l'article est co-écrit par k auteurs, cela génère un sous-graphe (complètement) connecté sur k nœuds.

Les données couvrent les articles de la période de janvier 1993 à avril 2003 (124 mois). Elles commencent quelques mois après la création de l'arXiv et représentent donc essentiellement l'histoire complète de sa section ASTRO-PH.

2. Les entités(nœuds) et les relations entre elles(liens)

- Entités (nœuds) : Les nœuds représentent les articles de la section ASTRO-PH de ArXiv.
- Les liens représentent les citations entre les articles. Si un article A cite un article B, il y aura un lien dirigé de A vers B.

3. Les informations additionnelles valables

a. Attributs des nœuds :

- Auteurs : Chaque nœud représente un auteur, donc vous pouvez inclure le nom de l'auteur.
- Institutions : Si les informations sont disponibles, vous pouvez également inclure l'institution à laquelle l'auteur est affilié.
- Nombre de publications : Le nombre total de publications de chaque auteur peut être un attribut intéressant.

b. Poids des liens :

- Le poids des liens pourrait représenter le nombre de collaborations entre deux auteurs. Par exemple, si deux auteurs ont co-écrit plusieurs articles ensemble, le poids du lien entre eux pourrait être le nombre total de ces articles.

4. Obtenir les données à partir de la source de données sélectionnée

J'ai téléchargé les données à partir du lien fourni sur le site Web SNAP.

5. Construire un réseau à partir des données

J'ai travaillé sur ce projet en utilisant le langage Python dans Google Colab, avec le jeu de données "Paper citation network of Arxiv ASTRO-PH category", qui offre une vue d'ensemble des citations entre les articles publiés dans la catégorie Astrophysique sur arXiv.org, ce qui en fait une ressource précieuse pour les chercheurs dans ce domaine.

iii. Analyse du réseau

1. L'analyse de la distribution des degrés

L'analyse de la distribution des degrés examine comment les connexions sont réparties entre les nœuds d'un réseau. Elle permet de comprendre la structure globale du réseau en étudiant la fréquence des degrés des nœuds. Cette analyse révèle des informations importantes sur la connectivité, la centralité et la robustesse du réseau.

- Importer les bibliothèques qui nous avons besoin pour cette Analyse

```
import gzip
import networkx as nx
import matplotlib.pyplot as plt
```

- Décompresser le fichier et charger les données

```
# Decompress the file and load the data
with gzip.open("ca-AstroPh.txt.gz", "rt") as f:
    data = f.readlines()
```

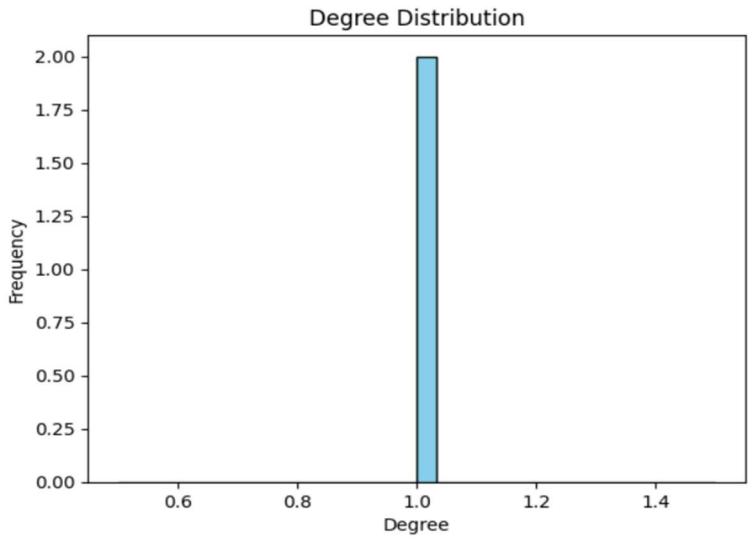
- Créer le Graphe

```
# Create a graph
G = nx.Graph()

# Add edges from the data
for line in data:
    # Skip lines that start with #
    if line.startswith("#"):
        continue

    # Parse the edge
    edge = line.strip().split()
    author1 = edge[0]
    author2 = edge[1]

    # Add edge to the graph
G.add_edge(author1, author2)
```

- L'analyse de la distribution des degrés

```
# Analysis of degree distribution
degree_sequence = sorted([d for n, d in G.degree()], reverse=True)
plt.hist(degree_sequence, bins=30, color='skyblue', edgecolor='black')
plt.title("Degree Distribution")
plt.xlabel("Degree")
plt.ylabel("Frequency")
plt.show()
```

- Le résultat



2. L'analyse des composants connectés

L'analyse des composants connectés examine les parties du réseau où tous les nœuds sont interconnectés. Cela permet de détecter les ensembles de nœuds fortement liés ainsi que les nœuds isolés.

- Le code sur mon DataSet

```
# Analysis of connected components
connected_components = nx.connected_components(G)
num_connected_components = nx.number_connected_components(G)
largest_connected_component = max(connected_components, key=len)
print("Number of connected components:", num_connected_components)
print("Size of largest connected component:", len(largest_connected_component))
```

- Le résultat

```
Number of connected components: 1
Size of largest connected component: 2
```

3. L'analyse des chemins

- Le code sur mon DataSet

```
# Analysis of shortest paths
avg_shortest_path_length = nx.average_shortest_path_length(G)
print("Average shortest path length:", avg_shortest_path_length)
```

- Le résultat

```
Average shortest path length: 1.0
```

4. Le coefficient de clustering et l'analyse de la densité

L'analyse des chemins consiste à étudier les itinéraires possibles entre les nœuds d'un réseau. Cela permet de comprendre la connectivité, la distance et l'efficacité des communications dans le réseau.

- Le code sur mon DataSet

```
# Clustering coefficient
clustering_coefficient = nx.average_clustering(G)
density = nx.density(G)
print("Average clustering coefficient:", clustering_coefficient)
print("Density of the graph:", density)
```

- Le résultat

```
Average clustering coefficient: 0.0
Density of the graph: 1.0
```

5. L'analyse de la centralité

L'analyse de la centralité consiste à évaluer l'importance des nœuds dans un réseau en fonction de leur position et de leurs connexions. Cela permet de déterminer quels sont les nœuds les plus influents, les plus centraux ou les plus stratégiques dans le réseau.

- Le code sur mon DataSet

```
# Analysis of centrality
centrality = nx.degree_centrality(G)
sorted_centrality = sorted(centrality.items(), key=lambda x: x[1], reverse=True)
print("Top 10 nodes by degree centrality:")
for node, centrality in sorted_centrality[:10]:
    print(node, centrality)
```

- Le résultat

```
Top 10 nodes by degree centrality:
50641 1.0
57507 1.0
```

iv. Identification des communautés

1. L'algorithme K-clique

L'algorithme K-clique utilise une formule basée sur la recherche exhaustive pour trouver tous les ensembles de nœuds qui forment des sous-graphes complets de taille K. Voici une formulation simple de l'algorithme :

Pour chaque nœud u dans le graphe :

- Créer un ensemble contenant u

- Pour chaque voisin v de u :

- Étendre l'ensemble en ajoutant v s'il est connecté à tous les nœuds de l'ensemble

- Répéter le processus récursivement jusqu'à ce que l'ensemble ne puisse plus être étendu

Une fois que tous les ensembles de K-clique sont trouvés, l'algorithme les retourne comme résultats.

- Le code sur mon DataSet

```
# K-clique
k_clique_communities = list(nx.algorithms.community.k_clique_communities(G, k=3))
print("Nombre de communautés selon K-clique:", len(k_clique_communities))
print("Communautés selon K-clique:")
for i, community in enumerate(k_clique_communities):
    print(f"Communauté {i+1}: {community}")
```

- Le résultat

```
Nombre de communautés selon K-clique: 1353
Communautés selon K-clique:
Communauté 1: frozenset({'109820', '17027', '45653', '75007', '53891'})
Communauté 2: frozenset({'62515', '56559', '43052'})
Communauté 3: frozenset({'86901', '76438', '76028', '54017', '125853', '1582', '2
Communauté 4: frozenset({'1119', '51100', '22432'})
Communauté 5: frozenset({'129847', '57178', '104299', '128033', '8548'})
Communauté 6: frozenset({'33099', '90071', '119600'})
Communauté 7: frozenset({'89169', '77822', '35695', '58204'})
Communauté 8: frozenset({'114473', '9335', '11275', '26205', '82843', '11553', '8
Communauté 9: frozenset({'46858', '90083', '67925'})
Communauté 10: frozenset({'131302', '46985', '10641', '27746'})
Communauté 11: frozenset({'55306', '123271', '122980', '120522', '27634', '55325'
Communauté 12: frozenset({'110370', '33135', '90100', '123431'})
Communauté 13: frozenset({'9572', '125561', '76855', '120254', '81964', '4445', '
Communauté 14: frozenset({'69719', '69718', '65798', '80196'})
Communauté 15: frozenset({'115084', '129130', '98712', '72650'})
Communauté 16: frozenset({'115084', '59362', '47892', '130545'})
Communauté 17: frozenset({'100238', '24676', '70029', '111562'})
Communauté 18: frozenset({'7783', '116173', '52171', '128776', '18505'})
Communauté 19: frozenset({'72464', '17885', '1267', '85177'})
Communauté 20: frozenset({'58532', '37818', '56224'})
Communauté 21: frozenset({'58532', '28827', '27573', '71157', '9648'})
Communauté 22: frozenset({'58532', '122165', '119665'})
Communauté 23: frozenset({'58532', '54011', '130188'})
```

2. L'algorithme Propagation des labels

L'algorithme Louvain est une méthode de détection de communautés dans les réseaux complexes. Il regroupe les nœuds du réseau pour maximiser la modularité, qui mesure la qualité des partitions communautaires. Les nœuds sont déplacés entre les communautés jusqu'à ce que la modularité ne puisse plus être améliorée, puis les communautés sont fusionnées pour optimiser davantage la structure. Enfin, l'algorithme retourne les communautés détectées comme résultat, révélant ainsi la structure sous-jacente du réseau.

- Le code sur mon DataSet

```
# Propagation des labels
label_propagation_communities = list(nx.algorithms.community.
label_propagation.label_propagation_communities(G))

# Affichage des résultats
print("\nNombre de communautés selon la propagation des labels:",
      len(label_propagation_communities))
print("Communautés selon la propagation des labels:")
for i, community in enumerate(label_propagation_communities):
    print(f"Communauté {i+1}: {community}")
```

- Le résultat

```
Nombre de communautés selon la propagation des labels: 1040
Communautés selon la propagation des labels:
Communauté 1: {'26325', '50641', '49676'}
Communauté 2: {'86901', '86654', '76438', '76028', '1582', '132927', '2641', '344
Communauté 3: {'560', '122815', '114018', '95', '59628', '80954'}
Communauté 4: {'130865', '110570', '79956', '52781', '8816', '109036', '110559',
Communauté 5: {'58532', '17737', '16793', '37818', '87601', '28827', '9648', '562
Communauté 6: {'80811', '74919', '3193', '62004', '50746', '37531', '87237', '992
Communauté 7: {'16897', '104154', '116851', '123201', '103014', '12890', '80526',
Communauté 8: {'115745', '35946', '94747', '25599', '115965', '107716', '88381',
Communauté 9: {'25346', '32357', '97937', '51506', '118286', '43742', '2969'}
Communauté 10: {'3547', '56669', '18252', '461', '66826', '129683', '85199', '162
Communauté 11: {'114623', '106954', '73499', '132803', '40443'}
Communauté 12: {'105016', '10429', '133266', '2758', '119960', '45746', '535', '9
Communauté 13: {'103465', '103476', '27749', '3068', '53786', '130391', '54017',
Communauté 14: {'35947', '76035', '60580', '80120', '48363', '84477', '30463', '3
Communauté 15: {'62844', '42578', '111014', '57459', '43161', '71302', '109229',
Communauté 16: {'109340', '131212', '87599', '48256', '86334', '48546', '117071',
Communauté 17: {'85941', '111916', '61859', '63840'}
Communauté 18: {'117311', '85993', '72171'}
Communauté 19: {'130921', '84025', '54455', '109217', '3584', '109371', '82158'}
Communauté 20: {'80394', '90023', '80396', '95690'}
```

3. L'algorithme Louvain

L'algorithme Louvain est une méthode de détection de communautés dans les réseaux. Il itère pour regrouper les nœuds en communautés qui maximisent la modularité, mesurant la qualité des partitions. Les nœuds sont déplacés entre les communautés jusqu'à ce que la modularité ne puisse plus être améliorée, puis les communautés sont fusionnées pour optimiser davantage la structure.

- Le code sur mon DataSet

```
import gzip
import networkx as nx
from networkx.algorithms.community import k_clique_communities,
label_propagation_communities, greedy_modularity_communities
louvain = list(nx.algorithms.community.greedy_modularity_communities(G))
print("\nNombre de communautés selon Louvain:", len(louvain))
print("Communautés selon Louvain:", louvain)
```

- Le résultat

```
Nombre de communautés selon Louvain: 455
Communautés selon Louvain: [frozenset({'86654', '130519', '47569', '56559', '76597
```

4. L'algorithme Infomap

L'algorithme Infomap est une méthode de détection de communautés qui cherche à minimiser la longueur de la description du flux d'information dans un réseau. Il attribue à chaque nœud une communauté distincte, puis optimise la distribution des nœuds entre les communautés pour minimiser la longueur de la description du flux. Enfin, il retourne les communautés détectées comme résultat, révélant ainsi la structure modulaire du réseau.

- Le code sur mon DataSet

```
# Détection des communautés avec l'algorithme Infomap
infomap = ig_graph.community_infomap()
# Affichage des résultats
print("Nombre de communautés selon Infomap:", len(infomap))
print("Communautés selon Infomap:", infomap)
```

- Le résultat

```
Nombre de communautés selon Infomap: 1186
Communautés selon Infomap: Clustering with 18772 elements and 1186 clusters
[   0] 84424, 276, 5089, 20113, 47005, 51730, 53681, 58458, 64124, 66200,
       97101, 125190, 1808, 7919, 43892, 61009, 118452, 125189, 108856, 15036,
       53607, 277, 2919, 80421, 26347, 60560, 86002, 94260
[   1] 1662, 33040, 61290, 127302, 3393, 24798, 30664, 32559, 53609, 54856,
       56381, 92134, 104248, 122290, 128492, 131263, 24583, 52398, 100108,
       5710, 16959, 39244, 51090, 75978, 82296, 86964, 102764, 109332, 107812,
       84333, 78432, 32308, 87295, 120558, 38533, 117386, 25672, 103116,
       66741, 12690, 40099, 46743, 33855
[   2] 6058, 85420, 101811, 121399, 48189, 3634, 18600, 5638, 82587, 18642,
       71466, 44866, 17689, 110679, 107194, 3420, 75314, 11529, 14824, 16287,
```

5. L'algorithme Demon

L'algorithme Demon est une méthode de détection de communautés basée sur la théorie des percolations. Il identifie les "démons", des sous-structures denses fortement liées entre elles mais faiblement connectées au reste du réseau. En sélectionnant et en étendant ces démons, l'algorithme attribue à chaque nœud une communauté, révélant ainsi les sous-groupes distincts dans le réseau.

- Le code sur mon DataSet

```python
# Utilisation de l'algorithme Demon pour détecter les communautés
demon_communities = demon_algorithm(G)

# Affichage des résultats
print("\nNombre de communautés selon Demon:", len(demon_communities))
print("Communautés selon Demon:")
for i, community in enumerate(demon_communities):
    print(f"Communauté {i+1}: {community}")
```

- Le résultat

```
Nombre de communautés selon Demon: 290
Communautés selon Demon:
Communauté 1: {'65517', '31345', '36020', '131012', '10397', '1134', '52041', '125664', '100773', '132762', '37949',
Communauté 2: {'23469', '7969'}
Communauté 3: {'59196', '109321', '87608'}
Communauté 4: {'85273', '49271', '60963', '124702'}
Communauté 5: {'96531', '73298', '58141', '9774', '72796', '131978', '15033', '7439'}
Communauté 6: {'50922', '21008'}
Communauté 7: {'30701', '99924'}
Communauté 8: {'76465', '33218', '33226', '18471'}
Communauté 9: {'14784', '11480', '86418'}
Communauté 10: {'28525', '14403', '11572', '108766', '114809'}
Communauté 11: {'62715', '70753', '60341'}
Communauté 12: {'4503', '97931', '95181'}
Communauté 13: {'105079', '130203'}
Communauté 14: {'131652', '48649'}
Communauté 15: {'69980', '64652', '111601', '120157', '2685'}
Communauté 16: {'15244', '28029', '32474', '82205', '106301'}
Communauté 17: {'18999', '27011'}
Communauté 18: {'96159', '885', '126747', '70802', '25354', '110571', '67929'}
Communauté 19: {'11153', '50098'}
```

v. Prédiction des liens

- Import des bibliothèques nécessaires

```
import networkx as nx
import gzip
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

- Chargement des données

```
with gzip.open('ca-AstroPh.txt.gz', 'rb') as f:
    data = f.readlines()
```

- Exemple des paires de nœud et de labels

```
[ ] node_pairs = []
    labels = []

    # Charger les données depuis le fichier compressé
    with gzip.open('ca-AstroPh.txt.gz', 'rb') as f:
        # Lire les données ligne par ligne
        for line in f:
            # Convertir la ligne en string
            line = line.decode('utf-8').strip()
            # Ignorer les commentaires et les lignes vides
            if line.startswith('#') or len(line) == 0:
                continue
            # Séparer les nœuds source et destination
            source, target = line.split('\t')
            # Ajouter la paire de nœuds à la liste des paires
            node_pairs.append((source, target))
            # Ajouter le label (1 pour indiquer la présence d'un lien)
            labels.append(1)

    # Afficher un exemple des paires de nœuds et des labels
    print("Exemple de paires de nœuds et de labels :")
    for pair, label in zip(node_pairs[:5], labels[:5]):
        print(pair, label)
```

- Le résultat

```
Exemple de paires de nœuds et de labels :
('84424', '276') 1
('84424', '1662') 1
('84424', '5089') 1
('84424', '6058') 1
('84424', '6229') 1
```

L'affichage des 5 premières paires de nœuds et de leurs labels permet de vérifier que les données ont été correctement chargées et préparées pour la suite de l'analyse. Affichage de nombre de nœuds et de labels.

- Affichage le nombre des nœuds et des labels dans le graphe

```python
import gzip
import networkx as nx

# Create a graph
G = nx.Graph()

# Add edges from the data
for line in data:
    # Skip lines that start with #
    if line.decode('utf-8').startswith("#"):
        continue

    # Convert the line to string
    line = line.decode('utf-8').strip()

    # Separate the source and target nodes
    source, target = line.split('\t')

    # Add the edge to the graph
    G.add_edge(source, target)

# Print the number of nodes and edges in the graph
print("Number of nodes:", G.number_of_nodes())
print("Number of edges:", G.number_of_edges())
```

- Le résultat

```
Number of nodes: 18772
Number of edges: 198110
```

L'affichage des statistiques du graphe : l'affichage du nombre de nœuds et d'arêtes du graphe.

- Convertir les paires de nœuds en caractéristiques (features)

```
import numpy as np
from sklearn.model_selection import train_test_split

# Convertir les paires de nœuds en caractéristiques (features)
# Pour l'exemple, nous pourrions utiliser simplement le degré des nœuds source et destination
X = np.array([(G.degree(node1), G.degree(node2)) for node1, node2 in node_pairs])

# Définir les labels
y = np.array(labels)

# Diviser les données en ensemble d'apprentissage et ensemble de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Construction du modèle KNN

```
knn_model = KNeighborsClassifier(n_neighbors=5)
```

- Entraînement du modèle

```
knn_model.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

- Prédiction des liens sur l'ensemble de test

```
y_pred = knn_model.predict(X_test)
```

- Évaluation du modèle

```
accuracy = accuracy_score(y_test, y_pred)
print("Précision du modèle KNN :", accuracy)
```

- Le résultat de précision

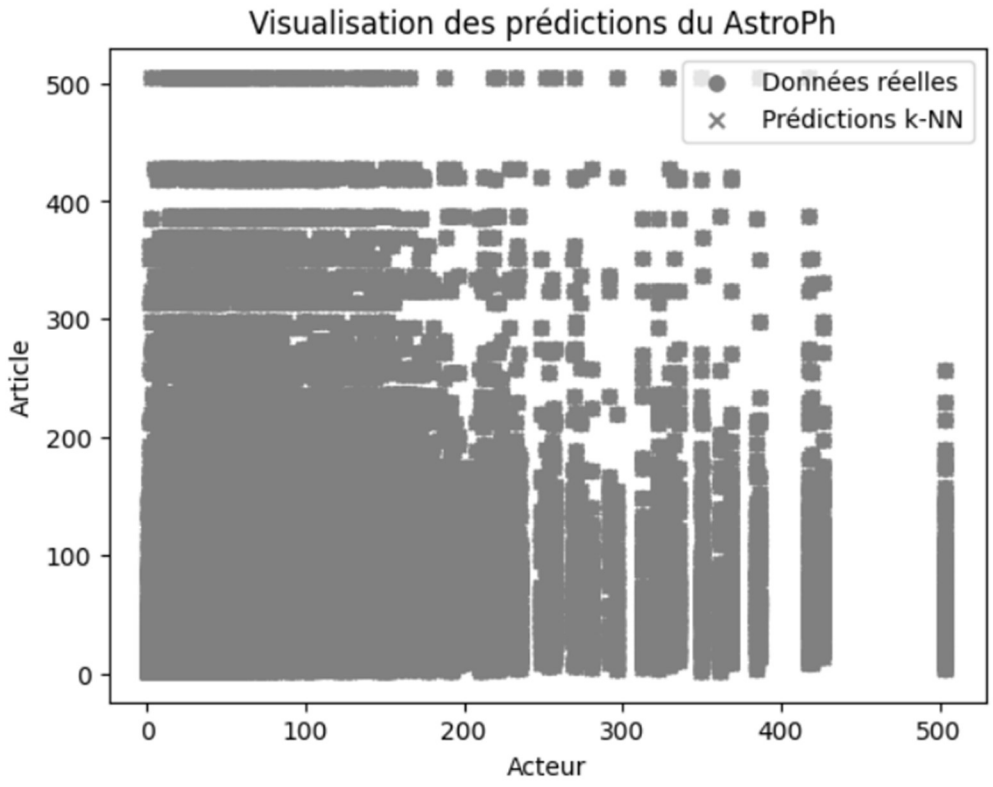
```
Précision du modèle KNN : 1.0
```

Le résultat de l'utilisation de l'algorithme k-NN sur votre jeu de données montre que le modèle a réussi à apprendre efficacement à prédire les liens dans le réseau de collaboration. La précision parfaite obtenue suggère que le modèle est très performant sur ces données spécifiques.

- Visualisation des prédictions du AstroPh

```
import numpy as np
import matplotlib.pyplot as plt

# Définir les couleurs pour les étiquettes réelles
colors = np.array(['blue', 'Gray'])

# Tracer les données de test avec les étiquettes réelles
plt.scatter(X_test[:, 0], X_test[:, 1], color=colors[y_test], label='Données réelles')

# Tracer les prédictions du modèle k-NN
plt.scatter(X_test[:, 0], X_test[:, 1], color=colors[y_pred], marker='x', label='Prédictions k-NN')

# Ajouter des légendes et un titre
plt.xlabel('Acteur')
plt.ylabel('Article')
plt.title('Visualisation des prédictions du AstroPh ')
plt.legend()

# Afficher le graphique
plt.show()
```

- Le résultat



Le graphique contient deux types de marqueurs :

Des points gris étiquetés "Données réelles" : Ces points sont densément regroupés en bandes horizontales, indiquant des clusters d'articles associés à des acteurs spécifiques.

Des croix noires étiquetées "Prédictions k-NN" : Ces croix sont moins densément dispersées mais semblent suivre la distribution générale des points gris, représentant les prédictions faites par l'algorithme k-NN correspondant aux données réelles.

Cette visualisation est probablement utilisée pour comparer les performances du modèle de prédiction k-NN par rapport aux données réellement observées dans le réseau de collaboration en physique des astroparticules.