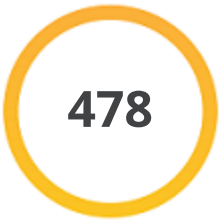
SENA_Project.py

Scanned on: 18:11 November 10, 2022 UTC

Overall Similarity Score

Results Found



Total Words in Text

Identical Words	0
Words with Minor Changes	0
Paraphrased Words	0
Omitted Words	0

SENA_Project.py

Scanned on: 18:11 November 10, 2022 UTC

Results

Sources that matched your submitted document.

No results found. The content is free of plagiarism.

IDENTICAL

Identical matches are one to one exact wording in the text.

MINOR CHANGES

Nearly identical with different form, ie "slow" becomes "slowly".

PARAPHRASED

Close meaning but different words used to convey the same message.

Unsure about your report?

The results have been found after comparing your submitted text to online sources, open databases and the Copyleaks internal database. For any questions about the report contact us on support@copyleaks.com

Learn more about different kinds of plagiarism here

SENA_Project.py

Scanned on: 18:11 November 10, 2022 UTC

Scanned Text

Your text is highlighted according to the matched content in the results above.

IDENTICAL MINOR CHANGES PARAPHRASED

```
from networkx.algorithms import bipartite
import csv
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np


movies = []
with open('SENA_DataSet - MCUActorList.csv',encoding="utf8") as f:
file = csv.reader(f)
for line in file:
movies.append(line[0])
movies


actors = set()
with open('SENA_DataSet - MCUActorList.csv', encoding="utf8") as f:
file = csv.reader(f)
for line in file:
for actor in line[1:]:
if actor != '':
actors.add(actor)
actors


ratings = {}
with open('SENA_DataSet - MCUCriticRatingList.csv', encoding="utf8") as f:
file = csv.reader(f)
for line in file:
if line[-1] != '':
ratings[line[0]] = int(line[-1])
print('Normalized ratings : ',ratings)


G = nx.Graph()


G.add_nodes_from(actors,bipartite='Actors')
G.add_nodes_from([(m,{'rating':ratings[m]}) for m in movies],bipartite='Movies and Series')
```

```
with open('SENA_DataSet - MCUActorList.csv', encoding = "utf8") as f:
file = csv.reader(f)
for line in file:
movie = line[0]
for i in line[1:]:
if i != '':
G.add_edge(movie,i)


actor_nodes = [n for n,d in G.nodes(data=True) if d['bipartite'] == 'Actors']
movie_nodes = [n for n,d in G.nodes(data=True) if d['bipartite'] == 'Movies and Series']
print('Actors : ',actor_nodes,end='\n\n')
print('Movies and Series : ',movie_nodes,end='\n\n')
print('Is Bipartite : ',bipartite.is_bipartite(G),end='\n\n')
print(f'Number of Actors : {len(actor_nodes)}\n\nNumber of Movies and Series : {len(movie_nodes)}')


nx.draw_networkx(G,pos=nx.drawing.bipartite_layout(G,actors),width=1)


nx.write_gexf(G,'MCU_actors_movies_bipartite.gexf')


actors_movie_count = {}
for n ,d in G.nodes(data=True):
if d['bipartite'] == 'Actors':
actors_movie_count[n] = G.degree[n]
actors_movie_count = sorted(list(actors_movie_count.items()),key=lambda x : x[1],reverse=True)
actors_movie_count


counts, bins = np.histogram([i[1] for i in actors_movie_count])
plt.stairs(counts, bins)
plt.xlabel("Movie Count")
plt.ylabel("Actor Count")
plt.title('Movie-Actor Count')


potential_actors = {}
for n , d in G.nodes(data=True):
if d['bipartite'] == 'Actors' and G.degree[n] <= 2:
rating = 0
for neighbour in G.neighbors(n):
rating += G.nodes[neighbour]['rating']
rating /= G.degree[n]
potential_actors[n] = rating
potential_actors = sorted(list(potential_actors.items()),key=lambda x : x[1],reverse=True)
potential_actors


result_movies = {}
for n,d in G.nodes(data=True):
if d['bipartite'] == 'Movies and Series':
count = 0
for i in G.neighbors(n):
if G.degree[i] == 1:
count += 1
result_movies[n] = count
result_movies = sorted(list(result_movies.items()),key=lambda x : x[1],reverse=True)
result_movies


counts, bins = np.histogram([i[1] for i in result_movies])
plt.stairs(counts, bins)
plt.xlabel("Debutant Count")
```

```
plt.ylabel("Movie Count")
plt.title('Debutants Count')


B = bipartite.weighted_projected_graph(G, actors, ratio=False)
B.edges(data=True)


nx.write_gexf(B,'MCU_actors_projections.gexf')


pair_count=sorted(B.edges(data=True), key=lambda t: t[2]['weight'],reverse=True)
pair_count


counts, bins = np.histogram([i[2]['weight'] for i in pair_count])
plt.stairs(counts, bins)
plt.xlabel("No. of Movies")
plt.ylabel("No. of Pairs")
plt.title('Counts of Pairs in Same Movie')


number_of_connected_components = nx.number_connected_components(B)
print('Number of Connected Components : ', number_of_connected_components,end='\n\n')
connected_components = list(nx.connected_components(B))
for i in connected_components:
print(i,end='\n\n')


largest_connected_component = list(sorted(connected_components,reverse=True,key=len)[0])
print('Number of actors in largest connected components : ',len(largest_connected_component))
sub_graph = B.subgraph(largest_connected_component).copy()
nx.draw_networkx(sub_graph)


nx.write_gexf(sub_graph,'MCU_actors_largest_component.gexf')


edge_betweenness_centrality = nx.edge_betweenness_centrality(sub_graph)
edge_betweenness_centrality = sorted(list(edge_betweenness_centrality.items()),key=lambda x :
x[1],reverse=True)
edge_betweenness_centrality


from networkx.algorithms import community

comp = community.girvan_newman(sub_graph)
communities = tuple(c for c in next(comp))
print('Communities detected : ',len(communities),end='\n\n')
for i in communities:
print(i,end='\n\n')
```