Clustering Assignment

There will be some functions that start with the word "grader" ex: grader_actors(), grader_movies(), grader_cost1() etc, you should not change those function definition.

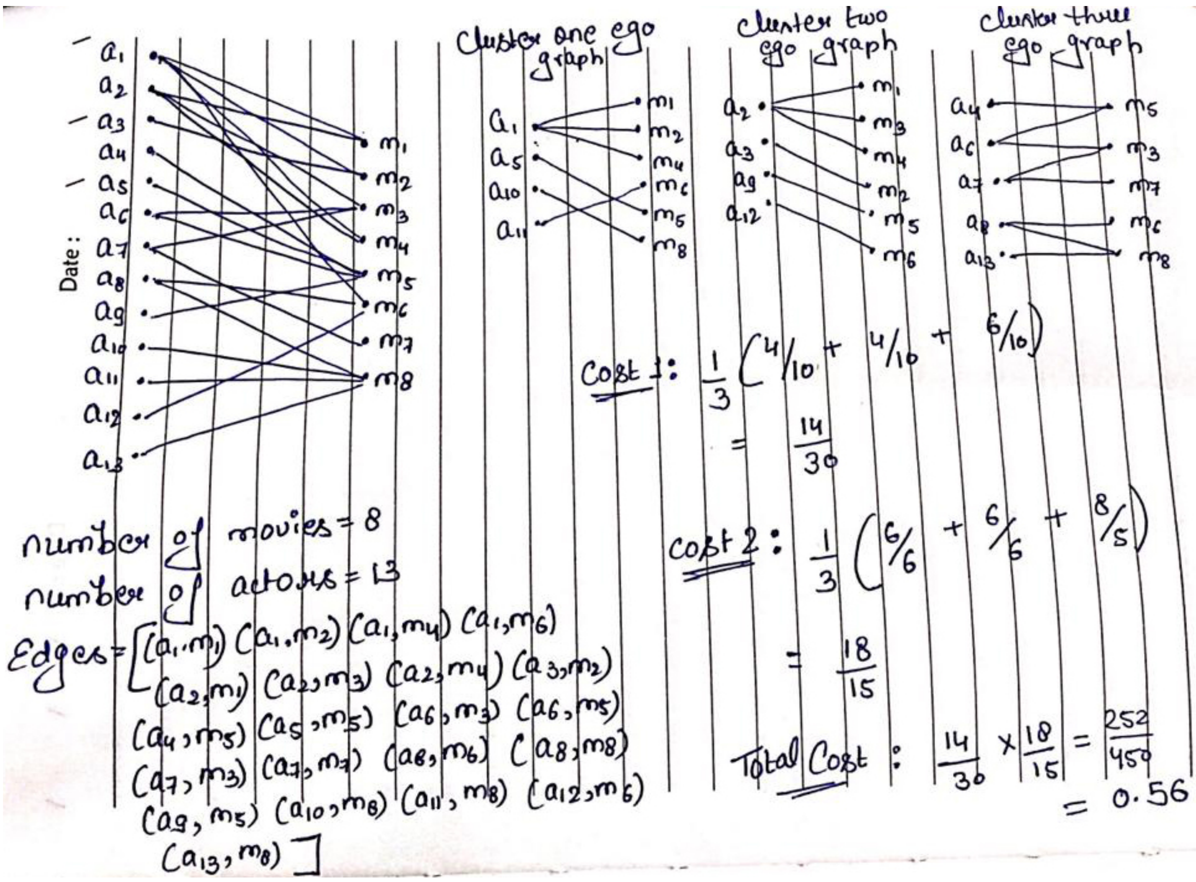
Every Grader function has to return True.

Please check [clustering assignment helper functions (https://drive.google.com/file/d/1V29KhKo3YnckMX32treEgdtH5r90DIjU/view?usp=sharing)](https://drive.google.com/file/d/1V29KhKo3YnckMX32treEgdtH5r90DIjU/view?usp=sharing) notebook before attempting this assignment.

- Read graph from the given movie_actor_network.csv (note that the graph is bipartite graph.)
- Using stellergaph and gensim packages, get the dense representation(128dimensional vector) of every node in the graph. [Refer Clustering_Assignment_Reference.ipynb]
- Split the dense representation into actor nodes, movies nodes.(Write you code in def data_split())

Task 1 : Apply clustering algorithm to group similar actors

1. For this task consider only the actor nodes
2. Apply any clustering algorithm of your choice
   Refer : [https://scikit-learn.org/stable/modules/clustering.html (https://scikit-learn.org/stable/modules/clustering.html)](https://scikit-learn.org/stable/modules/clustering.html)
3. Choose the number of clusters for which you have maximum score of $Cost1 * Cost2$
4. Cost1 =
   $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(number of nodes in the largest connected component in the graph with the actor nodes an}}{\text{(total number of nodes in that cluster i)}}$
   where N= number of clusters
   (Write your code in def cost1())
5. Cost2 =
   $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(sum of degress of actor nodes in the graph with the actor nodes and its movie neighbours}}{\text{(number of unique movie nodes in the graph with the actor nodes and its movie neighbour}}$
   where N= number of clusters
   (Write your code in def cost2())
6. Fit the clustering algorithm with the opimal number_of_clusters and get the cluster number for each node
7. Convert the d-dimensional dense vectors of nodes into 2-dimensional using dimensionality reduction techniques (preferably TSNE)
8. Plot the 2d scatter plot, with the node vectors after step e and give colors to nodes such that same cluster nodes will have same color



Task 2 : Apply clustering algorithm to group similar movies

1. For this task consider only the movie nodes
2. Apply any clustering algorithm of your choice
3. Choose the number of clusters for which you have maximum score of $Cost1 * Cost2$

   Cost1 =
   $$\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(number of nodes in the largest connected component in the graph with the movie nodes a}}{\text{(total number of nodes in that cluster i)}}$$
   where N= number of clusters
   (Write your code in def cost1())
4. Cost2 =
   $$\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(sum of degress of movie nodes in the graph with the movie nodes and its actor neighbour}}{\text{(number of unique actor nodes in the graph with the movie nodes and its actor neighbours}}$$
   where N= number of clusters
   (Write your code in def cost2())

Algorithm for actor nodes

```
    for number_of_clusters in [3, 5, 10, 30, 50, 100, 200, 500]:
        algo = clustering_algorith(clusters=number_of_clusters)
        # you will be passing a matrix of size N*d where N number of actor nodes and d is dimension from gensim
        algo.fit(the dense vectors of actor nodes)
        You can get the labels for corresponding actor nodes (algo.labels_)
        Create a graph for every cluster(ie., if n_clusters=3, create 3 graphs)
        (You can use ego_graph to create subgraph from the actual graph)
        compute cost1,cost2
           (if n_cluster=3, cost1=cost1(graph1)+cost1(graph2)+cost1(graph3) # here we are doing summation
            cost2=cost2(graph1)+cost2(graph2)+cost2(graph3)
        computer the metric Cost = Cost1*Cost2
    return number_of_clusters which have maximum Cost
```

In [1]:
```
!pip install networkx==2.3
```

```
Collecting networkx==2.3
  Downloading https://files.pythonhosted.org/packages/85/08/f20aef
11d4c343b557e5de6b9548761811eb16e438cee3d32b1c66c8566b/networkx-2.
3.zip
(https://files.pythonhosted.org/packages/85/08/f20aef11d4c343b557e
5de6b9548761811eb16e438cee3d32b1c66c8566b/networkx-2.3.zip)
(1.7MB)
     |████████████████████████████████| 1.8MB 9.4MB/s eta 0:00:01
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/
python3.6/dist-packages (from networkx==2.3) (4.4.2)
Building wheels for collected packages: networkx
  Building wheel for networkx (setup.py) ... done
  Created wheel for networkx: filename=networkx-2.3-py2.py3-none-a
ny.whl size=1556408 sha256=255b1044b5a0e020bec46f972ba40eb90b2d575
09b9d9df9fdf5899566e83e81
  Stored in directory: /root/.cache/pip/wheels/de/63/64/3699be2a9d
0ccdb37c7f16329acf3863fd76eda58c39c737af
Successfully built networkx
ERROR: albumentations 0.1.12 has requirement imgaug<0.2.7,>=0.2.5,
but you'll have imgaug 0.2.9 which is incompatible.
Installing collected packages: networkx
  Found existing installation: networkx 2.5
    Uninstalling networkx-2.5:
      Successfully uninstalled networkx-2.5
Successfully installed networkx-2.3
```

In [3]:
```
import networkx as nx
from networkx.algorithms import bipartite
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
# import stellargraph as sg
# you need to have tensorflow
from stellargraph.data import UniformRandomMetaPathWalk
from stellargraph import StellarGraph
```

In [5]:
```
data=pd.read_csv('movie_actor_network.csv', index_col=False, names=
```

In [6]:
```
data.head(5)
```

Out[6]:

	movie	actor
0	m1	a1
1	m2	a1
2	m2	a2
3	m3	a1
4	m3	a3

In [7]:
```
edges = [tuple(x) for x in data.values.tolist()]
```

In [8]:
```
B = nx.Graph()
B.add_nodes_from(data['movie'].unique(), bipartite=0, label='movie'
B.add_nodes_from(data['actor'].unique(), bipartite=1, label='actor'
B.add_edges_from(edges, label='acted')
```

In [9]:
```
A = list(nx.connected_component_subgraphs(B))[0]
```

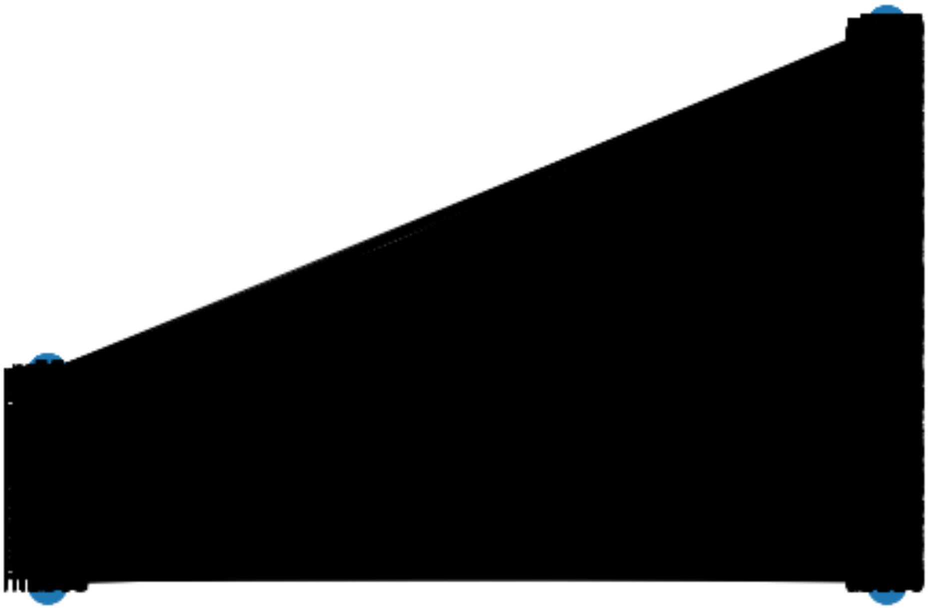
In [10]:
```
print("number of nodes", A.number_of_nodes())
print("number of edges", A.number_of_edges())
```

```
number of nodes 4703
number of edges 9650
```

In [11]:
```
l, r = nx.bipartite.sets(A)
pos = {}

pos.update((node, (1, index)) for index, node in enumerate(l))
pos.update((node, (2, index)) for index, node in enumerate(r))

nx.draw(A, pos=pos, with_labels=True)
plt.show()
```



In [12]:
```
movies = []
actors = []
for i in A.nodes():
    if 'm' in i:
        movies.append(i)
    if 'a' in i:
        actors.append(i)
print('number of movies ', len(movies))
print('number of actors ', len(actors))
```

```
number of movies  1292
number of actors  3411
```

In [13]:
```

# Create the random walker
rw = UniformRandomMetaPathWalk(StellarGraph(A))

# specify the metapath schemas as a list of lists of node types.
metapaths = [
    ["movie", "actor", "movie"],
    ["actor", "movie", "actor"]
]

walks = rw.run(nodes=list(A.nodes()), # root nodes
               length=100,  # maximum length of a random walk
               n=1,        # number of random walks per root node
               metapaths=metapaths
              )

print("Number of random walks: {}".format(len(walks)))
```

```
Number of random walks: 4703
```

In [14]:
```
from gensim.models import Word2Vec
model = Word2Vec(walks, size=128, window=5)
```

In [15]:
```
model.wv.vectors.shape  # 128-dimensional vector for each node in t
```

Out[15]:
```
(4703, 128)
```

In [16]:
```
# Retrieve node embeddings and corresponding subjects
node_ids = model.wv.index2word  # list of node IDs
node_embeddings = model.wv.vectors  # numpy.ndarray of size number
node_targets = [ A.node[node_id]['label'] for node_id in node_ids]
```

```
print(node_ids[:15], end='')
```

```
['a973', 'a967', 'a964', 'a1731', 'a969', 'a970', 'a1028', 'a1057', 'a965', 'a1003', 'm1094', 'a966', 'm67', 'a988', 'm1111']
```

```
print(node_targets[:15],end='')
```

```
['actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'movie', 'actor', 'movie', 'actor', 'movie']
```

In [17]:
```
print(node_ids)
print(node_embeddings.shape)
print(node_targets)
```

```
['a973', 'a967', 'a964', 'a1731', 'a970', 'a969', 'a1057', 'a1003'
, 'a1028', 'm1094', 'a965', 'a959', 'm67', 'a988', 'a966', 'm1111'
, 'a49', 'a1037', 'm1100', 'a93', 'a962', 'a963', 'a971', 'a960',
'a1076', 'a1030', 'a1016', 'a977', 'a204', 'm1095', 'a1027', 'a631
', 'a472', 'a768', 'a968', 'a2715', 'm1114', 'a1004', 'a1020', 'm1
001', 'a407', 'a1507', 'a1035', 'a1026', 'a972', 'a306', 'a138', '
a975', 'a1031', 'm1096', 'm376', 'm987', 'a1036', 'm1113', 'm1261'
, 'm1022', 'm1112', 'm1232', 'm165', 'm816', 'm1357', 'a1008', 'a9
87', 'm1097', 'a1435', 'm148', 'a1021', 'a1038', 'm453', 'a205', '
m1220', 'a976', 'a782', 'a1015', 'm32', 'm26', 'm25', 'm126', 'a92
1', 'm1106', 'm1272', 'm1000', 'a1467', 'm121', 'a974', 'm964', 'm
75', 'a1060', 'm115', 'm1213', 'm914', 'm796', 'm990', 'a1005', 'a
1011', 'a893', 'a228', 'a1750', 'a1436', 'm154', 'm616', 'm1101',
'a1505', 'm988', 'm122', 'm1090', 'm983', 'm915', 'm1011', 'm995',
'm1348', 'm1116', 'm1045', 'm157', 'm1098', 'm982', 'm1048', 'm587
', 'm902', 'm1023', 'm147', 'm159', 'a1018', 'm1092', 'm1024', 'm1
284', 'm1110', 'm918', 'a1029', 'm1138', 'm129', 'a1023', 'm377',
'm1169', 'm145', 'm128', 'm973', 'm1030', 'a1039', 'm743', 'm430',
'm1212', 'a363', 'a1032', 'm155', 'a1025', 'm1108', 'm963', 'm850'
```

In [18]:
```
def data_split(node_ids,node_targets,node_embeddings):
    '''In this function, we will split the node embeddings into act
    actor_nodes,movie_nodes=[],[]
    actor_embeddings,movie_embeddings=[],[]
    # split the node_embeddings into actor_embeddings,movie_embeddi
    # By using node_ids and node_targets, we can extract actor_node
    for i in range(0,len(node_ids)):
      if node_targets[i]=='movie':
        movie_embeddings.append(node_embeddings[i])
        movie_nodes.append(node_ids[i])
      else:
        actor_embeddings.append(node_embeddings[i])
        actor_nodes.append(node_ids[i])

    return actor_nodes,movie_nodes,actor_embeddings,movie_embedding
```

In [19]:
```
actor_nodes,movie_nodes,actor_embeddings,movie_embeddings = data_sp
```

Grader function - 1

In [20]:
```
def grader_actors(data):
    assert(len(data)==3411)
    return True
grader_actors(actor_nodes)
```

Out[20]: True

Grader function - 2

In [21]:
```
def grader_movies(data):
    assert(len(data)==1292)
    return True
grader_movies(movie_nodes)
```

Out[21]: True

Calculating cost1

Cost1 =
$\frac{1}{N} \sum_{\text{each cluster i}} \frac{\text{(number of nodes in the largest connected component in the graph with the actor nodes and its m}}{\text{(total number of nodes in that cluster i)}}$
where N= number of clusters

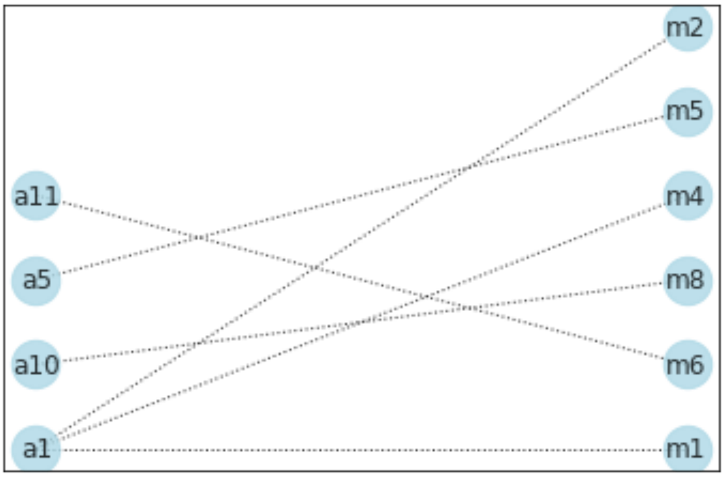
In [22]:
```
def cost1(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''

    connected = nx.connected_components(graph)
    max_cc = max(connected, key=len)
    number_of_nodes_in_cc = len(max_cc)
    total_nodes = graph.number_of_nodes()

    cost1= (1/number_of_clusters) * (number_of_nodes_in_cc / total_

    return cost1
```

In [23]:
```
import networkx as nx
from networkx.algorithms import bipartite
graded_graph= nx.Graph()
graded_graph.add_nodes_from(['a1','a5','a10','a11'], bipartite=0) #
graded_graph.add_nodes_from(['m1','m2','m4','m6','m5','m8'], bipart
graded_graph.add_edges_from([('a1','m1'),('a1','m2'),('a1','m4'),('
l={'a1','a5','a10','a11'};r={'m1','m2','m4','m6','m5','m8'}
pos = {}
pos.update((node, (1, index)) for index, node in enumerate(l))
pos.update((node, (2, index)) for index, node in enumerate(r))
nx.draw_networkx(graded_graph, pos=pos, with_labels=True,node_color
```



Grader function - 3

In [24]:
```
graded_cost1=cost1(graded_graph,3)
def grader_cost1(data):
    assert(data==((1/3)*(4/10))) # 1/3 is number of clusters
    return True
grader_cost1(graded_cost1)
```

Out[24]: True

Calculating cost2

Cost2 =

$$\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(sum of degress of actor nodes in the graph with the actor nodes and its movie neighbours in clu}}{\text{(number of unique movie nodes in the graph with the actor nodes and its movie neighbours in cl}}$$

where N= number of clusters

In [25]:
```
def cost2(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''

    actors = []
    movies = []
    sum_of_degrees = unique_movies = 0

    for node in graph.nodes():
      if 'a' in node:
        actors.append(node)
      else:
        movies.append(node)

    unique_movies = len(movies)
    for a in actors:
      sum_of_degrees += graph.degree(a)

    cost2= (1/number_of_clusters) * (sum_of_degrees/unique_movies)

    return cost2
```

Grader function - 4

In [26]:
```
graded_cost2=cost2(graded_graph,3)
def grader_cost2(data):
    assert(data==((1/3)*(6/6))) # 1/3 is number of clusters
    return True
grader_cost2(graded_cost2)
```

Out[26]: True

Grouping similar actors

In [27]:
```
from sklearn.cluster import KMeans

number_of_clusters = [3, 5, 10, 30, 50, 100, 200, 500]
Cost={}

for i in number_of_clusters:
  model_k = KMeans(n_clusters=i, random_state=0)
  model_k.fit(actor_embeddings)
  actor_labels = model_k.labels_
  unique_clusters = np.unique(actor_labels)
  dict_of_actor_nodes = dict(zip(actor_nodes, actor_labels))
  list_of_clusters = []
  for n in unique_clusters:
    clusters = []
    for node, cluster in dict_of_actor_nodes.items():
      if cluster == n:
        clusters.append(node)
    list_of_clusters.append(clusters)

  Cost1 = 0
  Cost2 = 0

  for cluster in list_of_clusters:
    G = nx.Graph()
    for node in cluster:
      subgraph = nx.ego_graph(B, node)
      G.add_nodes_from(subgraph.nodes())
      G.add_edges_from(subgraph.edges())

    Cost1 += cost1(G, len(list_of_clusters))
    Cost2 += cost2(G, len(list_of_clusters))

  Cost[i] = Cost1*Cost2
```

In [28]:
```
Cost
```

Out[28]:
```
{3: 3.7203671609990723,
 5: 3.0646420904674807,
 10: 2.314357257923274,
 30: 1.7472179188042063,
 50: 1.8862603650374872,
 100: 1.658561915875417,
 200: 1.7524042212579942,
 500: 1.8276966939381516}
```

In [29]:
```
model = KMeans(n_clusters=3)
model.fit(actor_embeddings)

predict = model.predict(actor_embeddings)
predict
```

Out[29]:
```
array([1, 1, 1, ..., 0, 0, 0], dtype=int32)
```

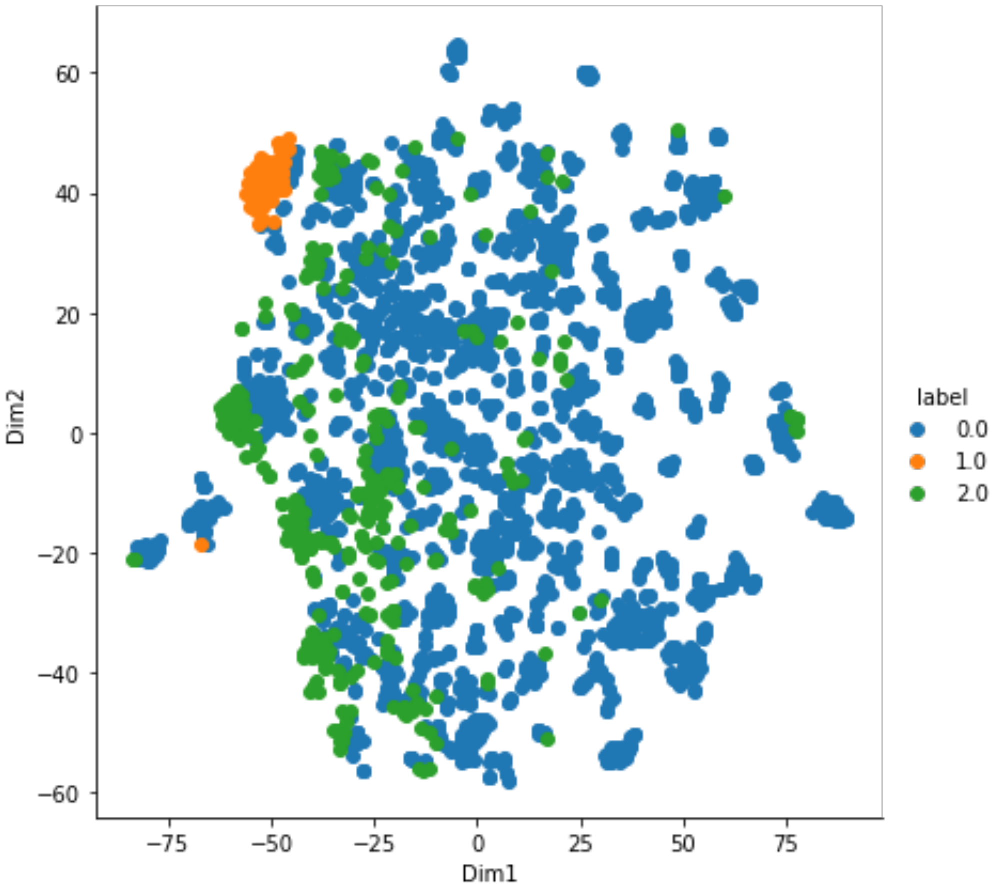
In [30]:
```
from sklearn.manifold import TSNE
transform = TSNE #PCA

trans = transform(n_components=2)
twoD_data = trans.fit_transform(actor_embeddings)
```

In [31]:
```
req_data = np.vstack((twoD_data.T, predict.T))
final_data = pd.DataFrame(req_data.T, columns=("Dim1", "Dim2", "lab
```

Displaying similar actor clusters

In [32]:
```
import seaborn as sns
sns.FacetGrid(final_data, hue="label", size=6).map(plt.scatter, "Di
plt.show()
```



Grouping similar movies

In [33]:
```
from sklearn.cluster import KMeans

Cost={}

for i in number_of_clusters:
  model_k = KMeans(n_clusters=i, random_state=0)
  model_k.fit(movie_embeddings)
  movie_labels = model_k.labels_
  unique_clusters = np.unique(movie_labels)
  dict_of_movie_nodes = dict(zip(movie_nodes, movie_labels))
  list_of_clusters = []
  for n in unique_clusters:
    clusters = []
    for node, cluster in dict_of_movie_nodes.items():
      if cluster == n:
        clusters.append(node)
    list_of_clusters.append(clusters)

  Cost1 = 0
  Cost2 = 0

  for cluster in list_of_clusters:
    G = nx.Graph()
    for node in cluster:
      subgraph = nx.ego_graph(B, node)
      G.add_nodes_from(subgraph.nodes())
      G.add_edges_from(subgraph.edges())

    Cost1 += cost1(G, len(list_of_clusters))
    Cost2 += cost2(G, len(list_of_clusters))

  Cost[i] = Cost1*Cost2
```

In [34]:
```
Cost
```

Out[34]:
```
{3: 8.390984895667694,
 5: 8.810817169244412,
 10: 9.195355972005016,
 30: 11.452968950826092,
 50: 13.6475579364147,
 100: 13.744249909737656,
 200: 12.87422187622036,
 500: 10.373694115050744}
```

In [39]:
```
model = KMeans(n_clusters=100)
model.fit(movie_embeddings)

predict = model.predict(movie_embeddings)
predict
```

Out[39]:
```
array([25, 59, 54, ..., 14, 14, 14], dtype=int32)
```

In [40]:
```
from sklearn.manifold import TSNE
transform = TSNE #PCA

trans = transform(n_components=2)
twoD_data = trans.fit_transform(movie_embeddings)
```

In [41]:
```
req_data = []
final_data = []
req_data = np.vstack((twoD_data.T, predict.T))
final_data = pd.DataFrame(req_data.T, columns=("Dim1", "Dim2", "lab
```

Displaying similar movie clusters

In [42]:
```
import seaborn as sns
sns.FacetGrid(final_data, hue="label", size=6).map(plt.scatter, "Di
plt.show()
```