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.

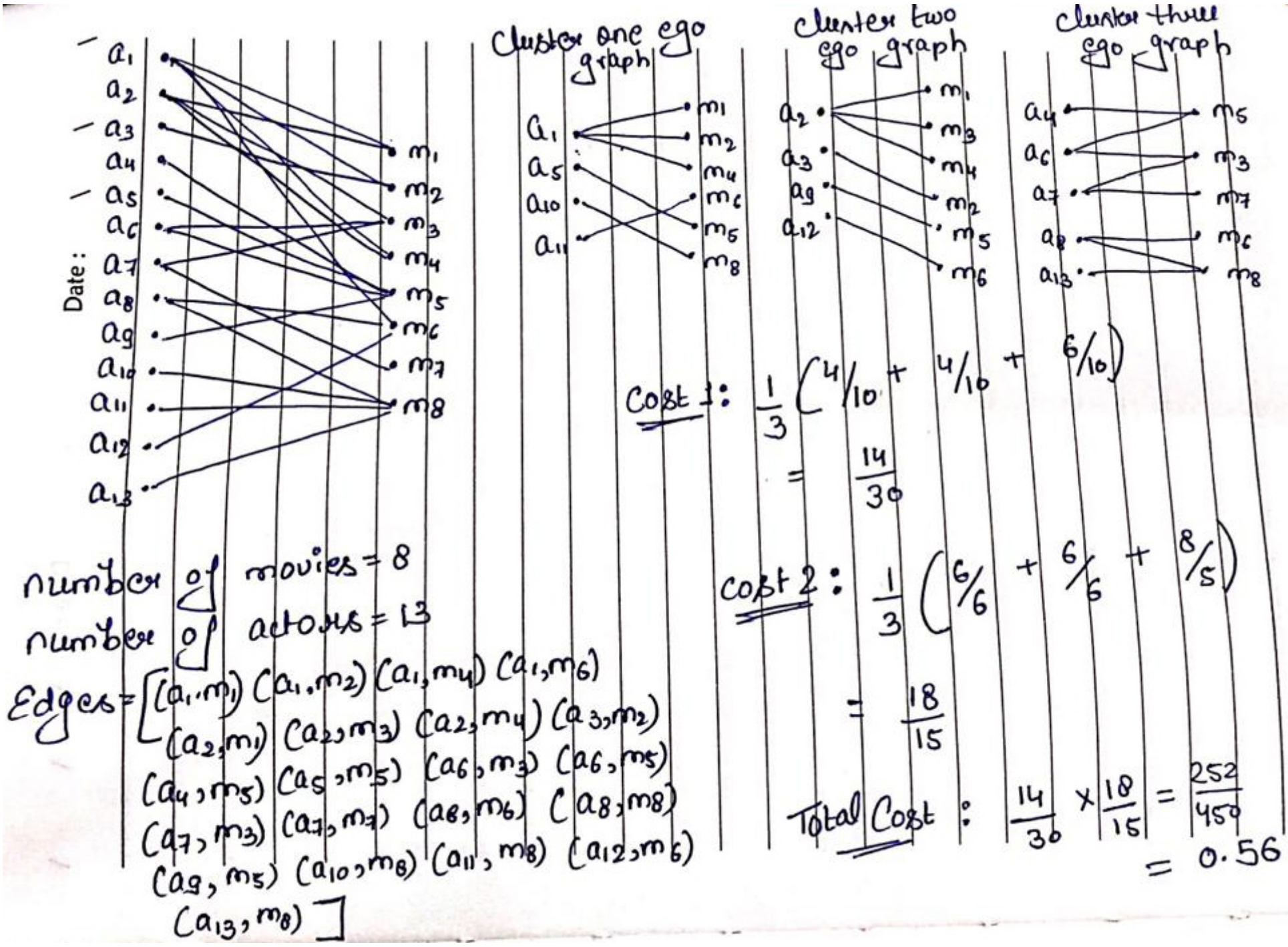
Every Grader function has to return True.

Please check clustering assignment helper functions 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
3. Choose the number of clusters for which you have maximum score of $Cost1 * Cost2$
4. Cost1 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(number of nodes in the largest connected component in the graph with the actor nodes and its movie neighbours in cluster i)}}{\text{(total number of nodes in that cluster i)}}$
   where N= number of clusters
   (Write your code in def cost1())
5. Cost2 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(sum of degress of actor nodes in the graph with the actor nodes and its movie neighbours in cluster i)}}{\text{(number of unique movie nodes in the graph with the actor nodes and its movie neighbours in cluster i)}}$ 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 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(number of nodes in the largest connected component in the graph with the movie nodes and its actor neighbours in cluster i)}}{\text{(total number of nodes in that cluster i)}}$

   where N= number of clusters

   (Write your code in def cost1())
4. Cost2 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(sum of degress of movie nodes in the graph with the movie nodes and its actor neighbours in cluster i)}}{\text{(number of unique actor nodes in the graph with the movie nodes and its actor neighbours in cluster i)}}$ 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
```

Importing libraries

```
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
# you need to have tensorflow
from stellargraph.data import UniformRandomMetaPathWalk
from stellargraph import StellarGraph
```

Importing dataset

```
from google.colab import files
uploaded = files.upload()
```

```
Choose Files movie_actor_network.csv
```

- **movie_actor_network.csv**(application/vnd.ms-excel) - 114060 bytes, last modified: 5/3/2020 - 100% done

```
Saving movie_actor_network.csv to movie_actor_network.csv
```

```
data=pd.read_csv('movie_actor_network.csv', index_col=False, names=['movie','actor'])
```

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

```
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')
```

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

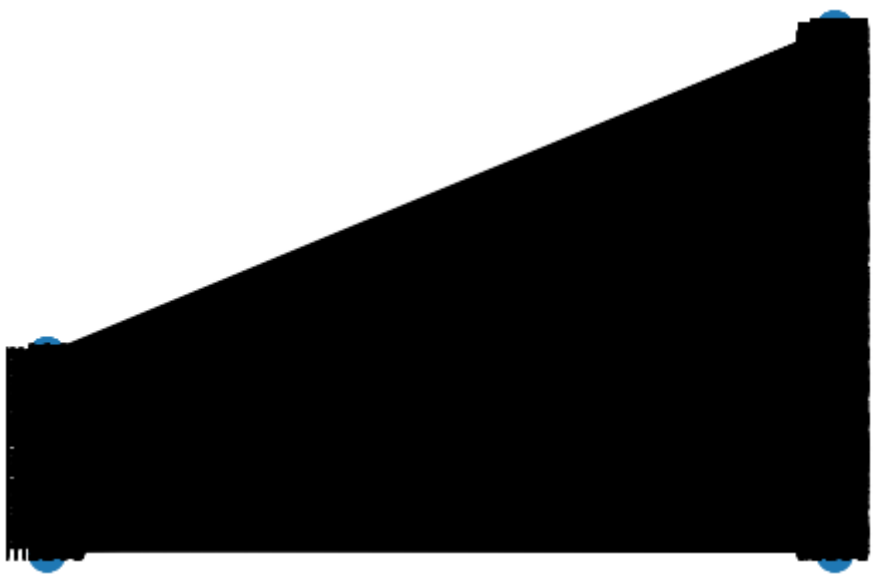
```
print("number of nodes", A.number_of_nodes())
print("number of edges", A.number_of_edges())
```

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

```
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()
```



```
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
```

```

# 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
```

```
from gensim.models import Word2Vec
model = Word2Vec(walks, size=128, window=5)
```

```
model.wv.vectors.shape  # 128-dimensional vector for each node in the graph
```

```
(4703, 128)
```

```
# 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 of nodes times embeddings dimensionality
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']
```

```
def data_split(node_ids,node_targets,node_embeddings):
    '''In this function, we will split the node embeddings into actor_embeddings , movie_embeddings '''
    actor_nodes,movie_nodes=[],[]
    actor_embeddings,movie_embeddings=[],[]
    for i in range(len(node_ids)):
      if 'm' in node_targets[i]:
        movie_nodes.append(node_ids[i])
        movie_embeddings.append(node_embeddings[i])
      else:
        actor_nodes.append(node_ids[i])
        actor_embeddings.append(node_embeddings[i])


    return actor_nodes,movie_nodes,actor_embeddings,movie_embeddings

```

```
actor_nodes,movie_nodes,actor_embeddings,movie_embeddings=data_split(node_ids,node_targets,node_embeddings)
```

Graded function - 1

```
def grader_actors(data):
    assert(len(data)==3411)
    return True
grader_actors(actor_nodes)
```

```
True
```

Graded function - 2

```
def grader_movies(data):
    assert(len(data)==1292)
    return True
grader_movies(movie_nodes)
```

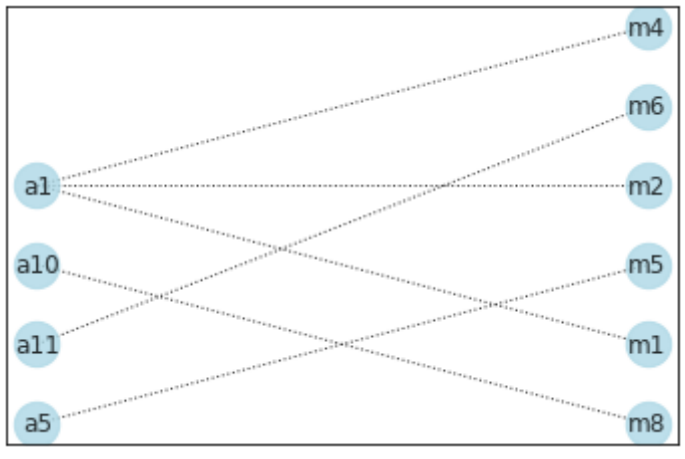
```
True
```

Calculating cost1

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

```
def cost_1(graph,cluster):
  Gc = max(nx.connected_component_subgraphs(graph), key=len)
  connected=Gc.number_of_nodes()
  total_nodes=graph.number_of_nodes()
  value=((1/cluster)*(connected/total_nodes))
  return value
```

```
import networkx as nx
from networkx.algorithms import bipartite
graded_graph= nx.Graph()
graded_graph.add_nodes_from(['a1','a5','a10','a11'], bipartite=0) # Add the node attribute "bipartite"
graded_graph.add_nodes_from(['m1','m2','m4','m6','m5','m8'], bipartite=1)
graded_graph.add_edges_from([('a1','m1'),('a1','m2'),('a1','m4'),('a11','m6'),('a5','m5'),('a10','m8')])
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='lightblue',alpha=0.8,style='dotted',node_size=500)
```



Graded function - 3

```
graded_cost1=cost_1(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)
```

```
True
```

Calculating cost2

Cost2 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(sum of degress of actor nodes in the graph with the actor nodes and its movie neighbours in cluster i)}}{\text{(number of unique movie nodes in the graph with the actor nodes and its movie neighbours in cluster i)}}$ where N= number of clusters

```
def cost_2(graph,cluster):
  degree=graph.degree()
  degree_value=0 # to store sum of all  degree of actor nodes
  mov_nodes=0  #to store sum of all unique actor nodes
  for i in degree:
    if "a" in i[0]:
      degree_value=degree_value+i[1]
    else:
```

```
      mov_nodes=mov_nodes+1
  value=((1/cluster)*(degree_value/mov_nodes))
  return value

```

Graded function - 4

```
graded_cost2=cost_2(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)
```

```
True
```

Grouping similar actors

```
from sklearn.cluster import KMeans
```

```
cost_value={}
for clu in [3,5,10,50,200,300]:
  cost1=0
  cost2=0
  label=[]
  algo = KMeans(n_clusters=clu)
  algo.fit(actor_embeddings) # FITTING WITH KMEANS ALGO
  label=algo.labels_
  for i in range(clu):
     G1=nx.Graph()  # drawing graph for each cluster
     label_divi=[]
     k=[index for index, value in enumerate(label) if value == i]# accesing each cluster basesd on labels_
     label_divi=[ actor_nodes[l] for l in k]
     for node in label_divi:
       sub_graph1=nx.ego_graph(B,node)
       G1.add_nodes_from(sub_graph1.nodes) # adding nodes
       G1.add_edges_from(sub_graph1.edges()) # adding edges
     cst1=cost_1(G1,clu)
     cost1=cost1+cst1 # calculating cost functions
     cst2=cost_2(G1,clu)
     cost2=cost2+cst2
  value=cost1*cost2
  cost_value[clu]=value
```

Fitting the clustering algorithm with the opimal number_of_clusters

```
optimal_cluster_number = max(cost_value, key=cost_value.get)
print("optimal cluster number = ",optimal_cluster_number) #https://www.geeksforgeeks.org/python-get-key-with-maximum-value-in-dict
```

```
optimal cluster number =  3
```

```
model= KMeans(n_clusters=optimal_cluster_number)
model.fit(actor_embeddings)
label=model.labels_
```

```
node_cluster={}
for i in range(optimal_cluster_number):
  k=[index for index, value in enumerate(label) if value == i]  # getting the cluster number for each node
  label_divi=[ actor_nodes[l] for l in k]
  for node in label_divi:
    node_cluster[node]=i
```

```
node_cluster['a1435']
```

```
2
```

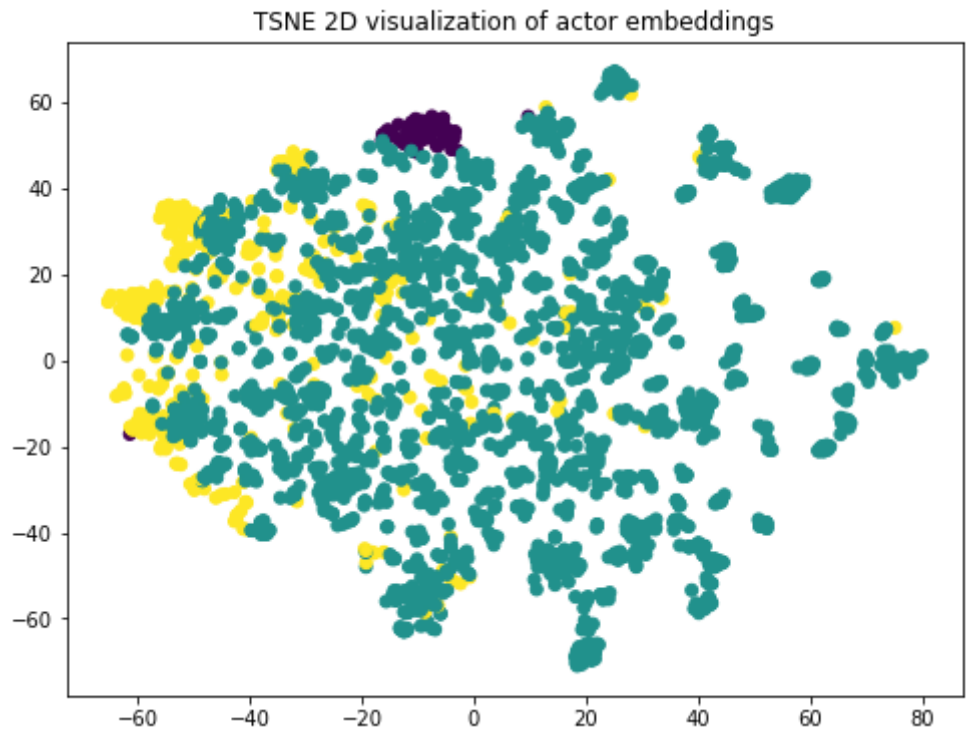
Displaying similar actor clusters

```
from sklearn.manifold import TSNE
transform = TSNE #PCA
trans = transform(n_components=2)
actor_embeddings_2d = trans.fit_transform(actor_embeddings)
```

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(8, 6)) #https://stackoverflow.com/questions/28227340/kmeans-scatter-plot-plot-different-colors-per-cluster
plt.scatter(actor_embeddings_2d[:,0], actor_embeddings_2d[:,1], c=model.labels_.astype(float))
plt.title('TSNE 2D visualization of actor embeddings')
```

```
Text(0.5, 1.0, 'TSNE 2D visualization of actor embeddings')
```



Task 2 : Apply clustering algorithm to group similar movies

```
def cost_2(graph,cluster):
  degree=graph.degree()
  degree_value=0 # to store sum of all  degree of actor nodes
  act_nodes=0  #to store sum of all unique actor nodes
  for i in degree:
    if "m" in i[0]:
      degree_value=degree_value+i[1]
    else:
      act_nodes=act_nodes+1
  value=((1/cluster)*(degree_value/act_nodes))
  return value
```

Grouping similar movies

```
cost_value={}
for clu in [3,5,10,50,200,300]:
  cost1=0
  cost2=0
  label=[]
  algo = KMeans(n_clusters=clu)
  algo.fit(movie_embeddings) # FITTING WITH KMEANS ALGO
  label=algo.labels_
  for i in range(clu):
     G1=nx.Graph()  # drawing graph for each cluster
     label_divi=[]
     k=[index for index, value in enumerate(label) if value == i]# accesing each cluster basesd on labels_
     label_divi=[ movie_nodes[l] for l in k]
     for node in label_divi:
       sub_graph1=nx.ego_graph(B,node)
       G1.add_nodes_from(sub_graph1.nodes) # adding nodes
       G1.add_edges_from(sub_graph1.edges()) # adding edges
     cst1=cost_1(G1,clu)
     cost1=cost1+cst1 # calculating cost functions
     cst2=cost_2(G1,clu)
     cost2=cost2+cst2
  value=cost1*cost2
  cost_value[clu]=value
```

Fitting the clustering algorithm with the opimal number_of_clusters

```
optimal_cluster_number = max(cost_value, key=cost_value.get)
print("optimal cluster number =",optimal_cluster_number) #https://www.geeksforgeeks.org/python-get-key-with-maximum-value-in-dicti
```

```
optimal cluster number = 3
```

```
model= KMeans(n_clusters=optimal_cluster_number)
```

```
model.fit(movie_embeddings)
label=model.labels_
```

```
node_cluster={}
for i in range(optimal_cluster_number):
  k=[index for index, value in enumerate(label) if value == i]
  label_divi=[ movie_nodes[l] for l in k]
  for node in label_divi:
    node_cluster[node]=i
```

```
node_cluster['m858']
```

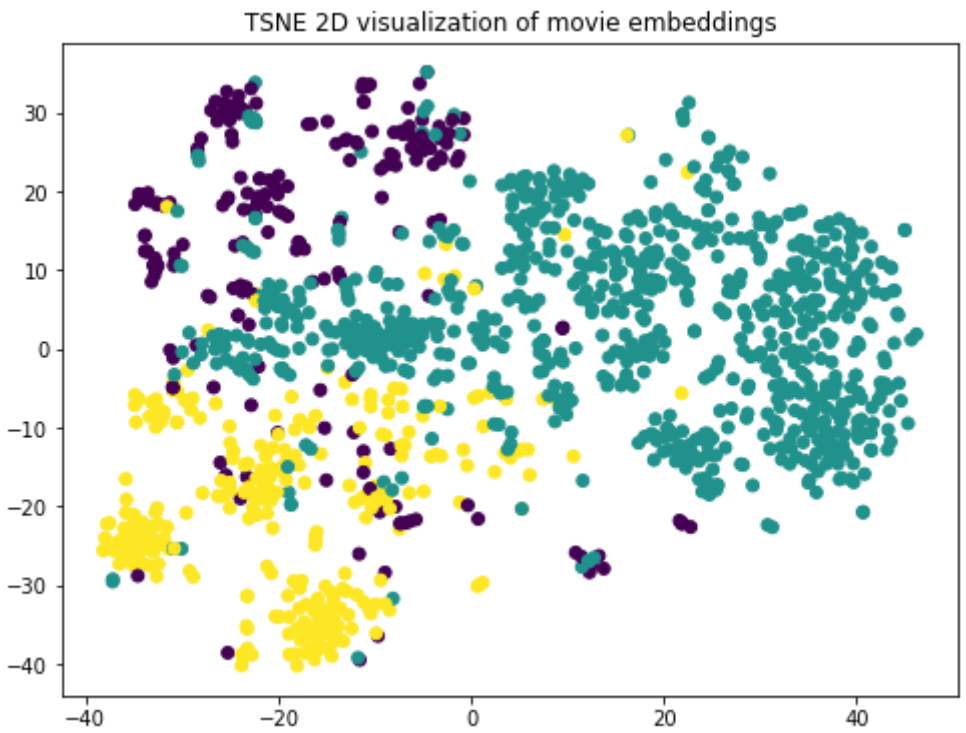
```
1
```

Displaying similar movie clusters

```
from sklearn.manifold import TSNE
transform = TSNE #PCA
trans = transform(n_components=2)
movie_embeddings_2d = trans.fit_transform(movie_embeddings)
```

```
import matplotlib.pyplot as plt
# Visualize it:
plt.figure(figsize=(8, 6))
plt.scatter(movie_embeddings_2d[:,0], movie_embeddings_2d[:,1], c=model.labels_.astype(float))
plt.title('TSNE 2D visualization of movie embeddings')
```

```
Text(0.5, 1.0, 'TSNE 2D visualization of movie embeddings')
```



1