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.

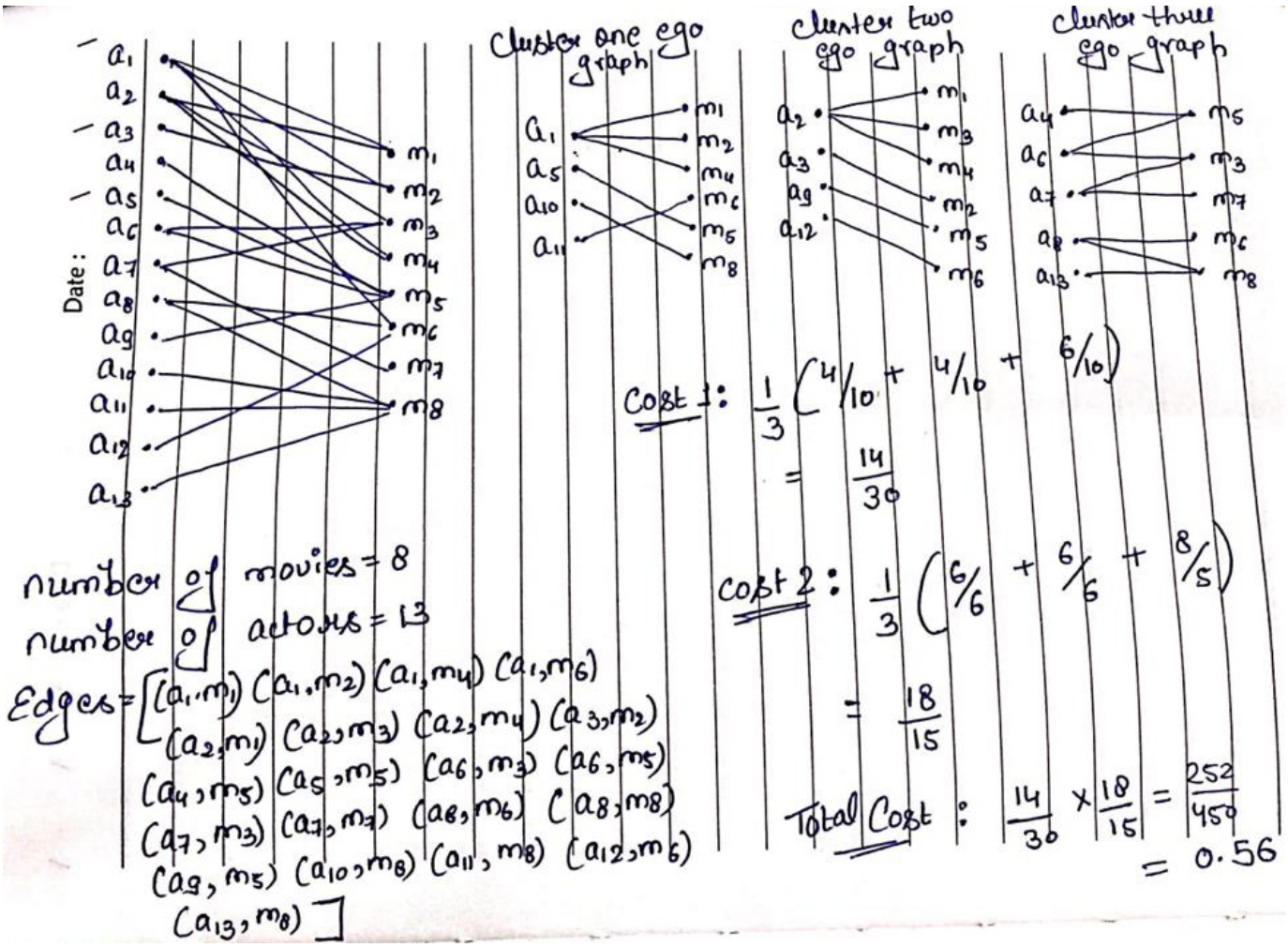
Please check clustering assignment helper functions notebook before attempting this assignment.

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

   $$\frac{1}{N}\sum_{\text{each cluster i}} \frac{(\text{number of nodes in the largest connected component in the graph with the movie nodes}}{(\text{total number of nodes in that cluster i})}$$

   where N= number of clusters

   (Write your code in def cost1())

4. Cost2 =

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

where N= number of clusters

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 i
    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 ar
        cost2=cost2(graph1)+cost2(graph2)+cost2(graph3)
    computer the metric Cost = Cost1*Cost2
return number_of_clusters which have maximum Cost
```

```
!pip install networkx==2.3
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Collecting networkx==2.3
  Downloading networkx-2.3.zip (1.7 MB)
     |████████████████████████████████| 1.7 MB 5.3 MB/s
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.7/dist-pack
Building wheels for collected packages: networkx
  Building wheel for networkx (setup.py) ... done
  Created wheel for networkx: filename=networkx-2.3-py2.py3-none-any.whl size=155600
  Stored in directory: /root/.cache/pip/wheels/44/e6/b8/4efaab31158e9e9ca9ed80b11f6b1
Successfully built networkx
Installing collected packages: networkx
  Attempting uninstall: networkx
    Found existing installation: networkx 2.6.3
    Uninstalling networkx-2.6.3:
      Successfully uninstalled networkx-2.6.3
Successfully installed networkx-2.3
```

```
pip install stellargraph
```

```
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: gensim>=3.4.0 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: matplotlib>=2.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.7/dist
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.7/dist-package
```

```
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/lo
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/di
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: protobuf<3.20,>=3.9.2 in /usr/local/lib/python3.7/d
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: libclang>=9.0.1 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/
Requirement already satisfied: keras<2.9,>=2.8.0rc0 in /usr/local/lib/python3.7/di
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: gast>=0.2.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dis
Requirement already satisfied: tensorflow-estimator<2.9,>=2.8 in /usr/local/lib/py
Requirement already satisfied: flatbuffers>=1.12 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python
Requirement already satisfied: tensorboard<2.9,>=2.8 in /usr/local/lib/python3.7/d
Requirement already satisfied: absl-py>=0.4.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.7/d
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/pyt
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/d
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.7
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/di
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/loc
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-pa
Installing collected packages: stellargraph
Successfully installed stellargraph-1.2.1
```

```
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



data=pd.read_csv('/content/drive/MyDrive/movie_actor_network.csv', index_col=False, names=


from google.colab import drive
drive.mount('/content/drive')
```

```
     Mounted at /content/drive
```

```
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 embeddin
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']
```

actor node movie node actor embedding, movie embedding for range len node_ids if m in node_target[i]

```
def data_split(node_ids,node_targets,node_embeddings):
    '''In this function, we will split the node embeddings into actor_embeddings , movie_e

    actor_nodes,movie_nodes=[],[]
    actor_embeddings,movie_embeddings=[],[]

    for i in range (len(node_ids)):
      if 'm' in node_targets[i]:

          # here i m appending the node_ids into created variable movie _nodes
        movie_nodes.append(node_ids[i])
           # here i m appending the node_embeddings into created variable movie_embeddings
        movie_embeddings.append(node_embeddings[i])

      else:
        # here i m appending the node_ids into created variable actor_nodes
        actor_nodes.append(node_ids[i])
  # here i m appending the node_embeddings into created variable actor_embeddings
        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


print(len(movie_nodes))  # checking the lenth of movie_nodes
```

```
1292
```

```
print(len(actor_nodes)) # checking the lenth of actor_nodes
```

```
3411
```

Grader function - 1

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

```
True
```

Grader function - 2

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

```
True
```

Calculating cost1

Cost1 =

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

where N= number of clusters

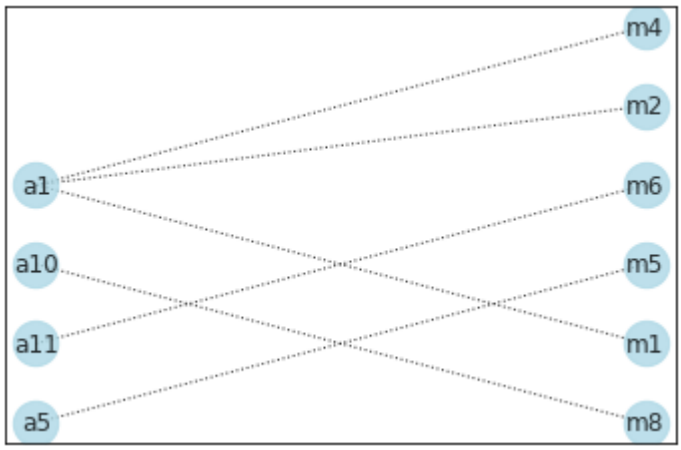
```
def cost_1(graph,cluster):
  # creating a variable gc grapg cluster
  Gc = max(nx.connected_component_subgraphs(graph), key=len)

  # here we are checking the number of nodes connected to gc
  connected=Gc.number_of_nodes()
  # here we are cheking the total number of nodes to graph
  total_nodes=graph.number_of_nodes()

  # here we are calculating the cost1 as per formula given above
  cost1=((1/cluster)*(connected/total_nodes))
  return cost1
```

```
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
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'),
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,
```



Grader function - 3

```
graded_cost1=cost_1(graded_graph,3)
def grader_cost1(data):
    assert(data==((1/3)*(4/10)))
    return True
grader_cost1(graded_cost1)
```

```
True
```

Calculating cost2

Cost2 =

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

where N= number of clusters

```
def cost_2(graph,cluster):

  # here we have computed the variable degree to compute the degree of graph
  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

      # here computing the cost 2 as per formula given above
```

```
  cost_2=((1/cluster)*(degree_value/mov_nodes))
  return cost_2
```

Grader function - 4

```
graded_cost2=cost_2(graded_graph,3)
def grader_cost2(data):
    assert(data==((1/3)*(6/6)))
    return True
grader_cost2(graded_cost2)
```

```
True
```

Grouping similar actors

```
from sklearn.cluster import KMeans
# importing kmeans algo rom sklearn



cost_value={}
for cluster in [3,5,10,50,200,300,500]:
  cost1=0
  cost2=0
  label=[]

 #  refer : https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
  algo = KMeans(n_clusters=cluster)
  algo.fit(actor_embeddings)              # FITTING WITH KMEANS ALGO
  label=algo.labels_


  for i in range(cluster):
     G1=nx.Graph()  # drawing graph for each cluster
     label_division=[]

     k=[index for index, value in enumerate(label) if value == i]# accesing each cluster b
     label_division=[ actor_nodes[l] for l in k]
     for node in label_division:
       sub_graph1=nx.ego_graph(B,node)

       # refer : https://networkx.org/documentation/stable/tutorial.html
       G1.add_nodes_from(sub_graph1.nodes) # here we are adding nodes to G1
       G1.add_edges_from(sub_graph1.edges())# here we are adding the edges to G1

         # here our cluster 1 is a group1 " g1" of cluster
     cluster1=cost_1(G1,cluster) #

      # here we are calculating the cost 1 and that is our define variable cost1 + cluster
     cost1=cost1+cluster1
      # calculating cost functions
```

```
# here also calculating the cluster 2 of group group 1
     cluster2=cost_2(G1,cluster)

     # calculating the cost2 and that is define variable cost2+cluster2
     cost2=cost2+cluster2

  # now we will calculate the value of cost1 and cost2
  value_of_cost1_cost2=cost1*cost2
  cost_value[cluster]=value_of_cost1_cost2



perfect_cluster_number = max(cost_value, key=cost_value.get)
print("perfect_cluster_number = ",perfect_cluster_number) #https://www.geeksforgeeks.org/p

     perfect_cluster_number =  5


# refer : https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

model= KMeans(n_clusters=perfect_cluster_number)
model.fit(actor_embeddings)
label=model.labels_


node_cluster={}

# taking range from variable perfect_cluster_number
for i in range(perfect_cluster_number):
  M=[index for index, value in enumerate(label) if value == i]  # getting the cluster numb
  label_division=[ actor_nodes[l] for l in M]
  # for each node in label_division
  for node in label_division:
    node_cluster[node]=i


node_cluster['a1621']

     1
```

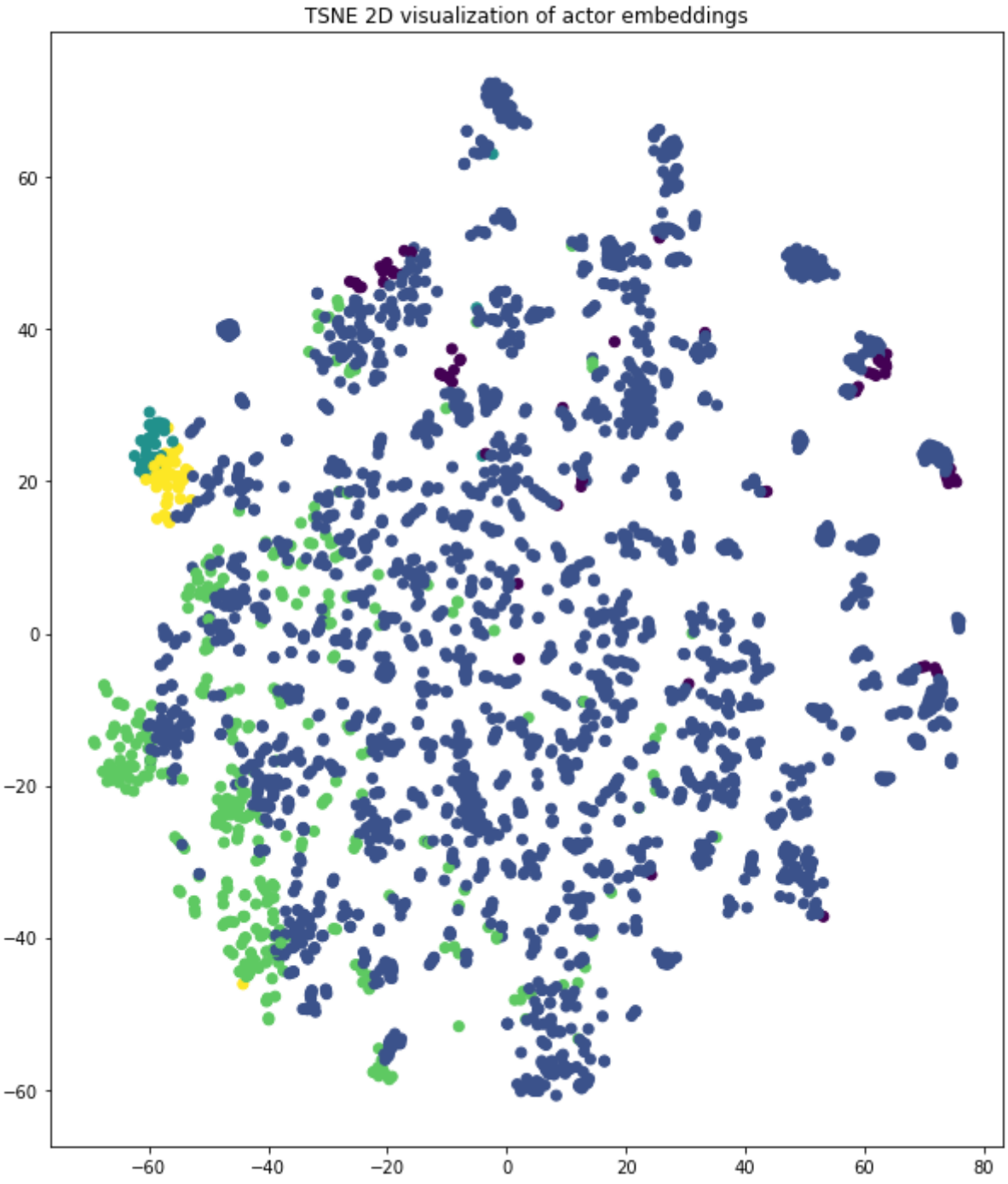
Displaying similar actor clusters

```
# refer : https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

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


import matplotlib.pyplot as plt
plt.figure(figsize=(10,12))             # refer :  https://stackoverflow.com/questions/282
plt.scatter(actor_embeddings_2d[:,0], actor_embeddings_2d[:,1], c=model.labels_.astype(flo
plt.title('TSNE 2D visualization of actor embeddings')
```

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



Grouping similar movies

```
def cost_2(graph,cluster):
  degree=graph.degree()

  # here we will store the sum of all degree of actor_nodes
  degree_value=0

   #to store sum of all unique actor nodes
  actor_nodes=0
  for i in degree:
   if "m" in i[0]:
      # here will add the m in degree value
      degree_value=degree_value+i[1]
```

```
      else:
         actor_nodes=actor_nodes+1


    value=((1/cluster)*(degree_value/actor_nodes))
    return value

  cost_value={}
  for cluster in [3,5,10,50,200,300]:
    cost1=0
    cost2=0
    label=[]

   #  refer : https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
    algo = KMeans(n_clusters=cluster)
    algo.fit(movie_embeddings)              # FITTING WITH KMEANS ALGO
    label=algo.labels_


    for i in range(cluster):
       G1=nx.Graph()  # drawing graph for each cluster
       label_division=[]

       k=[index for index, value in enumerate(label) if value == i]# accesing each cluster b
       label_division=[ movie_nodes[l] for l in k]
       for node in label_division:
         sub_graph1=nx.ego_graph(B,node)

         # refer : https://networkx.org/documentation/stable/tutorial.html
         G1.add_nodes_from(sub_graph1.nodes) # here we are adding nodes to G1
         G1.add_edges_from(sub_graph1.edges())# here we are adding the edges to G1

           # here our cluster 1 is a group1 " g1" of cluster
       cluster1=cost_1(G1,cluster) #

        # here we are calculating the cost 1 and that is our define variable cost1 + cluster
       cost1=cost1+cluster1
        # calculating cost functions

  # here also calculating the cluster 2 of group group 1
       cluster2=cost_2(G1,cluster)

       # calculating the cost2 and that is define variable cost2+cluster2
       cost2=cost2+cluster2

    # now we will calculate the value of cost1 and cost2
    value_of_cost1_cost2=cost1*cost2
    cost_value[cluster]=value_of_cost1_cost2


  perfect_cluster_number = max(cost_value, key=cost_value.get)
  print("perfect_cluster_number = ",perfect_cluster_number) #https://www.geeksforgeeks.org/p

       perfect_cluster_number =  3
```

```
# refer : https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

model= KMeans(n_clusters=perfect_cluster_number)
# fitting the movie data
model.fit(movie_embeddings)

# labeling the model
# refer : https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
label=model.labels_


node_cluster={} # created the empty dict

# taking range from variable perfect_cluster_number
for i in range(perfect_cluster_number):
  S=[index for index, value in enumerate(label) if value == i]  # getting the cluster numb
  label_division=[ movie_nodes[l] for l in S]
  for node in label_division:
    node_cluster[node]=i


node_cluster['m890']
```

```
1
```

Displaying similar movie clusters

```
# refer : https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

from sklearn.manifold import TSNE

transform = TSNE #PCA
trans_form = transform(n_components=2)
movie_embeddings_2d = trans_form.fit_transform(movie_embeddings)
```

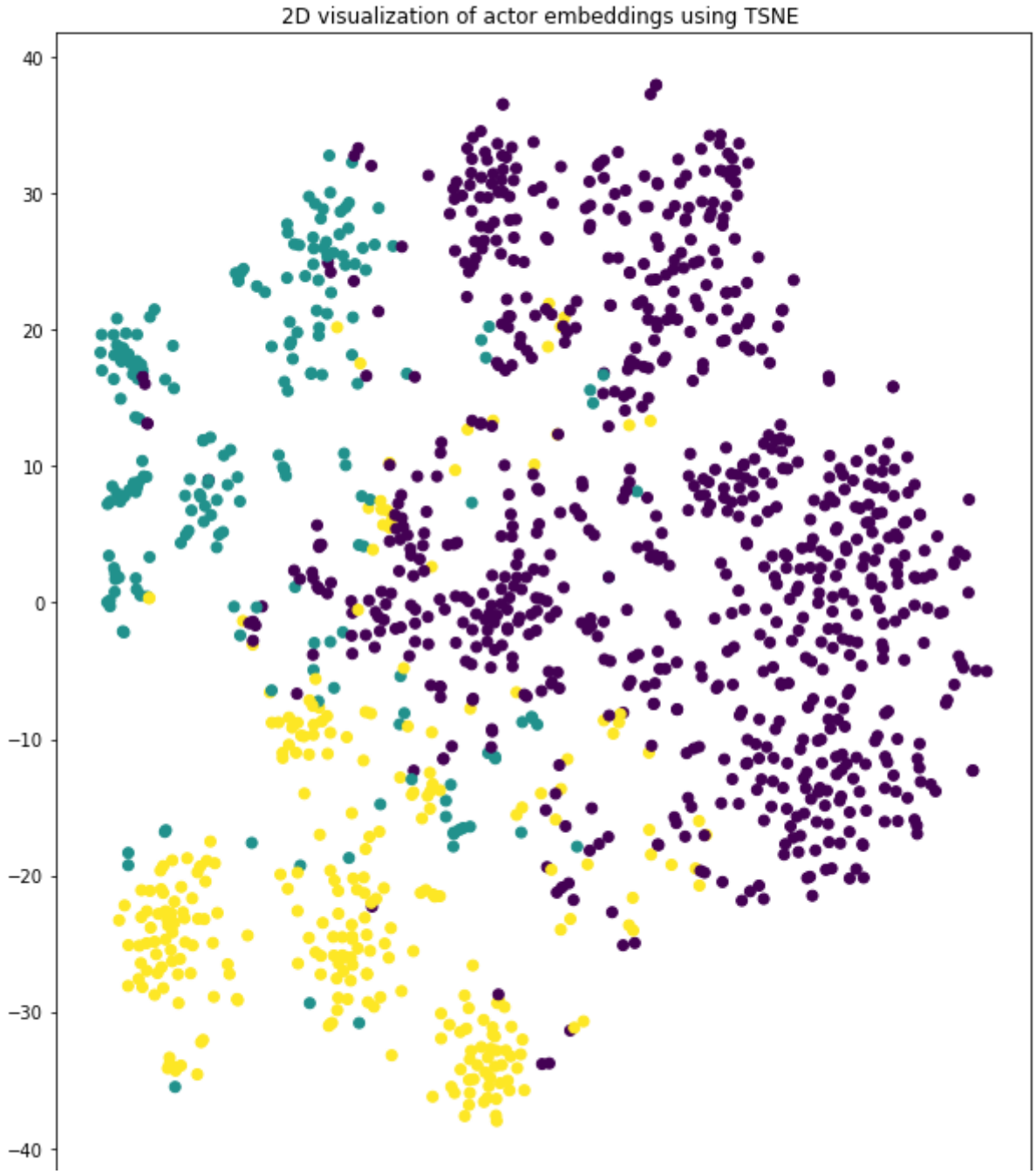
here i m using a tsne for visulization of clusters

```
#refer : #https://stackoverflow.com/questions/28227340/kmeans-scatter-plot-plot-different-

import matplotlib.pyplot as plt

plt.figure(figsize=(10,12))
plt.scatter(movie_embeddings_2d[:,0], movie_embeddings_2d[:,1], c=model.labels_.astype(flo
plt.title('2D visualization of actor embeddings using TSNE ')
```

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



Colab paid products - Cancel contracts here

✓ 0s completed at 8:45 AM