Clustering Assignment

New Section

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.

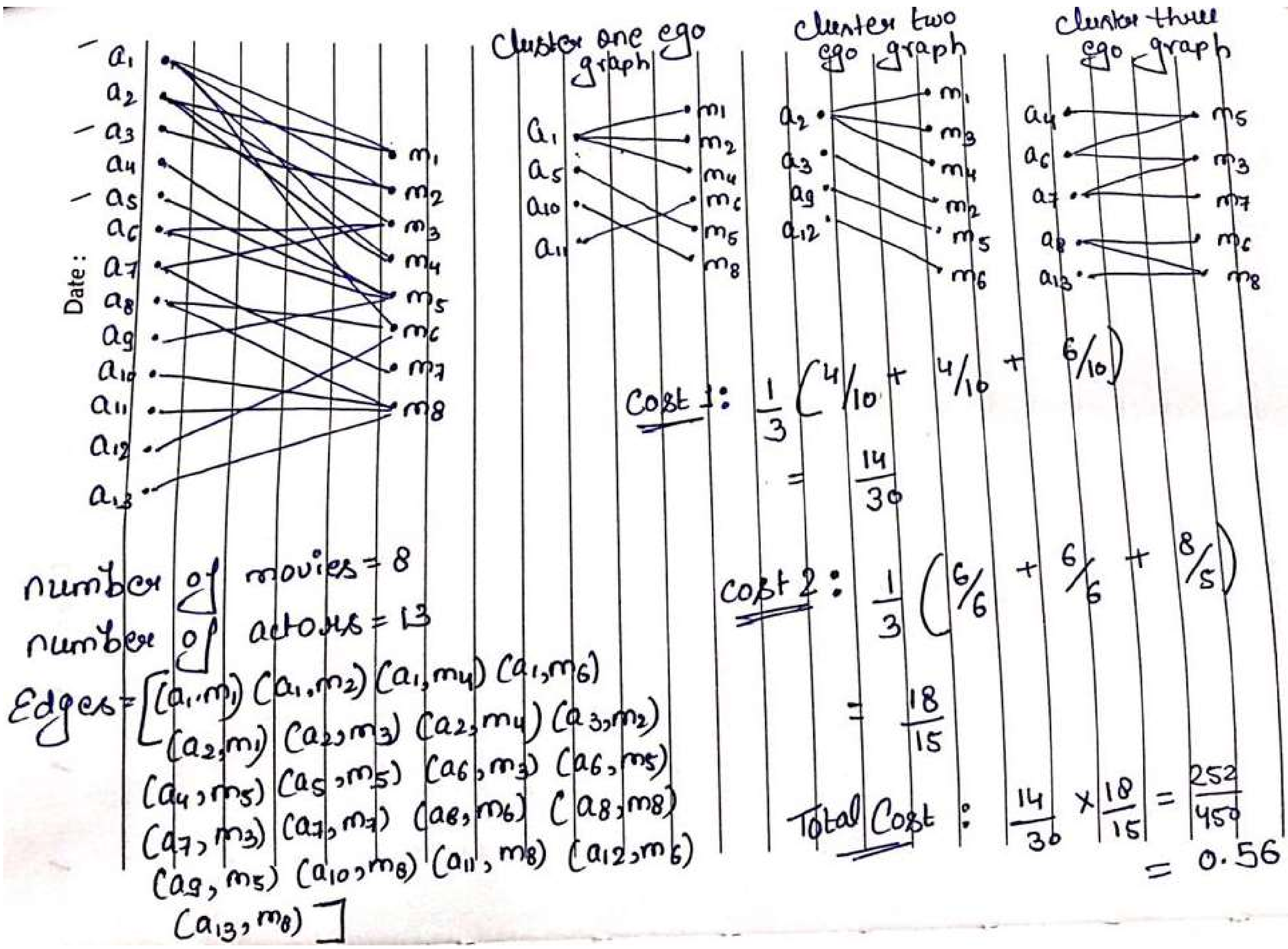
Please check clustering assignment helper functions notebook before attempting this assignment.

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

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

where N= number of clusters

(Write your code in def cost1())

4. Cost2 =

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

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

```
pip install stellargraph
```

```
  Downloading https://files.pythonhosted.org/packages/74/78/16b23ef04cf6fb24a/dea9fde
     |████████████████████████████████| 440kB 5.0MB/s
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: tensorflow>=2.1.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy>=1.14 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: matplotlib>=2.2 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: gensim>=3.4.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: h5py~=3.1.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: wheel~=0.35 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: google-pasta~=0.2 in /usr/local/lib/python3.7/dist-pac
```

```
Requirement already satisfied: keras-nightly~=2.5.0.dev in /usr/local/lib/python3.7/d
Requirement already satisfied: termcolor~=1.1.0 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: tensorflow-estimator<2.6.0,>=2.5.0rc0 in /usr/local/li
Requirement already satisfied: grpcio~=1.34.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: absl-py~=0.10 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: opt-einsum~=3.3.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: keras-preprocessing~=1.1.2 in /usr/local/lib/python3.7
Requirement already satisfied: flatbuffers~=1.12.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: gast==0.4.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: typing-extensions~=3.7.4 in /usr/local/lib/python3.7/d
Requirement already satisfied: wrapt~=1.12.1 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: six~=1.15.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: astunparse~=1.6.3 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: tensorboard~=2.5 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: decorator<5,>=4.3 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: cached-property; python_version < "3.8" in /usr/local/
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/pyt
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/lib/python3.7/dist
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/li
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /usr/loc
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3.6" in /usr/local/l
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dist
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: pyasn1>=0.1.3 in /usr/local/lib/python3.7/dist-package
Installing collected packages: stellargraph
Successfully installed stellargraph-1.2.1
```

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

```
Collecting networkx==2.3
  Downloading https://files.pythonhosted.org/packages/85/08/f20aef11d4c343b557e5de6b9548
     |████████████████████████████████| 1.8MB 5.0MB/s
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.7/dist-package
```

```
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=1556427 sh
  Stored in directory: /root/.cache/pip/wheels/de/63/64/3699be2a9d0ccdb37c7f16329acf3863
Successfully built networkx
ERROR: albumentations 0.1.12 has requirement imgaug<0.2.7,>=0.2.5, but you'll have imgau
Installing collected packages: networkx
  Found existing installation: networkx 2.5.1
    Uninstalling networkx-2.5.1:
      Successfully uninstalled networkx-2.5.1
Successfully installed networkx-2.3
```

```
import networkx as nx
from networkx.algorithms import bipartite
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
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('movie_actor_network (1).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')
```

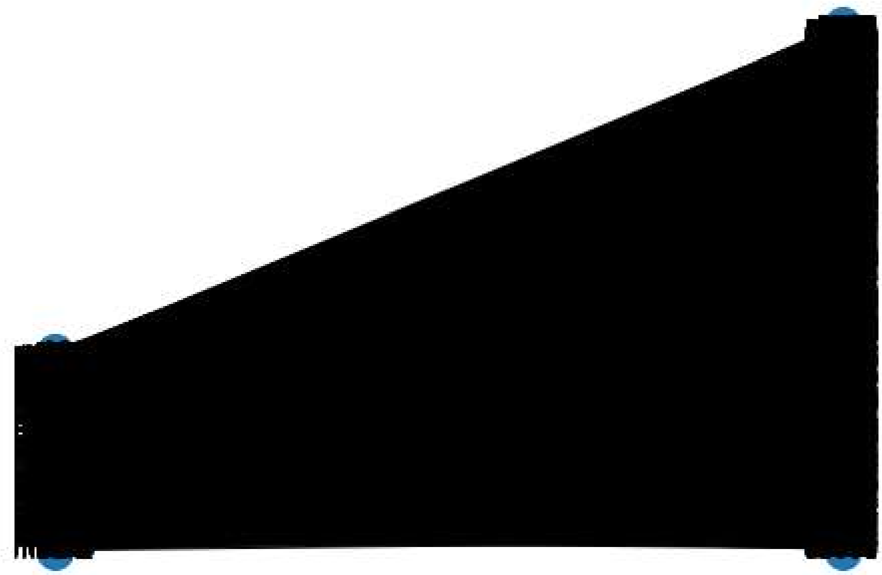
```
A = (B.subgraph(c) for c in nx.connected_components(B))
A = list(A)[0]
```

```
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
node_targets = [ A.nodes[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_embe
    actor_nodes,movie_nodes=[],[]
    actor_embeddings,movie_embeddings=[],[]
    # split the node_embeddings into actor_embeddings,movie_embeddings based on node_ids
    # By using node_embedding and node_targets, we can extract actor_embedding and movie embe
    # By using node_ids and node_targets, we can extract actor_nodes and movie nodes

    actor_embeddings = [x for i,x in enumerate(node_embeddings) if node_targets[i] == 'actor'
    actor_nodes = [x for i,x in enumerate(node_ids) if node_targets[i]=='actor']

    movie_embeddings = [x for i,x in enumerate(node_embeddings) if node_targets[i] == 'movie'
    movie_nodes = [x for i,x in enumerate(node_ids) if node_targets[i]=='movie']

    return actor_nodes,movie_nodes,actor_embeddings,movie_embeddings
```

Grader function - 1

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

```
def grader_actors(data):
    assert(len(data)==3411)
```

```
    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

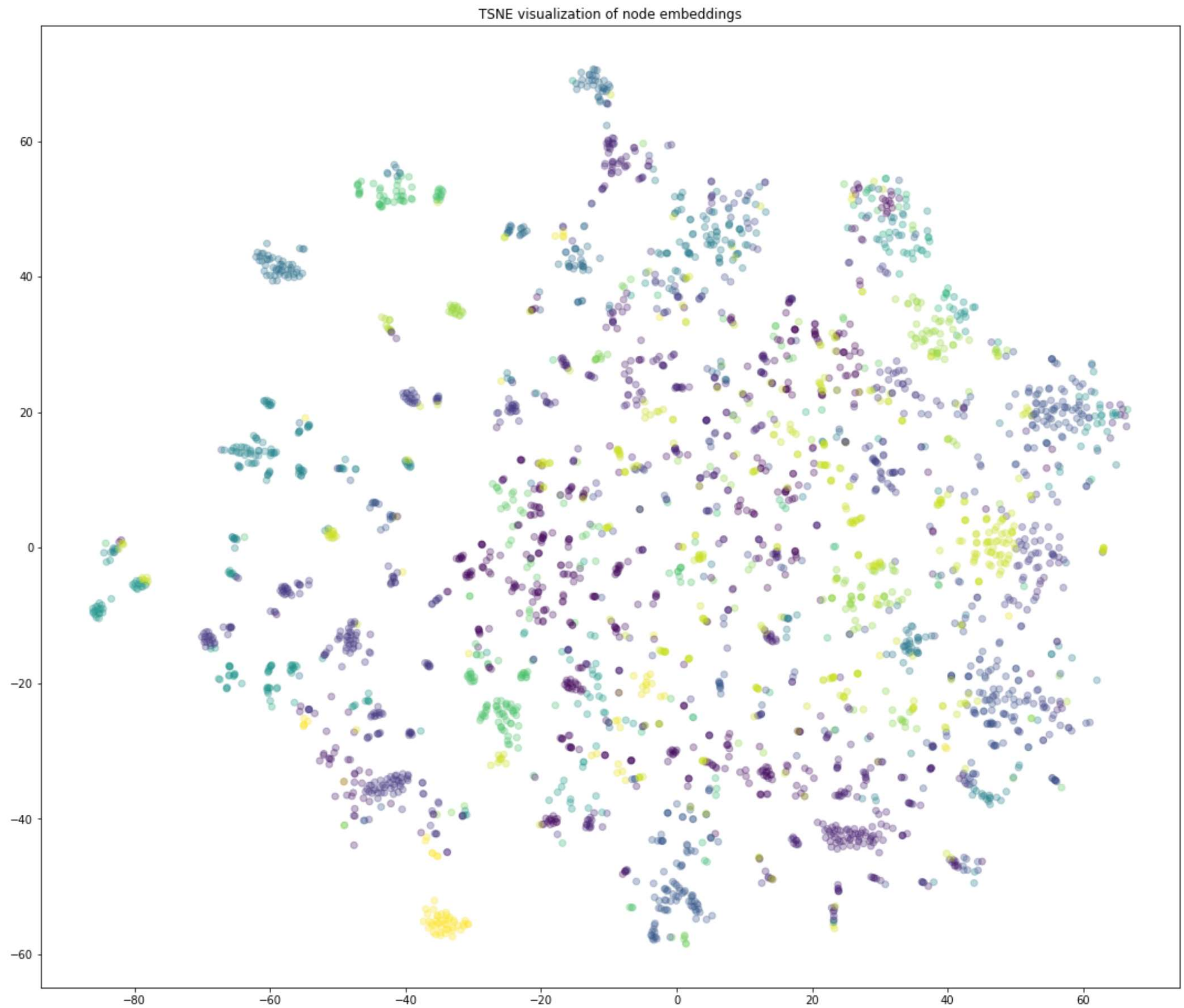
Cost1 =

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

where N= number of clusters

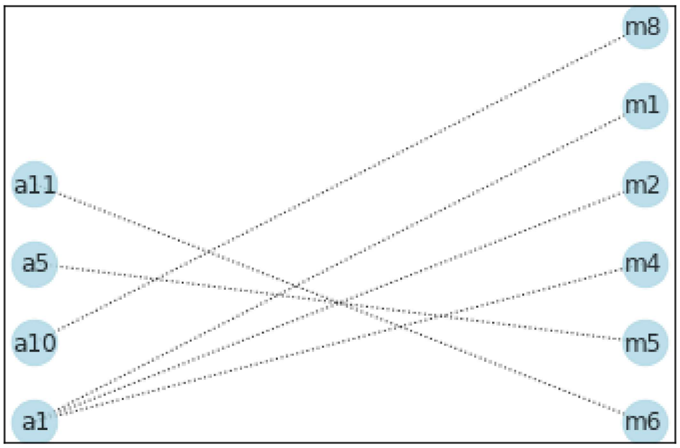
```
def cost1(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''
    cost1 = 0
    gg = max(nx.connected_component_subgraphs(graph),key=len)
    cost1 = cost1 + len(gg.nodes())/len(graph.nodes())

    cost1 = cost1/number_of_clusters


    return cost1
```

```
act_kmeans = KMeans(n_clusters=100, random_state=0).fit(actor_embeddings)
y = act_kmeans.predict(actor_embeddings)
transform = TSNE #PCA
trans = transform(n_components=2)
actor_embeddings_2d = trans.fit_transform(actor_embeddings)
plt.figure(figsize=(20,16))
plt.axes().set(aspect="equal")
plt.scatter(actor_embeddings_2d[:,0],actor_embeddings_2d[:,1],c=y, alpha=0.3)
plt.title('{} visualization of node embeddings'.format(transform.__name__))
plt.show()
```



Grader function - 3

```
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 "b
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'),('a
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,sty
```




```
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)
```

```
True
```

Calculating cost2

Cost2 =

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

where N= number of clusters

```
def cost2(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''
    cost2= 0  # calculate cost1
    act = []
    mov = []
    deg = 0
    for i in graph.node():
```

```
      if 'm' in i:
        mov.append(i)
      if 'a' in i:
        act.append(i)

    for i in act:
      deg = deg + graph.degree(i)

    cost2 = cost2 + deg/len(mov)
    cost2 = cost2/number_of_clusters

    return cost2
```

Grader function - 4

```
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)
```

```
True
```

```
cost_cluster =  []

for no_clusters in [3, 5, 10, 30, 50, 100, 200, 500]:
  cost_1 = 0
  cost_2 = 0
  kmeans = KMeans(n_clusters=no_clusters, random_state=0).fit(actor_embeddings)
  y = kmeans.predict(actor_embeddings)
  clusters=[[] for i in range(no_clusters)]
  for i,j in enumerate(y):
    clusters[j].append(actor_nodes[i])

  for cluster in clusters:
    edges = []
    m_set =set()
    deg = 0
    for i in cluster:
      for j in A[i]:
        m_set.add(j)
        edges.append((i,j))
    g = nx.Graph()
    g.add_nodes_from(cluster, bipartite=0, label='movie')
    g.add_nodes_from(m_set, bipartite=1, label='actor')
    g.add_edges_from(edges, label='acted')

    cost_1 = cost1(g,no_clusters)
    cost_2 = cost2(g,no_clusters)
```

```
  cost_cluster.append(cost_1*cost_2)


print(cost_cluster)
```

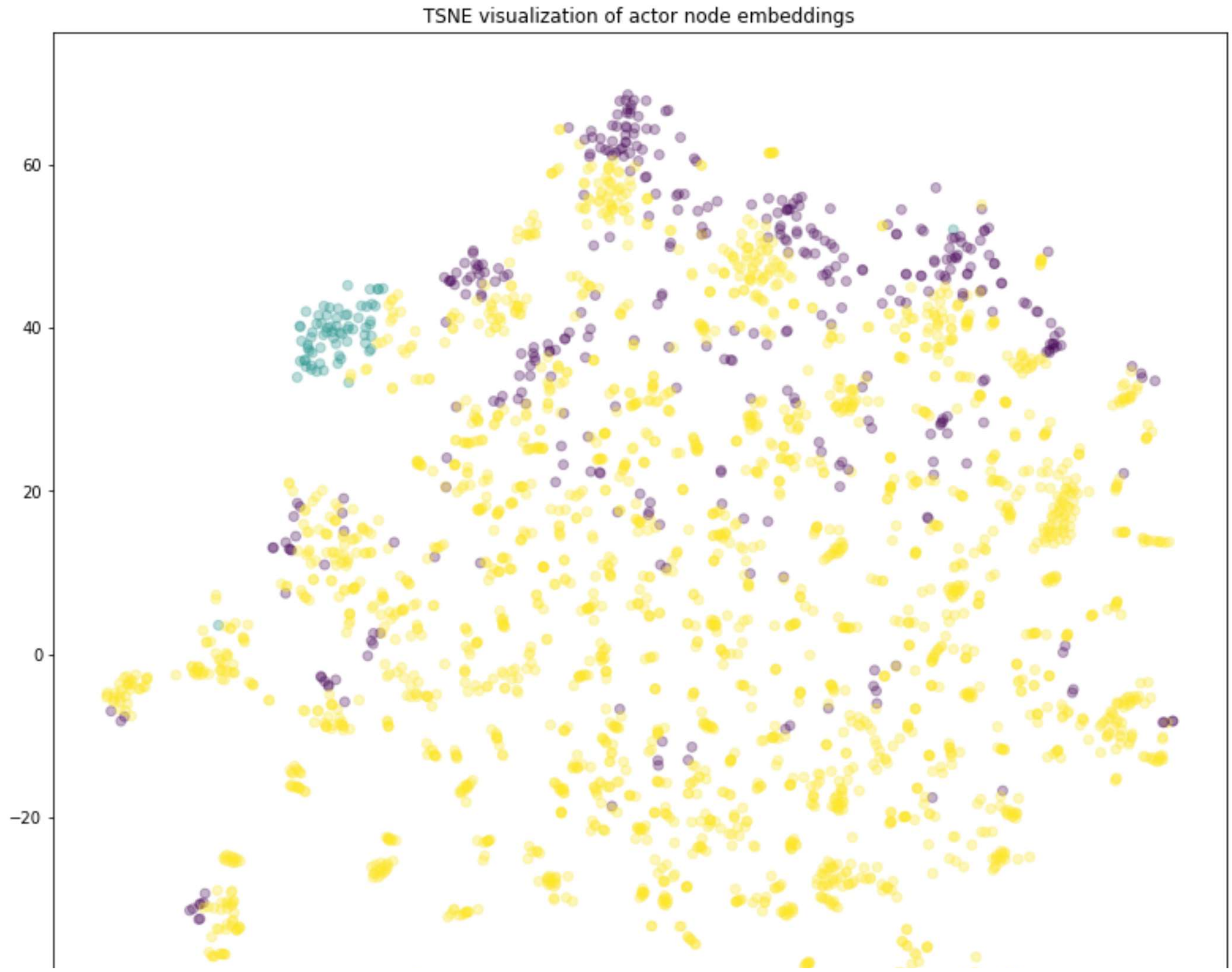
```
[0.4518519822879389, 0.0837878787878788, 0.025, 0.0003548772203240824, 0.0006351851851851
```

Grouping similar actors

```
act_kmeans = KMeans(n_clusters=3, random_state=0).fit(actor_embeddings)
y = act_kmeans.predict(actor_embeddings)
transform = TSNE #PCA
trans = transform(n_components=2)
actor_embeddings_2d = trans.fit_transform(actor_embeddings)
```

Displaying similar actor clusters

```
plt.figure(figsize=(20,16))
plt.axes().set(aspect="equal")
plt.scatter(actor_embeddings_2d[:,0],actor_embeddings_2d[:,1],c=y, alpha=0.3)
plt.title('{} visualization of actor node embeddings'.format(transform.__name__))
plt.show()
```



Grouping similar movies


```
cost_cluster =  []

for no_clusters in [3, 5, 10, 30, 50, 100, 200, 500]:
  cost_1 = 0
  cost_2 = 0
  kmeans = KMeans(n_clusters=no_clusters, random_state=0).fit(movie_embeddings)
  y = kmeans.predict(movie_embeddings)
  clusters=[[] for i in range(no_clusters)]
  for i,j in enumerate(y):
    clusters[j].append(movie_nodes[i])

  for cluster in clusters:
    edges = []
    m_set =set()
    deg = 0
    for i in cluster:
      for j in A[i]:
        m_set.add(j)
        edges.append((i,j))
    g = nx.Graph()
```

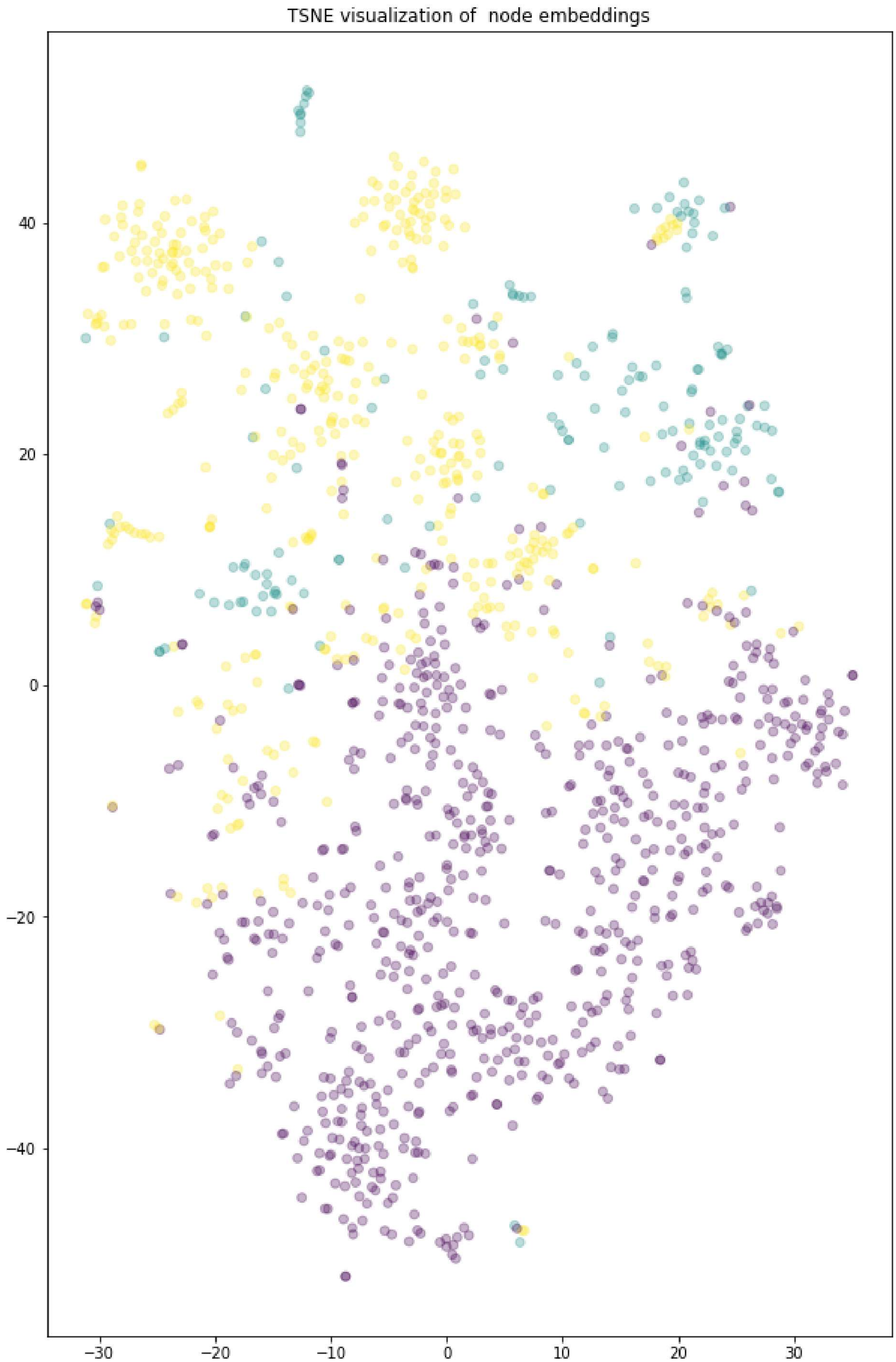
```
    g.add_nodes_from(cluster, bipartite=0, label='movie')
    g.add_nodes_from(m_set, bipartite=1, label='actor')
    g.add_edges_from(edges, label='acted')

    cost_1 = cost1(g,no_clusters)
    cost_2 = cost2(g,no_clusters)

  cost_cluster.append(cost_1*cost_2)
```

```
print(cost_cluster)
```

```
[0.7495423291996348, 0.5081690140845071, 0.08865979381443301, 0.02, 0.00196405797101449
```

Displaying similar movie clusters

```
act_kmeans = KMeans(n_clusters=3, random_state=0).fit(movie_embeddings)
y = act_kmeans.predict(movie_embeddings)
transform = TSNE #PCA
trans = transform(n_components=2)
movie_embeddings_2d = trans.fit_transform(movie_embeddings)
plt.figure(figsize=(20,16))
plt.axes().set(aspect="equal")
plt.scatter(movie_embeddings_2d[:,0],movie_embeddings_2d[:,1],c=y, alpha=0.3)
plt.title('{} visualization of  node embeddings'.format(transform.__name__))
plt.show()
```



✓ 14s completed at 13:12