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())

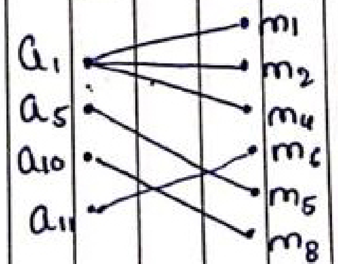
Task 1 : Apply clustering algorithm to group similar actors

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())

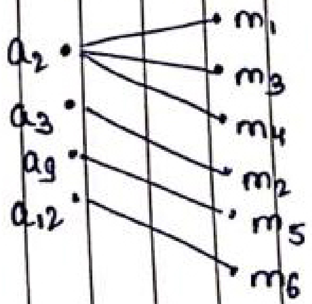
6. Fit the clustering algorithm with the opimal number_of_clusters and get the cluster number for each node
7. Convert the d-dimensional dense vectors of nodes into 2-dimensional using dimensionality reduction techniques (preferably TSNE)
8. Plot the 2d scatter plot, with the node vectors after step e and give colors to nodes such that same cluster nodes will have same color

Date :

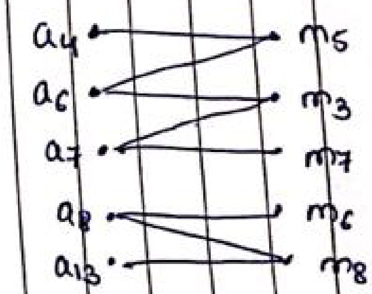
Cluster one ego graph



cluster two ego graph



cluster three ego graph



Cost 1 : $\frac{1}{3}\left(\frac{4}{10} + \frac{4}{10} + \frac{6}{10}\right)$

$= \frac{14}{30}$

Cost 2 : $\frac{1}{3}\left(\frac{6}{6} + \frac{6}{6} + \frac{8}{5}\right)$

$= \frac{18}{15}$

Total Cost : $\frac{14}{30} \times \frac{18}{15} = \frac{252}{450}$

$= 0.56$

number of movies = 8

number of actors = 13

Edges = [(a_1, m_1) (a_1, m_2) (a_1, m_4) (a_1, m_6)
(a_2, m_1) (a_2, m_3) (a_2, m_4) (a_3, m_2)
(a_4, m_5) (a_5, m_5) (a_6, m_3) (a_6, m_5)
(a_7, m_3) (a_7, m_7) (a_8, m_6) (a_8, m_8)
(a_9, m_5) (a_{10}, m_8) (a_{11}, m_8) (a_{12}, m_6)
(a_{13}, m_8)]

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())

3. 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
```

INSTALL REQUIRED LIBRARIES:-

In [1]:

```
!pip3 install networkx==2.3
```

```
Collecting networkx==2.3
  Downloading https://files.pythonhosted.org/packages/85/08/f20aef11d4c343b557e5de6b9548761811eb16e438cee3d32b1c66c85
66b/networkx-2.3.zip (1.7MB)
     |████████████████████████████████| 1.8MB 16.1MB/s
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.7/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-any.whl size=1556408 sha256=fa62dc8a02a71baf22e3dfc3
845a34f60cba3bf475f89c762837862a7ae81380
  Stored in directory: /root/.cache/pip/wheels/de/63/64/3699be2a9d0ccdb37c7f16329acf3863fd76eda58c39c737af
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 incompatibl
e.
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
```

In [2]:

```
!pip3 install stellargraph
```

```
Collecting stellargraph
  Downloading https://files.pythonhosted.org/packages/74/78/16b23ef04cf6fb24a7dea9fd0e03c8308a56681cc5efe29f16186210b
a04/stellargraph-1.2.1-py3-none-any.whl (435kB)
     |████████████████████████████████| 440kB 18.9MB/s
Requirement already satisfied: numpy>=1.14 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (1.19.5)
Requirement already satisfied: matplotlib>=2.2 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (3.2.2)
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (0.2
2.2.post1)
Requirement already satisfied: gensim>=3.4.0 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (3.6.0)
Requirement already satisfied: tensorflow>=2.1.0 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (2.4.
1)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (1.1.5)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (1.4.1)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (2.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.2->ste
llargraph) (1.3.1)
```

```
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.2->stellarg
raph) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.2->
stellargraph) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (fr
om matplotlib>=2.2->stellargraph) (2.4.7)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20->stell
argraph) (1.0.1)
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-packages (from gensim>=3.4.0->stell
argraph) (5.0.0)
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.7/dist-packages (from gensim>=3.4.0->stellargrap
h) (1.15.0)
Requirement already satisfied: keras-preprocessing~=1.1.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow>
=2.1.0->stellargraph) (1.1.2)
Requirement already satisfied: tensorboard~=2.4 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->st
ellargraph) (2.4.1)
Requirement already satisfied: wheel~=0.35 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellar
graph) (0.36.2)
Requirement already satisfied: opt-einsum~=3.3.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->s
tellargraph) (3.3.0)
Requirement already satisfied: flatbuffers~=1.12.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0-
>stellargraph) (1.12)
Requirement already satisfied: gast==0.3.3 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellar
graph) (0.3.3)
Requirement already satisfied: google-pasta~=0.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->s
tellargraph) (0.2.0)
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->ste
llargraph) (3.12.4)
Requirement already satisfied: grpcio~=1.32.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stel
largraph) (1.32.0)
Requirement already satisfied: absl-py~=0.10 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stell
argraph) (0.12.0)
Requirement already satisfied: astunparse~=1.6.3 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->s
tellargraph) (1.6.3)
Requirement already satisfied: termcolor~=1.1.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->st
ellargraph) (1.1.0)
Requirement already satisfied: tensorflow-estimator<2.5.0,>=2.4.0 in /usr/local/lib/python3.7/dist-packages (from ten
sorflow>=2.1.0->stellargraph) (2.4.0)
Requirement already satisfied: wrapt~=1.12.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stell
argraph) (1.12.1)
Requirement already satisfied: typing-extensions~=3.7.4 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=
2.1.0->stellargraph) (3.7.4.3)
Requirement already satisfied: h5py~=2.10.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stella
rgraph) (2.10.0)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24->stellargrap
```

```
h) (2018.9)
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.7/dist-packages (from networkx>=2.2->stella
rgraph) (4.4.2)
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.4->t
ensorflow>=2.1.0->stellargraph) (54.2.0)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.7/dist-packages (from tenso
rboard~=2.4->tensorflow>=2.1.0->stellargraph) (0.4.4)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.4->
tensorflow>=2.1.0->stellargraph) (2.23.0)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorbo
ard~=2.4->tensorflow>=2.1.0->stellargraph) (1.8.0)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.4->te
nsorflow>=2.1.0->stellargraph) (1.0.1)
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.4
->tensorflow>=2.1.0->stellargraph) (1.28.1)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.4->tens
orflow>=2.1.0->stellargraph) (3.3.4)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/dist-packages (from google-auth-o
authlib<0.5,>=0.4.1->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (1.3.0)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (fro
m requests<3,>=2.21.0->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0-
>tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0
->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (2020.12.5)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tens
orboard~=2.4->tensorflow>=2.1.0->stellargraph) (2.10)
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3.6" in /usr/local/lib/python3.7/dist-packages (from
google-auth<2,>=1.6.3->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (4.7.2)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from google-auth<2,>
=1.6.3->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (4.2.1)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from google-auth<2,>=
1.6.3->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (0.2.8)
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /usr/local/lib/python3.7/dist-packages
(from markdown>=2.6.8->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (3.10.1)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages (from requests-oauthlib>=0.
7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (3.1.0)
Requirement already satisfied: pyasn1>=0.1.3 in /usr/local/lib/python3.7/dist-packages (from rsa<5,>=3.1.4; python_ve
rsion >= "3.6"->google-auth<2,>=1.6.3->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (0.4.8)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata; python_v
ersion < "3.8"->markdown>=2.6.8->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (3.4.1)
Installing collected packages: stellargraph
Successfully installed stellargraph-1.2.1
```

IMPORT NECESSARY LIBRARIES:-

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

READ THE DATA AND CREATE A BIPARTITE-GRAPH :

In [4]:

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

In [5]:

```
# find edges(as tuple) between the nodes from the given data:-
edges = [tuple(x) for x in data.values.tolist()]
```

In [6]:

```
B = nx.Graph() #--> Create an empty graph

# add nodes & edges to the graph B from the data:-
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')
```

OBTAIN THE LARGEST CONNECTED COMPONENT OF GRAPH (B) :-

In [7]:

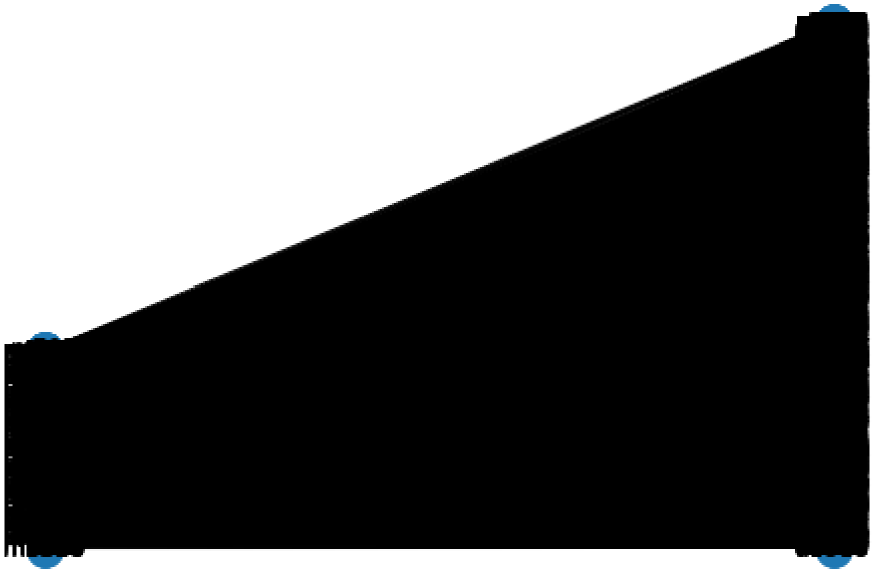
```
# Get the largest connected component from the subgraph:-
A = list(nx.connected_component_subgraphs(B))[0]
```

In [8]:
```
print("number of nodes", A.number_of_nodes())
print("number of edges", A.number_of_edges())
```

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

PLOT THE CONNECTED COMPONENT GRAPH USING nx.draw() :-

In [ ]:
```
l,r = nx.bipartite.sets(A) #--> We get the two sets of bipartite graph
pos = {}

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

# draw the graph:-
nx.draw(A,pos=pos, with_labels=True)
plt.show()
```



FIND THE NUMBER OF MOVIE & ACTOR NODES :

In [9]:

```
movies = [] # create list to store movie nodes
actors = [] # create list to store actor nodes
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 A RANDOM METAPATH WALK :

In [10]:

```
# 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
```

FIND NODE VECTORS USING WORD2VEC() :-

In [11]:

```
from gensim.models import Word2Vec   # import word2vec from gensim models
'''obtain word vectors for each distinct node id's ,
```

```
provided that those nodes must have explored in the path atleast 4walks'''
model = Word2Vec(walks,size=128,min_count=4,window = 5)
```

In [12]:
```
model.wv.vectors.shape  # 128-dimensional vector for each node in the graph
```

Out[12]: (4703, 128)

In [13]:
```
# 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']
```

OBTAIN THE NODE EMBEDDINGS FOR BOTH ACTOR & MOVIE NODE SEPERATELY - - >

In [14]:
```
def data_split(node_ids,node_targets,node_embeddings):
    '''In this function, we will split the node embeddings into 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 embedding
    # By using node_ids and node_targets, we can extract actor_nodes and movie nodes
    actor_nodes,movie_nodes=[],[]
    actor_embeddings,movie_embeddings=[],[]
    for j in range(0,4703):
        if node_targets[j] == 'actor':
            actor_nodes.append(node_ids[j])
            actor_embeddings.append(node_embeddings[j])
        else:
```

```
            movie_nodes.append(node_ids[j])
            movie_embeddings.append(node_embeddings[j])

    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)
```

PERFORM SANITY CHECKS:-

In [15]:

```
# Sanity checks
print(actor_nodes[0:5])
print(movie_nodes[0:5])
print(actor_embeddings[0].shape)
print(movie_embeddings[0].shape)
```

```
['a973', 'a967', 'a964', 'a1731', 'a970']
['m1094', 'm1111', 'm67', 'm1100', 'm1095']
(128,)
(128,)
```

Grader function - 1

In [16]:

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

Out[16]:

```
True
```

Grader function - 2

In [17]:

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

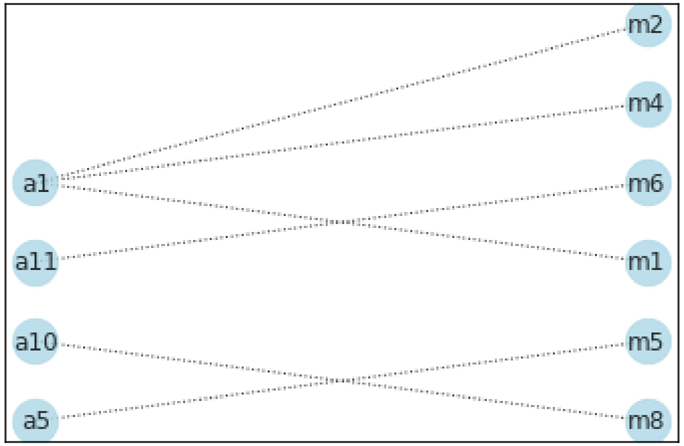
Out[17]:

```
True
```

CREATE A TOY-GRAPH TO GRADE CUSTOM COST FUNCTIONS:-

In [18]:

```
# Necessary libraries:-
import networkx as nx
from networkx.algorithms import bipartite

# create an empty graph & add nodes manually:

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))

# Plot the graph
nx.draw_networkx(graded_graph, pos=pos, with_labels=True,
                 node_color='lightblue',alpha=0.8,style='dotted',node_size=500)
```



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

DEFINE COST1 FUNCTION FOR NODE EMBEDDINGS:-

In [19]:
```
def cost1(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''
    n = len(list(max(nx.connected_component_subgraphs(graph),key=len)))
    t = graph.number_of_nodes()
    cost1 = (1/(number_of_clusters)) * (n/t)
    return cost1
```

Grader function - 3

In [20]:
```
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[20]:
```
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

DEFINE COST2 FUNCTION FOR ACTOR EMBEDDINGS:-

In [21]:
```
# reference:-https://networkx.org/documentation/stable/reference/algorithms/bipartite.html
def cost2(graph,number_of_clusters):
    '''In this function, we will calculate cost2'''
    movies = []     # To store movie nodes
    actors = []     # To store actor nodes
    for i in graph.nodes():
```

```
        if 'm' in i:
            movies.append(i)
        if 'a' in i:
            actors.append(i)
    lst = []       # To store number of degrees of each actor node.
    for j in list(set(actors)):
        lst.append(graph.degree(j))
    n = sum(lst)   # we get the numerator part of cost-2 formula.
    d = len(list(set(movies)))     # we get the denominator part of cost-2 formula.

    #apply the formula
    cost2 = (1/(number_of_clusters)) * (n/d)

    return cost2
```

Grader function - 4

In [22]:

```
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[22]: True

Grouping similar actors

In [23]:

```
# Define the number of clusters
no_of_clusters = [3,5,10,30,50,100,200,500]

# Run the kmeans algorithm with different number of clusters
cluster_labels = []
for i in no_of_clusters:
    algo = KMeans(n_clusters=i,random_state=0).fit(actor_embeddings)
    cluster_labels.append(algo.labels_)

list_of_all_clusters_for_diff_values_of_k = []

# assign the nodes with respective cluster labels for diff number of clusters:
for j in range(len(no_of_clusters)):
```

```
    list_of_all_clusters = [] # To store all the nodes with respect to unique cluster label
    unique = np.unique(cluster_labels[j]) # get the unique cluster labels

    # zip the nodes with their respective cluster labels:-
    dict_of_actor_nodes = dict(zip(actor_nodes,cluster_labels[j]))

    for n in unique:     # for each unique cluster label
        clust = []
        for i,j in dict_of_actor_nodes.items():
            if j == n:
                clust.append(i)
        list_of_all_clusters.append(clust)

    # append each group of cluster sets for different number of clusters(k):-
    list_of_all_clusters_for_diff_values_of_k.append(list_of_all_clusters)

print(len(list_of_all_clusters_for_diff_values_of_k)) # should be 8
```

```
8
```

OBTAIN OPTIMAL NUMBER OF CLUSTERS BASED ON COSTS:-

In [24]:

```
from networkx.algorithms import bipartite # import necessary library

no_of_clusters = [3,5,10,30,50,100,200,500] # define number_of_clusters

final_cost_on_k_cluster_sets = [] # to store final cost for diff number of clusters

for cluster_sets in list_of_all_clusters_for_diff_values_of_k:

    cost_1,cost_2 = 0,0       # initiate the cost2 to zero

    for groups in cluster_sets:

        G = nx.Graph()  # this is a empty cluster-graph which may be disconnected

        for a in groups:
            # create subgraphs using ego_graph()
            subgraph_object = nx.ego_graph(B,a)
```

```
            # add nodes,edges to the main cluster-graph(G) using subgraph object
            G.add_nodes_from(subgraph_object.nodes())
            G.add_edges_from(subgraph_object.edges())

        # Update the costs for each group of clusters:-
        cost_1 += cost1(G,len(cluster_sets))
        cost_2 += cost2(G,len(cluster_sets))

    # obtain the finalcost on the cluster_sets for different number of clusters(k)
    finalcost = (cost_1*cost_2)

    final_cost_on_k_cluster_sets.append(finalcost)

print("The finalcost values for different number of clusters : \n",
      final_cost_on_k_cluster_sets)

max_cost = max(final_cost_on_k_cluster_sets)

idx = final_cost_on_k_cluster_sets.index(max_cost)

print("The optimal number of clusters with max cost : ",no_of_clusters[idx])
```

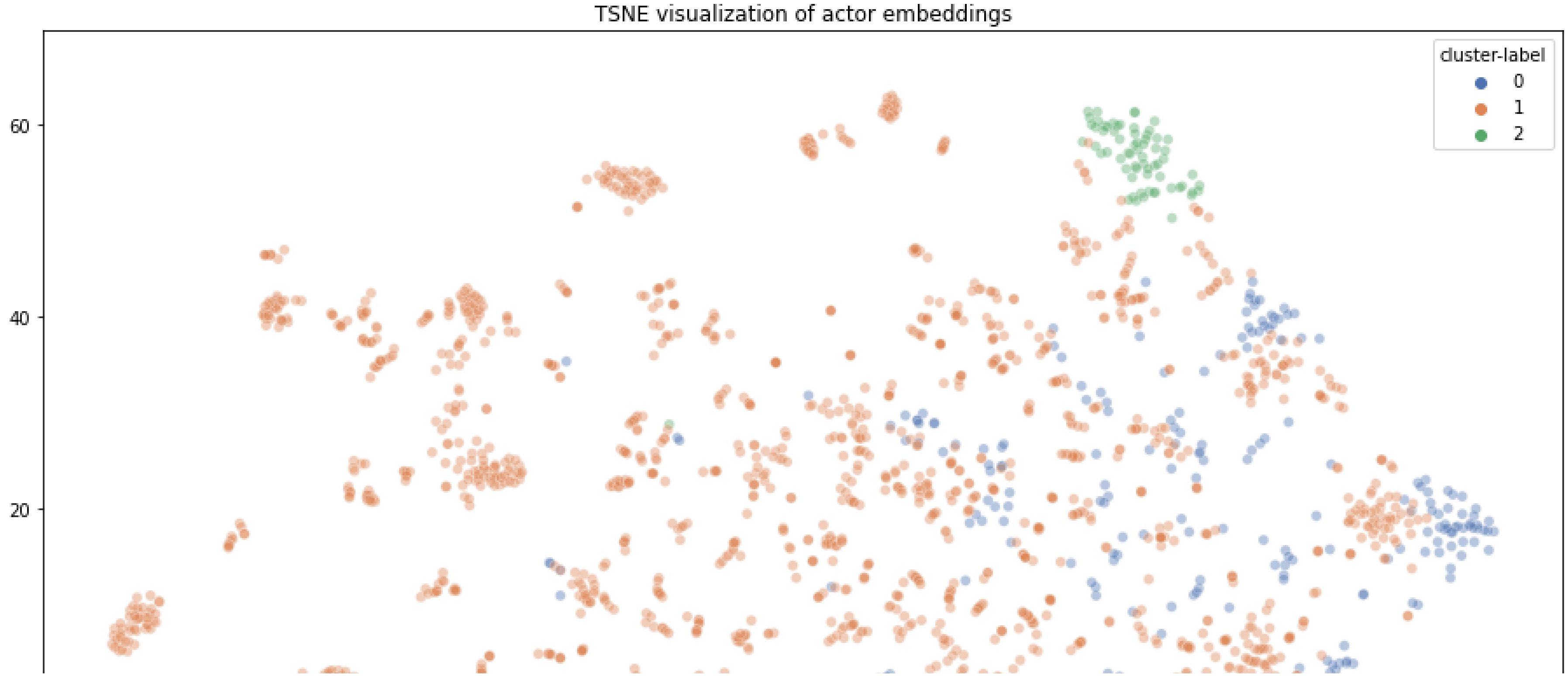
```
The finalcost values for different number of clusters :
 [3.7002041060442887, 3.1261159381321515, 2.247373963750886, 1.8028036161403806, 1.563847699889756, 1.674008450286440
5, 1.7264239256828882, 1.8629586024854314]
The optimal number of clusters with max cost :  3
```

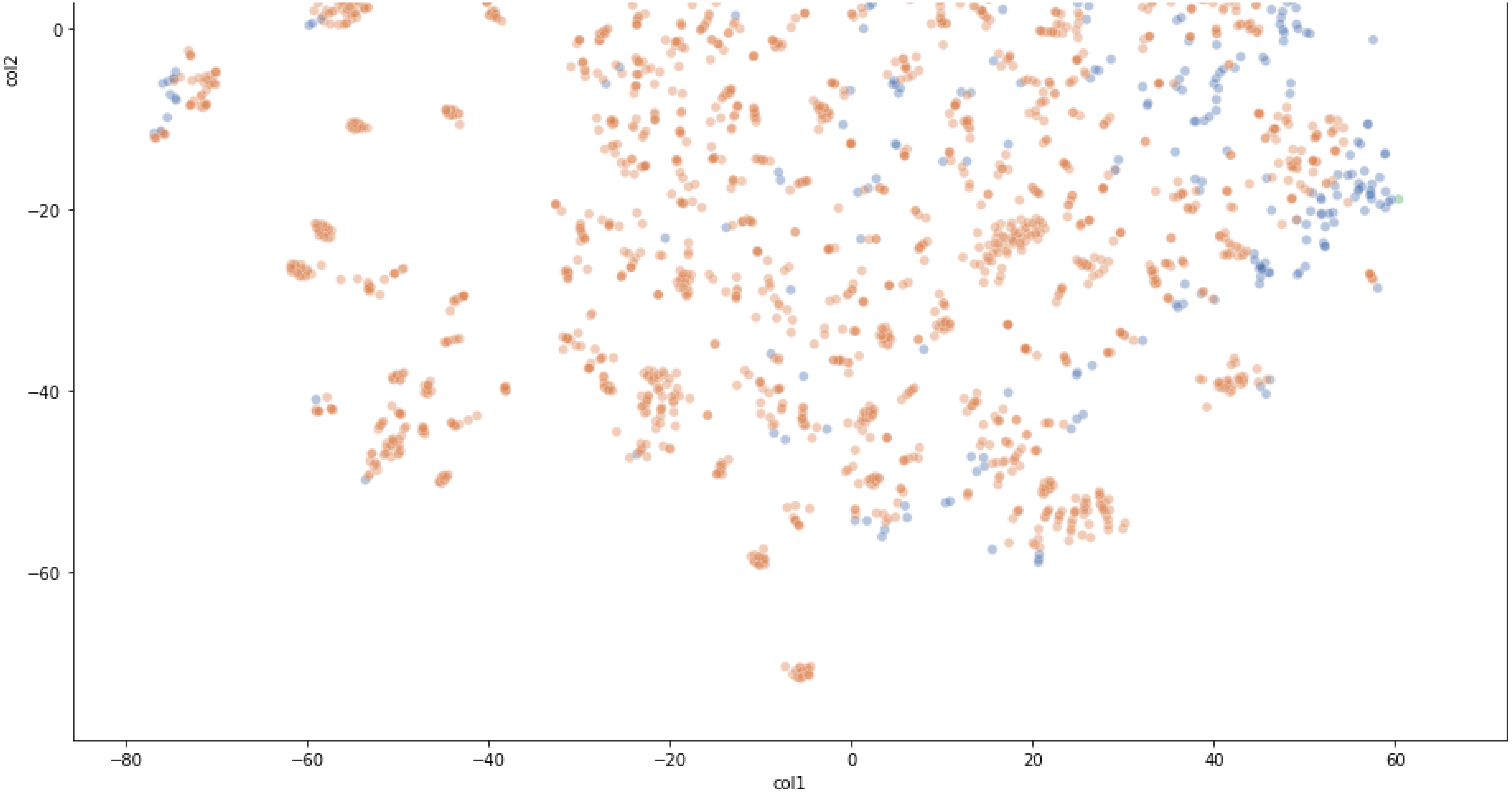
Displaying similar actor clusters

In [25]:

```
#with optimal number of clusters lets fit the algorithm and visualize the plot
optimal_no_of_clusters = 3
cluster_labels = []
algo = KMeans(n_clusters=optimal_no_of_clusters,random_state=0)
algo.fit(actor_embeddings)
cluster_labels.append(algo.labels_)

# Train Tsne to obtain vectors in low dimensions space:-
from sklearn.manifold import TSNE
import seaborn as sns
import pandas as pd

transform = TSNE
```

```
trans = transform(n_components=2) #--> get the top 2 component to visualize
node_embeddings_2d = trans.fit_transform(actor_embeddings)

# create a dataframe out of the node_embeddings & cluster_labels
cl_label = np.array(cluster_labels).reshape(-1,)
d = {'col1':node_embeddings_2d[:,0],'col2':node_embeddings_2d[:,1],'cluster-label':cl_label}
data = pd.DataFrame(data=d)

# draw the 2d points
plt.figure(figsize=(20,15))
plt.axes().set(aspect="equal")

sns.scatterplot(data=data,x = 'col1',y = 'col2',hue = 'cluster-label',
            alpha=0.4,legend = 'full',palette="deep")

plt.title('{} visualization of actor embeddings'.format(transform.__name__))
plt.show()
```





Grouping similar movies

In [26]:
```
# Define the number of clusters
no_of_clusters = [3,5,10,30,50,100,200,500]

# Run the kmeans algorithm with different number of clusters
cluster_labels = []
for i in no_of_clusters:
    algo = KMeans(n_clusters=i,random_state=0).fit(movie_embeddings)
    cluster_labels.append(algo.labels_)
```

```
list_of_all_clusters_for_diff_values_of_k = []

# assign the nodes with respective cluster labels for diff number of clusters:
for j in range(len(no_of_clusters)):

    list_of_all_clusters = [] # To store all the nodes with respect to unique cluster label
    unique = np.unique(cluster_labels[j]) # get the unique cluster labels

    # zip the nodes with their respective cluster labels:-
    dict_of_movie_nodes = dict(zip(movie_nodes,cluster_labels[j]))

    for n in unique:     # for each unique cluster label
        clust = []
        for i,j in dict_of_movie_nodes.items():
            if j == n:
                clust.append(i)
        list_of_all_clusters.append(clust)

    # append each group of cluster sets for different number of clusters(k):-
    list_of_all_clusters_for_diff_values_of_k.append(list_of_all_clusters)

print(len(list_of_all_clusters_for_diff_values_of_k)) # should be 8
```

8

DEFINE COST-2 FUNCTION FOR MOVIE EMBEDDINGS:-

In [27]:

```
# reference:-https://networkx.org/documentation/stable/reference/algorithms/bipartite.html
def cost2(graph,number_of_clusters):
    '''In this function, we will calculate cost2'''
    movies = []     # To store movie nodes
    actors = []     # To store actor nodes
    for i in graph.nodes():
        if 'm' in i:
            movies.append(i)
        if 'a' in i:
            actors.append(i)
    lst = []       # To store number of degrees of each actor node.
    for j in list(set(movies)):
        lst.append(graph.degree(j))
```

```
    n = sum(lst)   # we get the numerator part of cost-2 formula.
    d = len(list(set(actors)))     # we get the denominator part of cost-2 formula.

    #apply the formula
    cost2 = (1/(number_of_clusters)) * (n/d)

    return cost2
```

OBTAIN OPTIMAL NUMBER OF CLUSTERS BASED ON COSTS:

In [28]:

```
from networkx.algorithms import bipartite # import necessary library

no_of_clusters = [3,5,10,30,50,100,200,500] # define number_of_clusters

final_cost_on_k_cluster_sets = [] # to store final cost for diff number of clusters

for cluster_sets in list_of_all_clusters_for_diff_values_of_k:

    cost_1,cost_2 = 0,0       # initiate the cost2 to zero

    for groups in cluster_sets:

        G = nx.Graph()  # this is a empty cluster-graph which may be disconnected

        for m in groups:
            # create subgraphs using ego_graph()
            subgraph_object = nx.ego_graph(B,m)

            # add nodes,edges to the main cluster-graph(G) using subgraph object
            G.add_nodes_from(subgraph_object.nodes())
            G.add_edges_from(subgraph_object.edges())

        # Update the costs for each group of clusters:-
        cost_1 += cost1(G,len(cluster_sets))
        cost_2 += cost2(G,len(cluster_sets))

    # obtain the finalcost on the cluster_sets for different number of clusters(k)
    finalcost = (cost_1*cost_2)

    final_cost_on_k_cluster_sets.append(finalcost)
```

```
print("The finalcost values for different number of clusters : \n",
      final_cost_on_k_cluster_sets)

max_cost = max(final_cost_on_k_cluster_sets)

idx = final_cost_on_k_cluster_sets.index(max_cost)

print("The optimal number of clusters with max cost : ",no_of_clusters[idx])
```

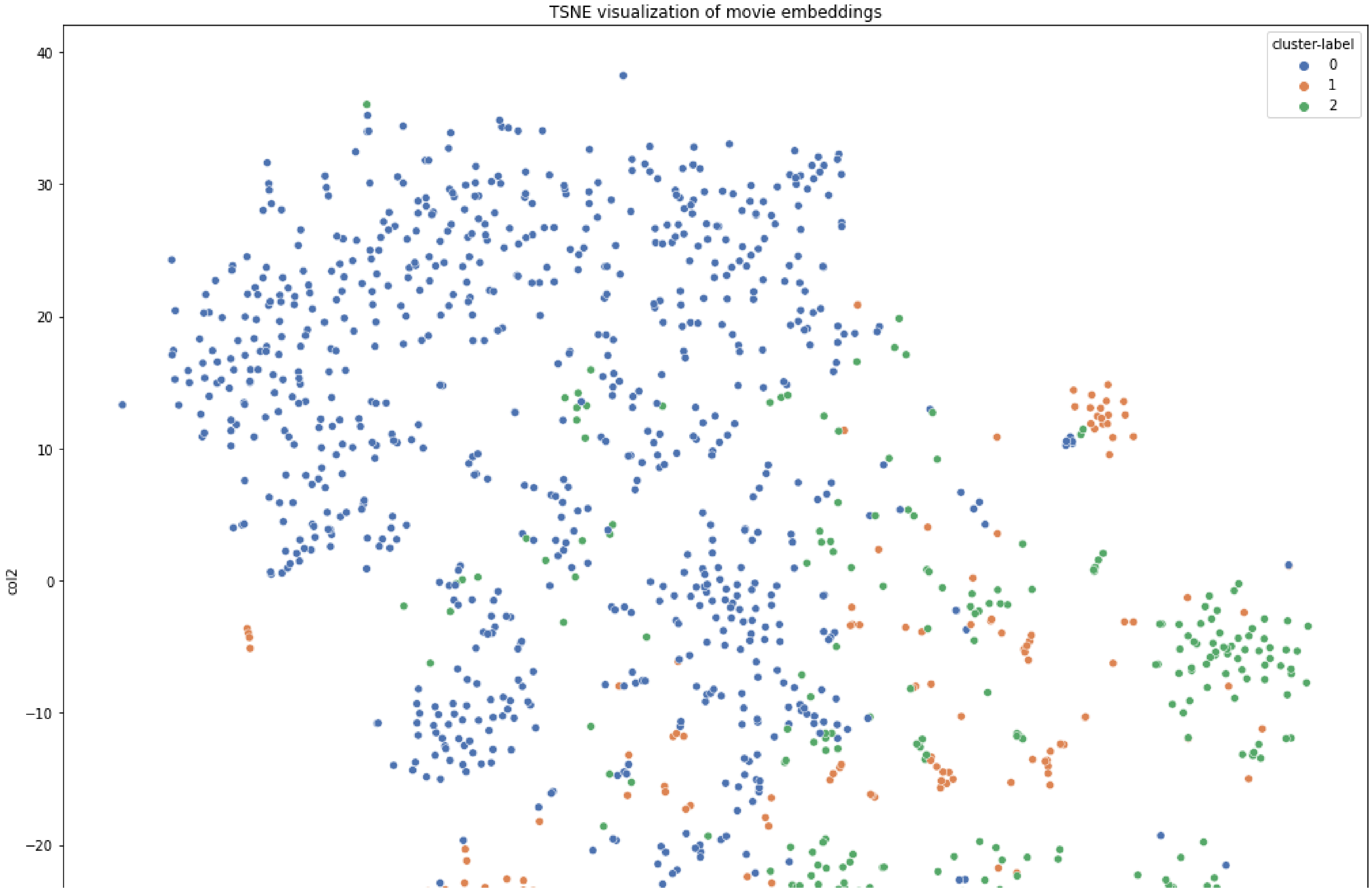
```
The finalcost values for different number of clusters :
 [2.701561790469029, 2.5294046060794866, 2.2735470974303675, 2.171647657680863, 1.8307389898663755, 1.624905438789246
3, 1.3714298880759872, 1.1996860363538708]
The optimal number of clusters with max cost :  3
```

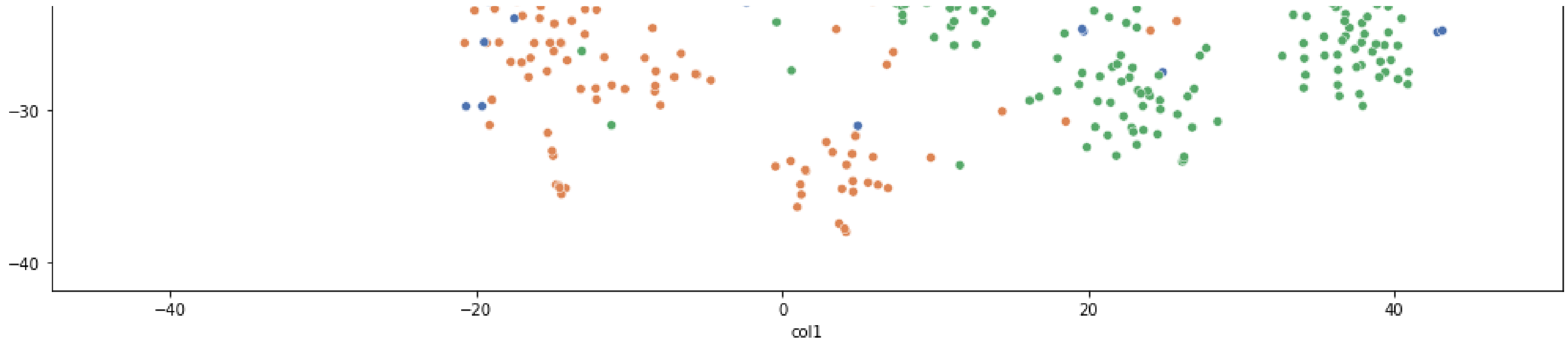
Displaying similar movie clusters

In [29]:

```
#with optimal number of clusters lets fit the algorithm and visualize the plot

optimal_no_of_clusters = 3
cluster_labels = []
algo = KMeans(n_clusters=optimal_no_of_clusters,random_state=0)
algo.fit(movie_embeddings)
cluster_labels.append(algo.labels_)

# Train Tsne to obtain vectors in low dimensions space:-
from sklearn.manifold import TSNE
import seaborn as sns
import pandas as pd

transform = TSNE
trans = transform(n_components=2)#--> get the top 2 component to visualize
node_embeddings_2d = trans.fit_transform(movie_embeddings)

# create a dataframe out of the node_embeddings & cluster_labels
cl_label = np.array(cluster_labels).reshape(-1,)
d = {'col1':node_embeddings_2d[:,0],'col2':node_embeddings_2d[:,1],'cluster-label':cl_label}
data = pd.DataFrame(data=d)

# draw the 2d points
plt.figure(figsize=(20,15))
plt.axes().set(aspect="equal")
```

```
sns.scatterplot(data=data,x = 'col1',y = 'col2',hue = 'cluster-label',legend = 'full',palette="deep")

plt.title('{} visualization of movie embeddings'.format(transform.__name__))
plt.show()
```





−30
−40
−40
−20
0
20
40
col1