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.

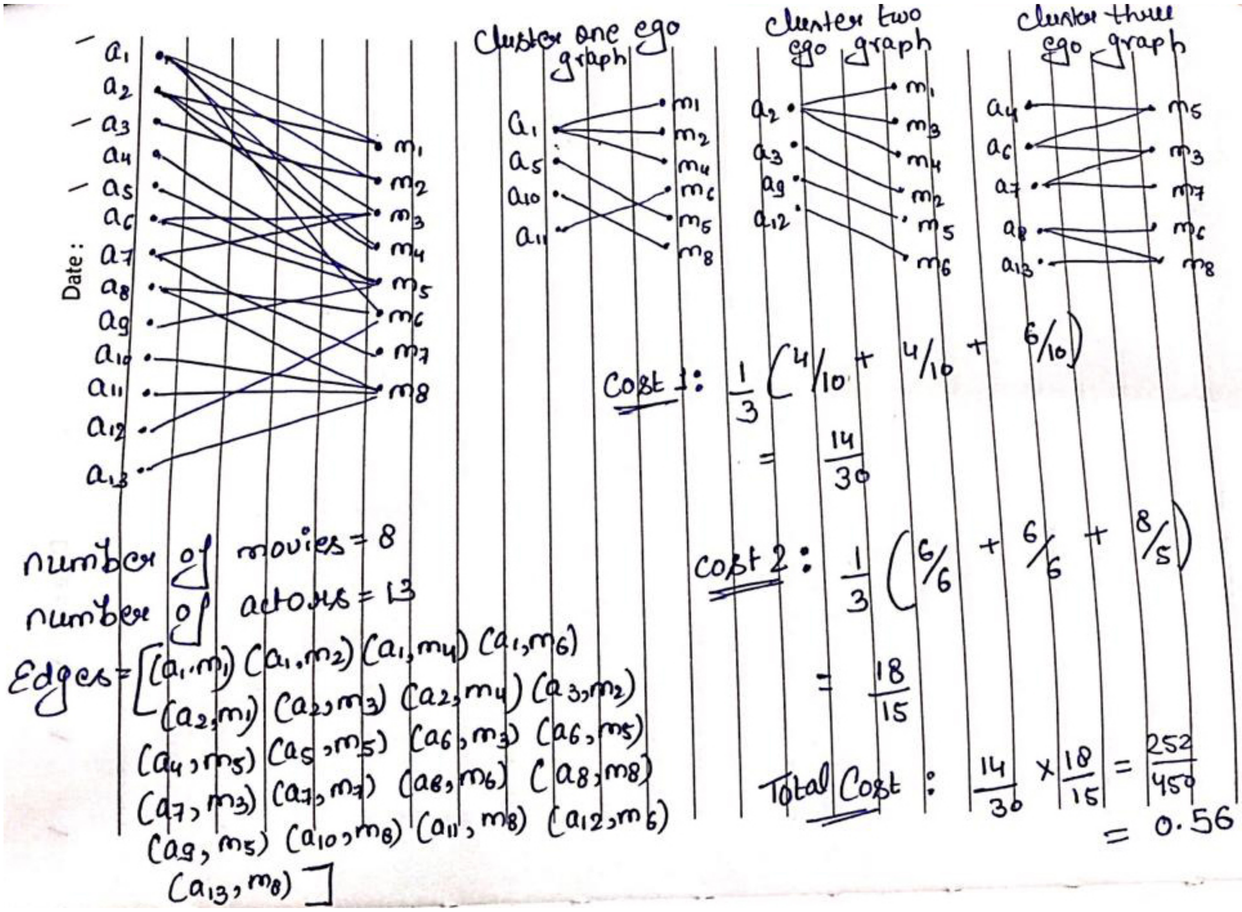
Every Grader function has to return True.

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

In [1]:

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

```
Collecting networkx==2.3
  Downloading networkx-2.3.zip (1.7 MB)
Requirement already satisfied: decorator>=4.3.0 in c:\users\buchi\anaconda\lib\site-pack
ages (from networkx==2.3) (4.4.2)
Building wheels for collected packages: networkx
  Building wheel for networkx (setup.py): started
  Building wheel for networkx (setup.py): finished with status 'done'
  Created wheel for networkx: filename=networkx-2.3-py2.py3-none-any.whl size=1555995 sh
a256=e24b4c2eb8941044b2fbca334a788d47926ee04a5c52fff55bbcba8816835e7b
  Stored in directory: c:\users\buchi\appdata\local\pip\cache\wheels\ff\62\9e\0ed2d25fd4
f5761e2d19568cda0c32716556dfa682e65ecf64
Successfully built networkx
Installing collected packages: networkx
  Attempting uninstall: networkx
    Found existing installation: networkx 2.5
    Uninstalling networkx-2.5:
      Successfully uninstalled networkx-2.5
Successfully installed networkx-2.3
```

In [3]:

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

In [2]:

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

In [3]:

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

In [4]:

```
B = nx.Graph()
B.add_nodes_from(data['movie'].unique(), bipartite=0, label='movie')
```

```
B.add_nodes_from(data['actor'].unique(), bipartite=1, label='actor')
B.add_edges_from(edges, label='acted')
```

In [5]:
```
A = list(nx.connected_component_subgraphs(B))[0]
```

In [6]:
```
print("number of nodes", A.number_of_nodes())
print("number of edges", A.number_of_edges())
```

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

In [ ]:
```
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))
```

In [8]:
```
# Create the random walker
rw = UniformRandomMetaPathWalk(StellarGraph(A))

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

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

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

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

In [9]:
```
from gensim.models import Word2Vec
model = Word2Vec(walks, vector_size=128, window=5)
```

In [10]:
```
model.wv.vectors.shape  # 128-dimensional vector for each node in the graph
```

Out[10]:
```
(4703, 128)
```

In [11]:
```
# Retrieve node embeddings and corresponding subjects
node_ids = list(model.wv.index_to_key)  # list of node IDs
node_embeddings = model.wv.vectors  # numpy.ndarray of size number of nodes times embed
node_targets = [ A.node[node_id]['label'] for node_id in node_ids]
```

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

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

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

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

In [12]:
```
print(node_ids[:15],end='')
```

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

```
['a973', 'a967', 'a964', 'a1731', 'a970', 'a969', 'a1028', 'a1057', 'a1003', 'a965', 'm1
094', 'm1111', 'm67', 'a988', 'a959']
['actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'acto
r', 'movie', 'movie', 'movie', 'actor', 'actor']
```

In [13]:

```
def data_split(node_ids,node_targets,node_embeddings):
    '''In this function, we will split the node embeddings into actor_embeddings , movi
    actor_nodes,movie_nodes=[],[]
    actor_embeddings,movie_embeddings=[],[]
    # split the node_embeddings into actor_embeddings,movie_embeddings based on node_id
    # By using node_embedding and node_targets, we can extract actor_embedding and movi
    # By using node_ids and node_targets, we can extract actor_nodes and movie nodes

    for i,x in enumerate(node_ids):
        if node_targets[i]=='actor':
            actor_nodes.append(x)
    for i,x in enumerate(node_ids):
        if node_targets[i]=='movie':
            movie_nodes.append(x)
    for i,x in enumerate(node_embeddings):
        if node_targets[i]=='actor':
            actor_embeddings.append(x)
    for i,x in enumerate(node_embeddings):
        if node_targets[i]=='movie':
            movie_embeddings.append(x)

    return actor_nodes,movie_nodes,np.array(actor_embeddings), np.array(movie_embedding
```

In [14]:

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

Grader function - 1

In [15]:

```
print(len(actor_nodes))
```

```
3411
```

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

In [18]:

```
actor_targets=[ x for x in node_targets if x=='actor']
movie_targets=[ x for x in node_targets if x=='movie']
```

Calculating cost1

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

In [19]:
```
def cost1(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''
    num= max([len(x) for x in list(nx.connected_components(graph))])
    Total_Nodes=graph.number_of_nodes()
    return (1/number_of_clusters)*num/Total_Nodes
```

Grader function - 3

In [21]:
```
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[21]: 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

In [22]:
```
def cost2(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''
    degree = graph.degree()
    nodes = list(graph.nodes())
    unique_nodes = []
    for i in nodes:
        if i not in unique_nodes:
            unique_nodes.append(i)
    summation = 0
    for i in degree:
        if 'a' in i[0]:
            summation+=i[1]
    movie_nodes=0
    for i in unique_nodes:
        if 'm' in i:
            movie_nodes+=1
    return (1/number_of_clusters)*summation/movie_nodes
```

Grader function - 4

In [23]:
```
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[23]: True

Grouping similar actors

In [39]:

```
number_of_clusters = [3, 5, 10, 30, 50, 100, 200, 500]
cost = []
for cl in number_of_clusters:
    kmeans = KMeans(n_clusters=cl)
    kmeans.fit(actor_embeddings)
    cluster_number_for_data_point = kmeans.labels_
    list_of_all_cluster=[]
    unique = np.unique(cluster_number_for_data_point)
    dict_of_actor_nodes = dict(zip(actor_nodes, cluster_number_for_data_point))
    for number in unique:
        cluster=[]
    for node, cluster_number in dict_of_actor_nodes.items():
        if cluster_number == number:
            cluster.append(node)
        list_of_all_cluster.append(cluster)
    cost_1=0
    cost_2=0
    for cluster_ in list_of_all_cluster:
        G= nx.Graph()
        for actor_node in cluster_:
            sub_graph = nx.ego_graph(B,actor_node)
            G.add_nodes_from(sub_graph.nodes())
            G.add_edges_from(sub_graph.edges())
        cost_1+=cost1(G,cl)
        cost_2+=cost2(G,cl)
    print(cost_1*cost_2)
    cost.append(cost_1*cost_2)
```

```
5324601.286348536
1673139.7870819669
306826.3202546481
2942.2907539119115
4653.968400000486
245.9071467391331
290.8730250000304
46.539683999992775
```

In [40]:

```
best_cluster=number_of_clusters[cost.index(max(cost))]
```

In [41]:

```
algo=KMeans(n_clusters=best_cluster)
algo.fit(actor_embeddings)
```

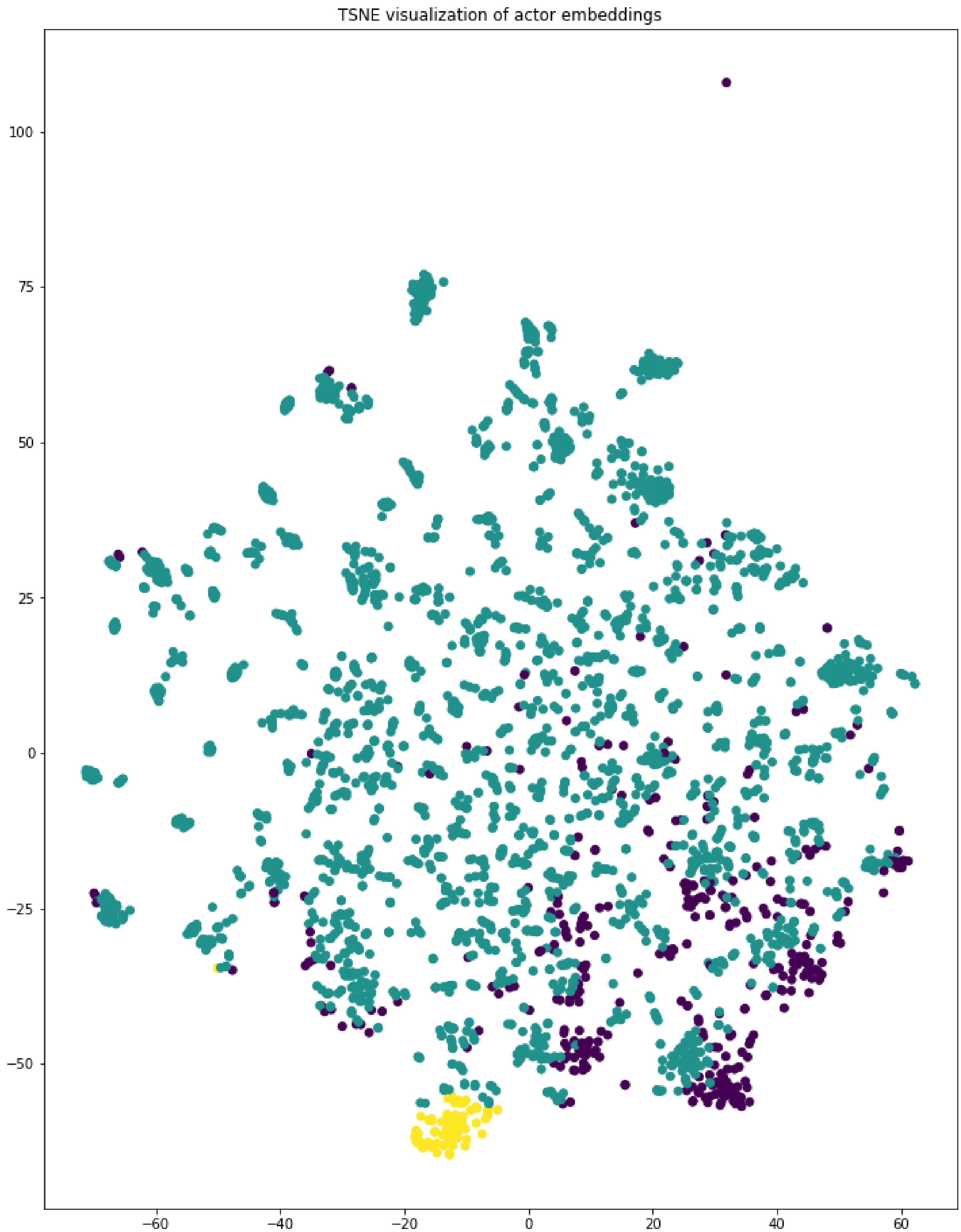
Out[41]: KMeans(n_clusters=3)

Displaying similar actor clusters

In [47]:

```
from sklearn.manifold import TSNE
transform = TSNE #PCA
trans = transform(n_components=2)
actor_embeddings_2d = trans.fit_transform(actor_embeddings)
label_map = { l: i for i, l in enumerate(np.unique(actor_targets))}
actor_colours = [ label_map[target] for target in actor_targets]
plt.figure(figsize=(20,16))
```

```
plt.axes().set(aspect="equal")
plt.scatter(actor_embeddings_2d[:,0],actor_embeddings_2d[:,1],c=algo.predict(actor_embe
plt.title('{} visualization of actor embeddings'.format(transform.__name__))
plt.show()
```



Grouping similar movies

In [ ]:
```
cluster_list=[3,5,10,30,50,100,200,500]
```

```
Cost_movies=[]
for cluster in cluster_list:
    algo_m=KMeans(n_clusters=cluster)
    algo_m.fit(movie_embeddings)
    label_m=algo_m.labels_
    dic=dict(zip(movie_nodes,label_m))
    c1=0
    c2=0
    for i in label_m:
        ac_node = [k for k,v in dic.items() if v == i]
        G1=nx.Graph()
        for n in range(len(ac_node)):
            sub_graph1 = nx.ego_graph(A,node_ids[n])
            G1.add_nodes_from(sub_graph1.nodes)
            G1.add_edges_from(sub_graph1.edges())
        c1+=cost1(G1,cluster)
        c2+=cost2(G1,cluster)
    print(c1*c2)
    Cost_movies.append(c1*c2)
```

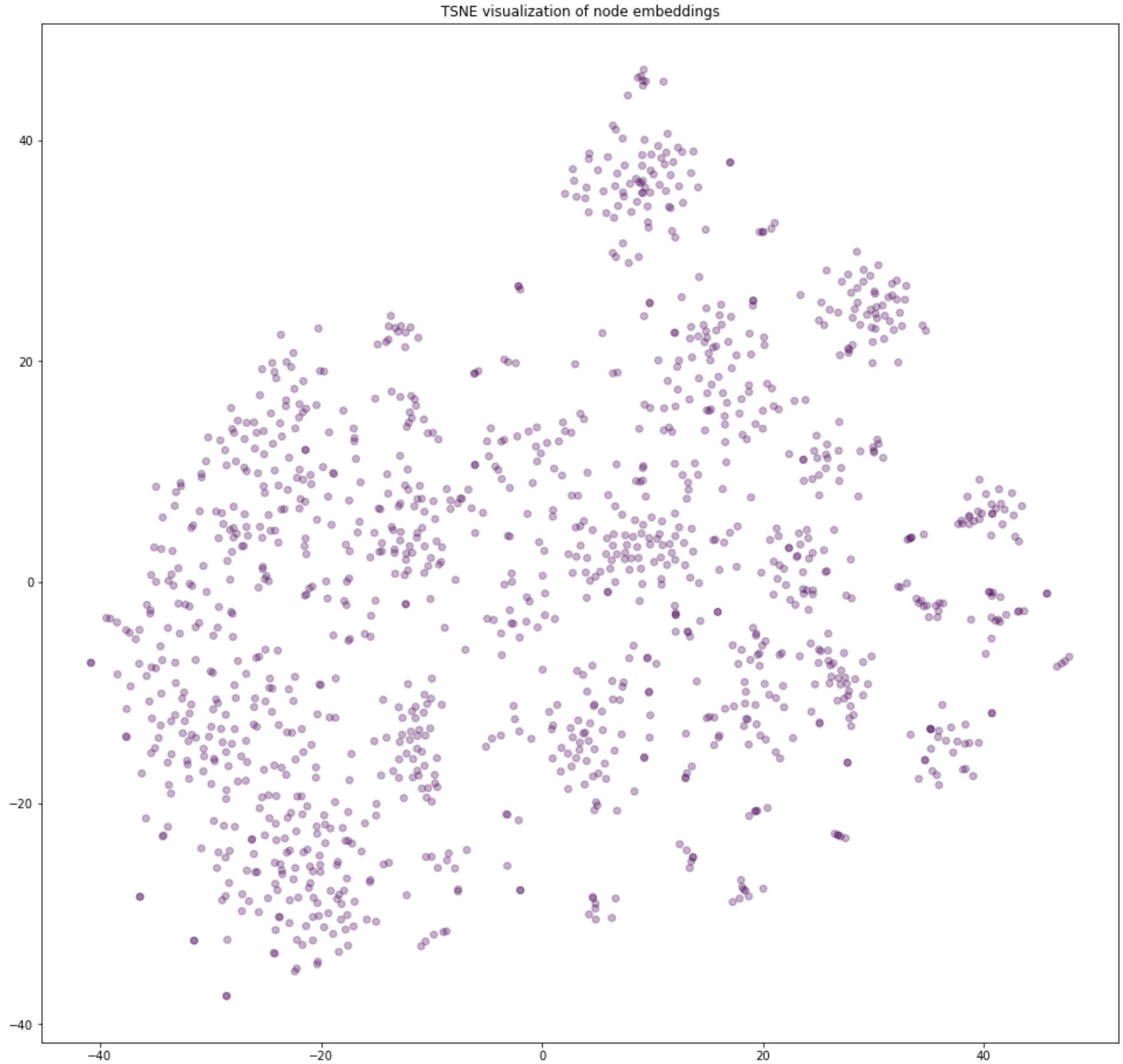
In [32]:
```
best_cluster=cluster_list[Cost_movies.index(max(Cost_movies))]
```

In [33]:
```
kmeans=KMeans(n_clusters=best_cluster)
kmeans.fit(movie_embeddings)
```

Out[33]:
```
KMeans(n_clusters=3)
```

Displaying similar movie clusters

In [35]:
```
from sklearn.manifold import TSNE
transform = TSNE #PCA
trans_ = transform(n_components=2)
movie_embeddings_2d = trans_.fit_transform(movie_embeddings)
import numpy as np
# draw the points
label_map = { l: i for i, l in enumerate(np.unique(movie_targets))}
node_colours = [ label_map[target] for target in movie_targets]
plt.figure(figsize=(20,16))
plt.axes().set(aspect="equal")
plt.scatter(movie_embeddings_2d[:,0],movie_embeddings_2d[:,1],c=node_colours, alpha=0.3
plt.title('{} visualization of node embeddings'.format(transform.__name__))
plt.show()
```


TSNE visualization of node embeddings
40
20
0
−20
−40
−40
−20
0
20
40

In [ ]:

In [ ]:

In [ ]: