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.

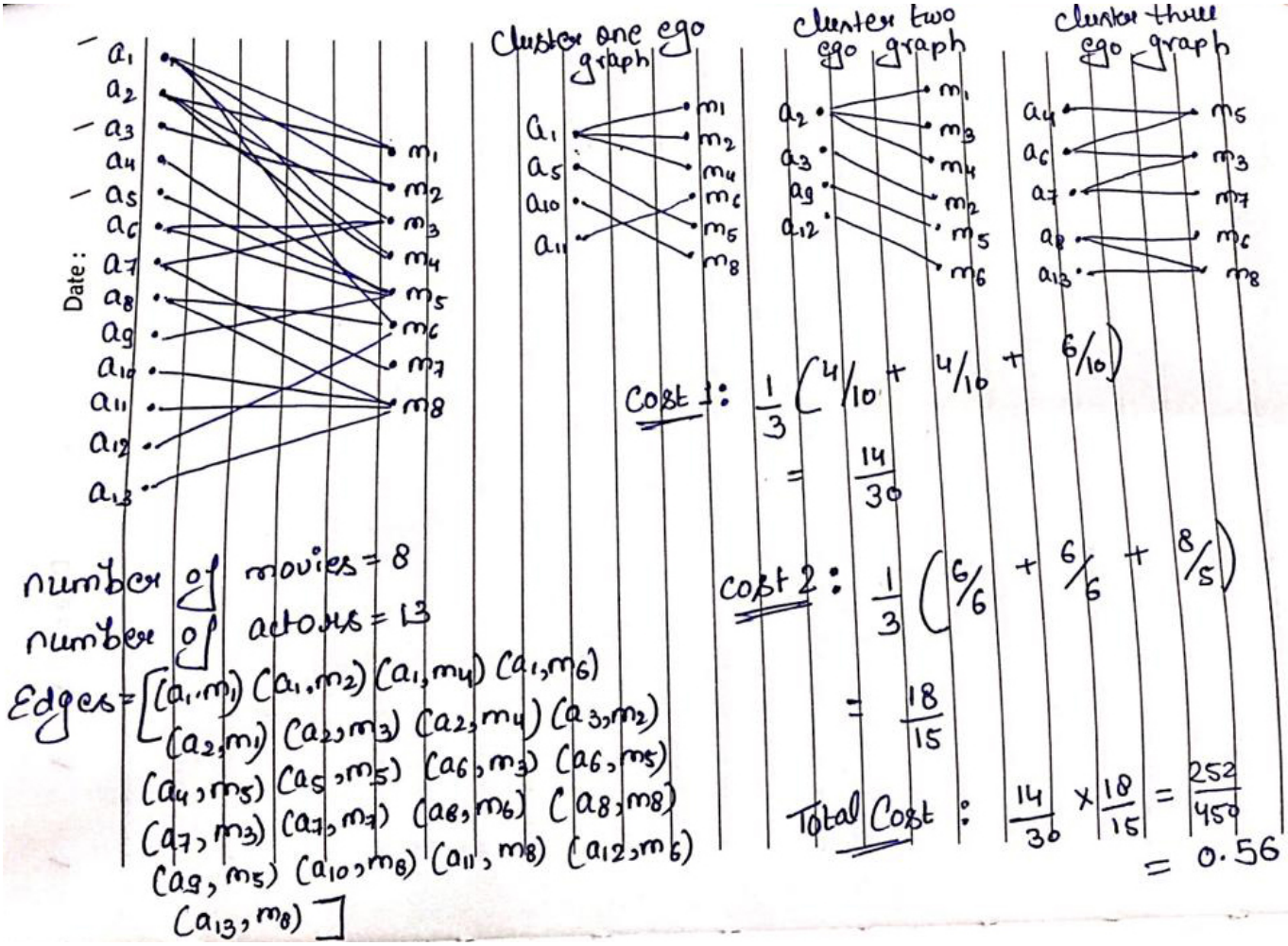
Every Grader function has to return True.

Please check clustering assignment helper functions (https://drive.google.com/file/d/1V29KhKo3YnckMX32treEgdtH5r90DIjU/view?usp=sharing) 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 (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 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(number of nodes in the largest connected component in the graph with the actor nodes and its movie neighbours in c}}{\text{(total number of nodes in that cluster i)}}$
   where N= number of clusters
   (Write your code in def cost1())
5. Cost2 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(sum of degress of actor nodes in the graph with the actor nodes and its movie neighbours in cluster i)}}{\text{(number of unique movie nodes in the graph with the actor nodes and its movie neighbours in cluster i)}}$
   where N= number of clusters
   (Write your code in def cost2())
6. Fit the clustering algorithm with the opimal number_of_clusters and get the cluster number for each node
7. Convert the d-dimensional dense vectors of nodes into 2-dimensional using dimensionality reduction techniques (preferably TSNE)
8. Plot the 2d scatter plot, with the node vectors after step e and give colors to nodes such that same cluster nodes will have same color



Task 2 : Apply clustering algorithm to group similar movies

1. For this task consider only the movie nodes
2. Apply any clustering algorithm of your choice 3.Choose the number of clusters for which you have maximum score of $Cost1 * Cost2$

   Cost1 =
   $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(number of nodes in the largest connected component in the graph with the movie nodes and its actor neighbours in c}}{\text{(total number of nodes in that cluster i)}}$
   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 gr
aphs)
        (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(graph
3) # 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)
     |████████████████████████████████| 1.7 MB 28.5 MB/s
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/py
thon3.7/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-an
y.whl size=1556007 sha256=f3f76ad41651305e81af7a47f298b7937c9ad00458
f5563d15a7ec1a9ea33797
  Stored in directory: /root/.cache/pip/wheels/44/e6/b8/4efaab31158e
9e9ca9ed80b11f6b11130bac9a9672b3cbbeaf
Successfully built networkx
Installing collected packages: networkx
  Attempting uninstall: networkx
    Found existing installation: networkx 2.5.1
    Uninstalling networkx-2.5.1:
      Successfully uninstalled networkx-2.5.1
ERROR: pip's dependency resolver does not currently take into accoun
t all the packages that are installed. This behaviour is the source
 of the following dependency conflicts.
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have im
gaug 0.2.9 which is incompatible.
Successfully installed networkx-2.3
```

In [2]:

```
pip install stellargraph
```

```
Collecting stellargraph
  Downloading stellargraph-1.2.1-py3-none-any.whl (435 kB)
     |████████████████████████████████| 435 kB 28.4 MB/s
Requirement already satisfied: matplotlib>=2.2 in /usr/local/lib/pyt
hon3.7/dist-packages (from stellargraph) (3.2.2)
Requirement already satisfied: numpy>=1.14 in /usr/local/lib/python
3.7/dist-packages (from stellargraph) (1.19.5)
Requirement already satisfied: gensim>=3.4.0 in /usr/local/lib/pytho
n3.7/dist-packages (from stellargraph) (3.6.0)
Requirement already satisfied: tensorflow>=2.1.0 in /usr/local/lib/p
ython3.7/dist-packages (from stellargraph) (2.5.0)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python
3.7/dist-packages (from stellargraph) (1.1.5)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/pytho
n3.7/dist-packages (from stellargraph) (2.3)
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/
python3.7/dist-packages (from stellargraph) (0.22.2.post1)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python
3.7/dist-packages (from stellargraph) (1.4.1)
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/p
ython3.7/dist-packages (from gensim>=3.4.0->stellargraph) (5.1.0)
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.
7/dist-packages (from gensim>=3.4.0->stellargraph) (1.15.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/li
b/python3.7/dist-packages (from matplotlib>=2.2->stellargraph) (2.8.
1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python
3.7/dist-packages (from matplotlib>=2.2->stellargraph) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/p
ython3.7/dist-packages (from matplotlib>=2.2->stellargraph) (1.3.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.
0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.2-
>stellargraph) (2.4.7)
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/py
thon3.7/dist-packages (from networkx>=2.2->stellargraph) (4.4.2)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python
3.7/dist-packages (from pandas>=0.24->stellargraph) (2018.9)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python
3.7/dist-packages (from scikit-learn>=0.20->stellargraph) (1.0.1)
Requirement already satisfied: opt-einsum~=3.3.0 in /usr/local/lib/p
ython3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (3.3.
0)
Requirement already satisfied: wrapt~=1.12.1 in /usr/local/lib/pytho
n3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (1.12.1)
Requirement already satisfied: termcolor~=1.1.0 in /usr/local/lib/py
thon3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (1.1.0)
Requirement already satisfied: absl-py~=0.10 in /usr/local/lib/pytho
n3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (0.12.0)
Requirement already satisfied: tensorflow-estimator<2.6.0,>=2.5.0rc0
in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->s
tellargraph) (2.5.0)
Requirement already satisfied: flatbuffers~=1.12.0 in /usr/local/li
b/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (1.
12)
Requirement already satisfied: keras-preprocessing~=1.1.2 in /usr/lo
cal/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargrap
h) (1.1.2)
Requirement already satisfied: h5py~=3.1.0 in /usr/local/lib/python
3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (3.1.0)
Requirement already satisfied: grpcio~=1.34.0 in /usr/local/lib/pyth
on3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (1.34.1)
```

```
Requirement already satisfied: astunparse~=1.6.3 in /usr/local/lib/p
ython3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (1.6.
3)
Requirement already satisfied: keras-nightly~=2.5.0.dev in /usr/loca
l/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph)
(2.5.0.dev2021032900)
Requirement already satisfied: gast==0.4.0 in /usr/local/lib/python
3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (0.4.0)
Requirement already satisfied: tensorboard~=2.5 in /usr/local/lib/py
thon3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (2.5.0)
Requirement already satisfied: google-pasta~=0.2 in /usr/local/lib/p
ython3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (0.2.
0)
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/pyt
hon3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (3.17.3)
Requirement already satisfied: typing-extensions~=3.7.4 in /usr/loca
l/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph)
(3.7.4.3)
Requirement already satisfied: wheel~=0.35 in /usr/local/lib/python
3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (0.36.2)
Requirement already satisfied: cached-property in /usr/local/lib/pyt
hon3.7/dist-packages (from h5py~=3.1.0->tensorflow>=2.1.0->stellargr
aph) (1.5.2)
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/
python3.7/dist-packages (from tensorboard~=2.5->tensorflow>=2.1.0->s
tellargraph) (57.2.0)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0
in /usr/local/lib/python3.7/dist-packages (from tensorboard~=2.5->te
nsorflow>=2.1.0->stellargraph) (0.6.1)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/pyt
hon3.7/dist-packages (from tensorboard~=2.5->tensorflow>=2.1.0->stel
largraph) (3.3.4)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /us
r/local/lib/python3.7/dist-packages (from tensorboard~=2.5->tensorfl
ow>=2.1.0->stellargraph) (1.8.0)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/li
b/python3.7/dist-packages (from tensorboard~=2.5->tensorflow>=2.1.0-
>stellargraph) (2.23.0)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /
usr/local/lib/python3.7/dist-packages (from tensorboard~=2.5->tensor
flow>=2.1.0->stellargraph) (0.4.4)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/p
ython3.7/dist-packages (from tensorboard~=2.5->tensorflow>=2.1.0->st
ellargraph) (1.0.1)
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/l
ib/python3.7/dist-packages (from tensorboard~=2.5->tensorflow>=2.1.0
->stellargraph) (1.32.1)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/pytho
n3.7/dist-packages (from google-auth<2,>=1.6.3->tensorboard~=2.5->te
nsorflow>=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
~=2.5->tensorflow>=2.1.0->stellargraph) (4.2.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/l
ib/python3.7/dist-packages (from google-auth<2,>=1.6.3->tensorboard~
=2.5->tensorflow>=2.1.0->stellargraph) (0.2.8)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/loca
l/lib/python3.7/dist-packages (from google-auth-oauthlib<0.5,>=0.4.1
->tensorboard~=2.5->tensorflow>=2.1.0->stellargraph) (1.3.0)
Requirement already satisfied: importlib-metadata in /usr/local/lib/
python3.7/dist-packages (from markdown>=2.6.8->tensorboard~=2.5->ten
```

```
sorflow>=2.1.0->stellargraph) (4.6.1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/li
b/python3.7/dist-packages (from pyasn1-modules>=0.2.1->google-auth<
2,>=1.6.3->tensorboard~=2.5->tensorflow>=2.1.0->stellargraph) (0.4.
8)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python
3.7/dist-packages (from requests<3,>=2.21.0->tensorboard~=2.5->tenso
rflow>=2.1.0->stellargraph) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/p
ython3.7/dist-packages (from requests<3,>=2.21.0->tensorboard~=2.5->
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~=2.5-
>tensorflow>=2.1.0->stellargraph) (2021.5.30)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.2
1.1 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.2
1.0->tensorboard~=2.5->tensorflow>=2.1.0->stellargraph) (1.24.3)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/pyt
hon3.7/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oau
thlib<0.5,>=0.4.1->tensorboard~=2.5->tensorflow>=2.1.0->stellargrap
h) (3.1.1)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.
7/dist-packages (from importlib-metadata->markdown>=2.6.8->tensorboa
rd~=2.5->tensorflow>=2.1.0->stellargraph) (3.5.0)
Installing collected packages: stellargraph
Successfully installed stellargraph-1.2.1
```

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 [4]:

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

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

In [5]:

```
data=pd.read_csv('/content/drive/MyDrive/Clustering/movie_actor_network.csv', in
dex_col=False, names=['movie','actor'])
```

In [6]:

```
data
```

Out[6]:

	movie	actor
0	m1	a1
1	m2	a1
2	m2	a2
3	m3	a1
4	m3	a3
...	...	...
9645	m1380	a816
9646	m1380	a962
9647	m1381	a1225
9648	m1381	a1436
9649	m1381	a1926

9650 rows × 2 columns

In [7]:

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

In [8]:

```
edges[0]
```

Out[8]:

```
('m1', 'a1')
```

In [9]:

```
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 [10]:

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

In [11]:

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

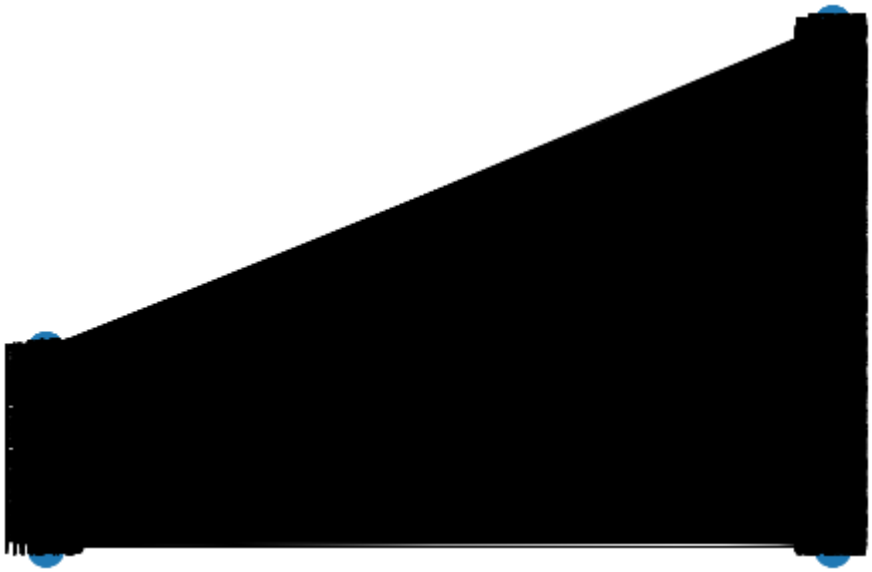
```
number of nodes 4703
number of edges 9650
```

In [12]:

```
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()
```



In [13]:

```
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
```

In [14]:

```
# 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 [15]:

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

In [16]:

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

Out[16]:

```
(4703, 128)
```

In [17]:

```
# 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 time
s 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']
```

In [18]:

```
def data_split(node_ids, node_targets, node_embeddings):
    '''In this function, we will split the node embeddings into actor_embeddings
, movie_embeddings '''
    actor_nodes,movie_nodes=[],[]
    actor_embeddings,movie_embeddings=[],[]
    # 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 a
nd movie embedding
    # By using node_ids and node_targets, we can extract actor_nodes and movie n
odes
    for i in range(len(node_targets)):
      if node_targets[i] == 'actor':
        actor_nodes.append(i)
      else:
        movie_nodes.append(i)
    actor_embeddings = np.array(node_embeddings[actor_nodes])
    movie_embeddings = np.array(node_embeddings[movie_nodes])

    return actor_nodes, movie_nodes, actor_embeddings,movie_embeddings
```

In [19]:

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

In [ ]:

```
movie_embeddings.shape
```

Out[ ]:

```
(1292, 128)
```

Grader function - 1

In [20]:

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

Out[20]:

```
True
```

Grader function - 2

In [21]:

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

Out[21]:

```
True
```

Calculating cost1

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

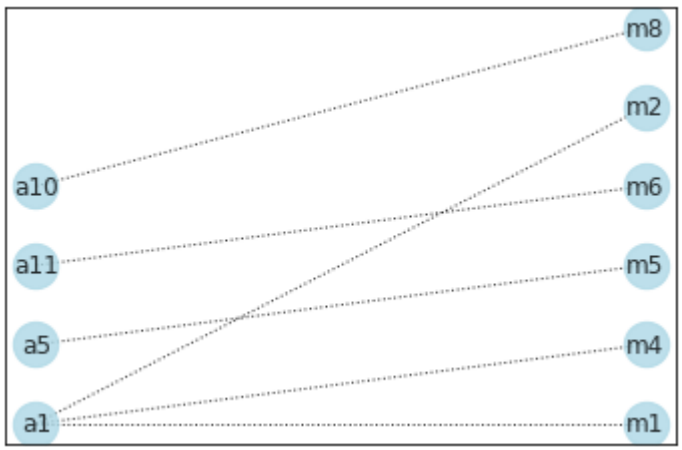
In [22]:

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

    #return cost1
```

In [23]:

```
import networkx as nx
from networkx.algorithms import bipartite
graded_graph= nx.Graph()
graded_graph.add_nodes_from(['a1','a5','a10','a11'], bipartite=0) # Add the node
attribute "bipartite"
graded_graph.add_nodes_from(['m1','m2','m4','m6','m5','m8'], bipartite=1)
graded_graph.add_edges_from([('a1','m1'),('a1','m2'),('a1','m4'),('a11','m6'),(
'a5','m5'),('a10','m8')])
l={'a1','a5','a10','a11'};r={'m1','m2','m4','m6','m5','m8'}
pos = {}
pos.update((node, (1, index)) for index, node in enumerate(l))
pos.update((node, (2, index)) for index, node in enumerate(r))
nx.draw_networkx(graded_graph, pos=pos, with_labels=True,node_color='lightblue',
alpha=0.8,style='dotted',node_size=500)
```



Grader function - 3

In [24]:

```
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[24]:

```
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 [25]:

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

Grader function - 4

In [26]:

```
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[26]:

```
True
```

Grouping similar actors

In [28]:

```
from sklearn.cluster import KMeans
clusters = [3, 5, 10, 30, 50, 100, 200, 500]
cost = []

for cluster in clusters:
  algo = KMeans(n_clusters= cluster)
  algo.fit(actor_embeddings)
  label = algo.labels_

  dic=dict(zip(actor_nodes, label))
  cost_1=0
  cost_2=0

  for i in label:
    ac_node = [k for k,v in dic.items() if v == i]
    G1=nx.Graph()

    for n in 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())
    cost_1 += cost1(G1,cluster)
    cost_2 += cost2(G1,cluster)
  print(cost_1*cost_2)
  cost.append(cost_1*cost_2)
```

```
4956447.15889135
1644239.5211498344
232715.58275306577
17445.08068184931
6017.817628830646
1584.8551970047936
537.0077918194822
119.99408604663631
```

In [35]:

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

In [36]:

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

Out[36]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=30
0,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='a
uto',
       random_state=None, tol=0.0001, verbose=0)
```

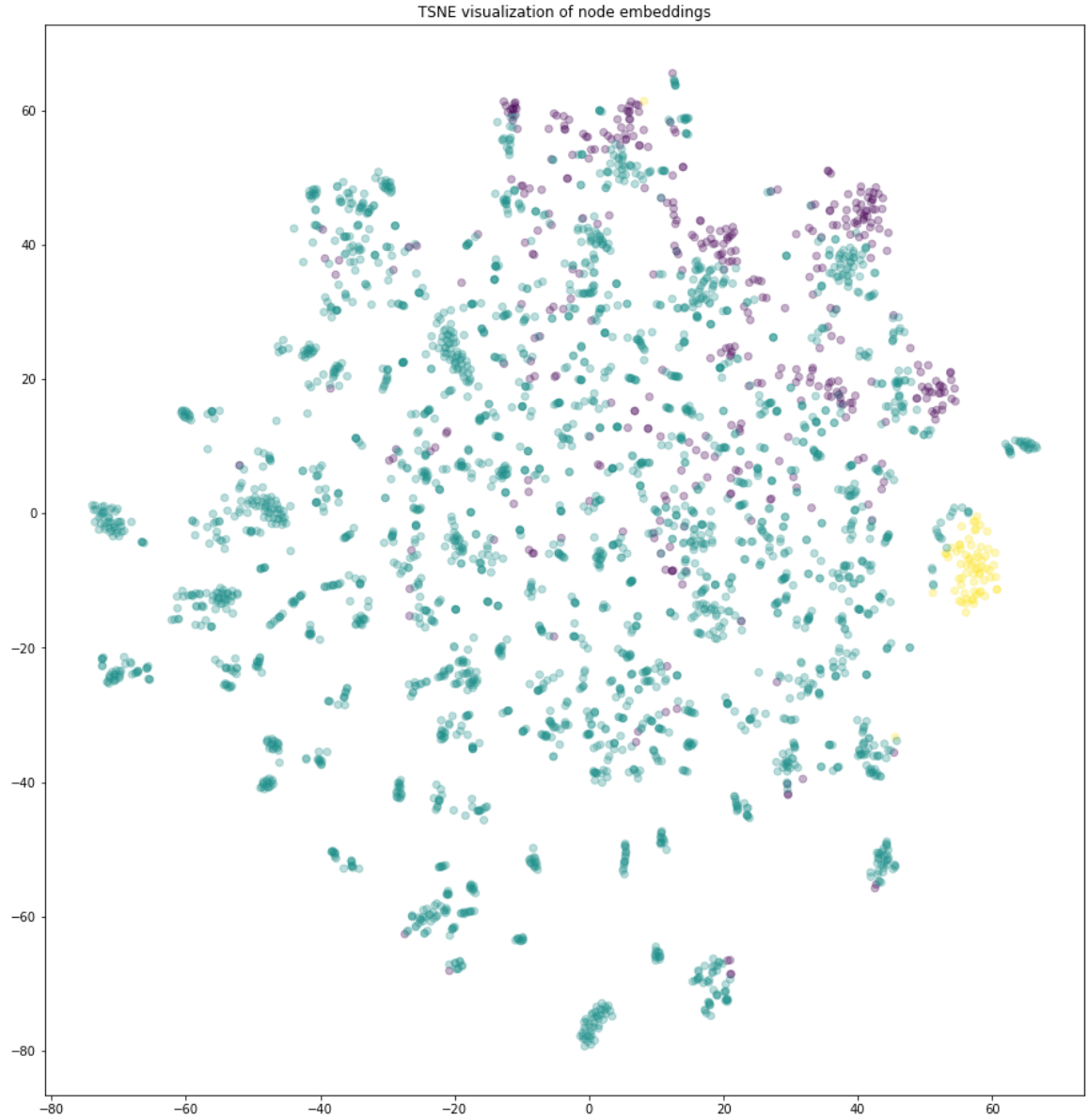
Displaying similar actor clusters

In [38]:

```
from sklearn.manifold import TSNE
transform = TSNE #PCA
trans = transform(n_components=2)
actor_2d = trans.fit_transform(actor_embeddings)
```

In [41]:

```
import numpy as np # draw the points
plt.figure(figsize=(20,16))
plt.axes().set(aspect="equal")
y_kmeans = algo.predict(actor_embeddings)
plt.scatter(actor_2d[:,0], actor_2d[:,1], c=y_kmeans, alpha=0.3)
plt.title('{} visualization of node embeddings'.format(transform.__name__))
plt.show()
```



Grouping similar movies

In [44]:

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

for cluster in cluster_list:
  algom     = KMeans(n_clusters=cluster)
  algom.fit(movie_embeddings)
  labelm    = algom.labels_
  dic       = dict(zip(movie_nodes, labelm))
  cost_1    = 0
  cost_2    = 0

  for i in labelm:
    ac_node = [k for k,v in dic.items() if v == i]
    G1      = nx.Graph()
    for n in 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())
    cost_1 += cost1(G1,cluster)
    cost_2 += cost2(G1,cluster)
  print(cost_1*cost_2)
  Cost_movies.append(cost_1*cost_2)
```

```
1358035.7895882116
487105.21413028863
113054.5428107436
12667.082039050782
4565.615317666703
1136.633166867937
286.6596344759171
47.4282015025055
```

Displaying similar movie clusters

In [45]:

```
best_clusterm=cluster_list[Cost_movies.index(max(Cost_movies))]
```

In [46]:

```
algom=KMeans(n_clusters=best_clusterm)
algom.fit(movie_embeddings)
```

Out[46]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=30
0,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='a
uto',
       random_state=None, tol=0.0001, verbose=0)
```

In [47]:

```
from sklearn.manifold import TSNE
transform = TSNE #PCA
trans = transform(n_components=2)
movies_2d = trans.fit_transform(movie_embeddings)
```

In [48]:

```
import numpy as np # draw the points
plt.figure(figsize=(20,16))
plt.axes().set(aspect="equal")
y_kmeansm = algo.predict(movie_embeddings)
plt.scatter(movies_2d[:,0],movies_2d[:,1],c=y_kmeansm, alpha=0.3)
plt.title('{} visualization of node embeddings'.format(transform.__name__))
plt.show()
```