In [1]:

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

In [2]:

```
# to install this you need to use "pip install stellargraph"
```

In [3]:

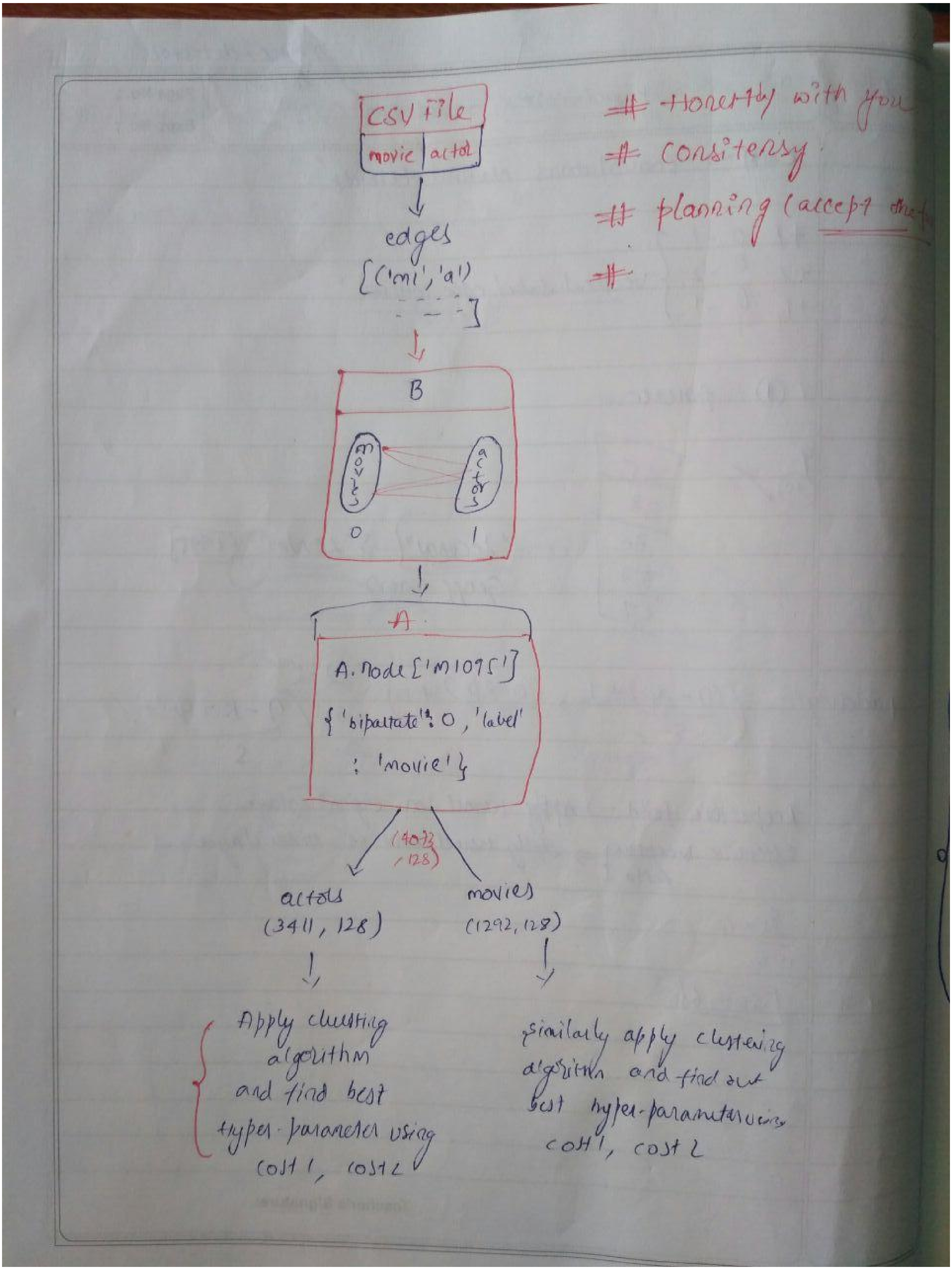
```
print(nx.__version__)
```

```
2.3
```

In [4]:

```
!pip3 freeze | grep matplot
```

```
matplotlib==3.1.2
```


CSV File
movie | actor
edges
[('m1', 'a1')
- - - -]
B
movies
actors
0
1
A
A.node['M1095']
{'bipartate': 0, 'label' : 'movie'}
(4073, 128)
actors
(3411, 128)
movies
(1292, 128)
Apply clustering algorithm and find best hyper-parameter using cost 1, cost 2
similarly apply clustering algorithm and find out best hyper-parameter using cost 1, cost 2
Honesty with you
consistency
planning (accept the
#

In [5]:

```
''' Loading the movie actor network file '''
data=pd.read_csv('movie_actor_network.csv', index_col=False, names=['movie','actor']
data
```

Out[5]:

	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 [6]:

```
''' creating edges from the pandas dataframe '''
edges = [tuple(x) for x in data.values.tolist()]
edges
```

Out[6]:

```
[('m1', 'a1'),
 ('m2', 'a1'),
 ('m2', 'a2'),
 ('m3', 'a1'),
 ('m3', 'a3'),
 ('m3', 'a4'),
 ('m3', 'a5'),
 ('m3', 'a6'),
 ('m3', 'a7'),
 ('m4', 'a1'),
 ('m4', 'a10'),
 ('m4', 'a8'),
 ('m4', 'a9'),
 ('m5', 'a1'),
 ('m5', 'a11'),
 ('m5', 'a12'),
 ('m5', 'a13'),
 ('m5', 'a14'),
```

In [7]:

```
''' Creating a graph network '''
B = nx.Graph()
#adding nodes to graph
B.add_nodes_from(data['movie'].unique(), bipartite=0, label='movie')
B.add_nodes_from(data['actor'].unique(), bipartite=1, label='actor')
#adding edges to graph
B.add_edges_from(edges, label='acted')
B.edges
```

Out[7]:

```
EdgeView([('m1', 'a1'), ('m2', 'a1'), ('m2', 'a2'), ('m3', 'a1'), ('m
3', 'a3'), ('m3', 'a4'), ('m3', 'a5'), ('m3', 'a6'), ('m3', 'a7'), ('m
4', 'a1'), ('m4', 'a10'), ('m4', 'a8'), ('m4', 'a9'), ('m5', 'a1'),
('m5', 'a11'), ('m5', 'a12'), ('m5', 'a13'), ('m5', 'a14'), ('m5', 'a1
5'), ('m5', 'a16'), ('m5', 'a17'), ('m5', 'a18'), ('m5', 'a19'), ('m
7', 'a21'), ('m7', 'a22'), ('m8', 'a22'), ('m9', 'a22'), ('m9', 'a2
3'), ('m10', 'a22'), ('m10', 'a24'), ('m10', 'a25'), ('m11', 'a22'),
('m11', 'a26'), ('m11', 'a27'), ('m11', 'a28'), ('m11', 'a29'), ('m2
4', 'a47'), ('m25', 'a47'), ('m25', 'a48'), ('m25', 'a49'), ('m25', 'a
50'), ('m25', 'a51'), ('m25', 'a52'), ('m25', 'a53'), ('m25', 'a54'),
('m25', 'a55'), ('m25', 'a56'), ('m25', 'a57'), ('m25', 'a58'), ('m2
5', 'a59'), ('m25', 'a60'), ('m25', 'a61'), ('m25', 'a62'), ('m25', 'a
63'), ('m25', 'a64'), ('m25', 'a65'), ('m25', 'a66'), ('m25', 'a67'),
('m25', 'a68'), ('m25', 'a69'), ('m25', 'a70'), ('m26', 'a47'), ('m2
6', 'a71'), ('m26', 'a72'), ('m26', 'a73'), ('m26', 'a74'), ('m26', 'a
75'), ('m26', 'a76'), ('m26', 'a77'), ('m26', 'a78'), ('m26', 'a79'),
('m26', 'a80'), ('m26', 'a81'), ('m26', 'a82'), ('m26', 'a83'), ('m2
6', 'a84'), ('m26', 'a85'), ('m26', 'a86'), ('m26', 'a87'), ('m27', 'a
```

In [8]:

```
''' Getting the coonected sub graph components'''
A = list(nx.connected_component_subgraphs(B))[0]
```

In [9]:

```
# File weights are already saved
```

In [10]:

```
from gensim.models import Word2Vec
clustering_model = Word2Vec.load("clustering_word2vec.model")
```

In [11]:

```
#shape afer using Word2Vec model of 128 dim
clustering_model.wv.vectors.shape
```

Out[11]:

```
(4703, 128)
```

In [12]:

```
# Word2Vec vectors
get_clustering_model_vectors = clustering_model.wv.vectors
get_clustering_model_vectors
```

Out[12]:

```
array([[ 0.49040353,  0.35513365, -0.09746691, ..., -0.5842914 ,
         0.7842303 , -1.5075505 ],
       [ 0.0539096 ,  0.2200051 , -0.84187573, ..., -1.1471574 ,
        -0.08991306, -0.43154967],
       [-1.3629588 ,  0.56075376,  0.17978881, ..., -0.6958599 ,
        -0.03780475,  0.28467494],
       ...,
       [-0.06870703, -0.01312153, -0.14908783, ..., -0.08773582,
        -0.0306545 , -0.14263871],
       [-0.06142563, -0.06422818, -0.16347292, ..., -0.12444828,
        -0.08794963, -0.04515016],
       [-0.06936088, -0.01347885, -0.11517613, ..., -0.09079985,
        -0.11277723,  0.00460553]], dtype=float32)
```

In [13]:

```
# stroing indexs from Word2Vec
get_clustering_model_ids = clustering_model.wv.index2word
get_clustering_model_ids
```

Out[13]:

```
['a973',
 'a967',
 'a1731',
 'a964',
 'a970',
 'a969',
 'a1057',
 'a1028',
 'a965',
 'm1094',
 'm1111',
 'a1003',
 'm1100',
 'a959',
 'a966',
 'a988',
 'a1037',
 'm67',
```

In [14]:

```
#getting movie actor ids
get_actor_model_target_ids = [ele_indx   for ele_indx,each_index in enumerate(get_cl
get_actor_model_target_ids
```

```
 737,
 767,
 777,
 783,
 790,
 791,
 794,
 799,
 800,
 809,
 821,
 828,
 831,
 835,
 839,
 848,
 851,
 852,
 853,
```

In [15]:

```
#getting the movie actor vectos - data points
get_actor_model_records = get_clustering_model_vectors[get_actor_model_target_ids]
get_actor_model_records.shape
```

Out[15]:

```
(3411, 128)
```

In [16]:

```
# cross checking whether the data points are actors or not
for i in get_actor_model_target_ids:
    print(get_clustering_model_ids[i])
```

```
a2361
a3561
a1
a434
a51
a316
a416
a239
a794
a1704
a1222
a1329
a818
a771
a115
a984
a854
a1384
a1416
a865
```

In [17]:

```
''' function to return the right side nodes in connected components - written for cu
def return_right_side_nodes_count(total_nodes,left_side_nodes):
    v_nodes_count = 0
    for item in total_nodes:
        if item not in left_side_nodes:
            v_nodes_count += 1
    return v_nodes_count
```

1. Read Graph from the given movie_actor_network.csv note that the graph is bipartite graph
2. using stellergraph and gensim packages, get the dense representation(128dimensional vector) of every node in the graph
3. Apply Clustering Algorithm to group similar actors

   a. For this task consider only the actor nodes
   b. Apply any clustering algorithm of your choice
   c. Choose the number of clusters for which you have maximum score of $Cost1 * Cost2$

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

   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

```
    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)
        computer the metric Cost = Cost1*Cost2
    return number_of_clusters which have maximum Cost
```

d. Fit the clustering algorithm with the opimal number_of_clusters and get the cluster number for each node
e. Convert the d-dimensional dense vectors of nodes into 2-dimensional using dimensionality reduction techniques (preferably TSNE)
f. 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

4. Apply Clustering Algorithm to group similar movies

   a. for this task consider only the movie nodes

   b. apply any clustering algorithm of your choice

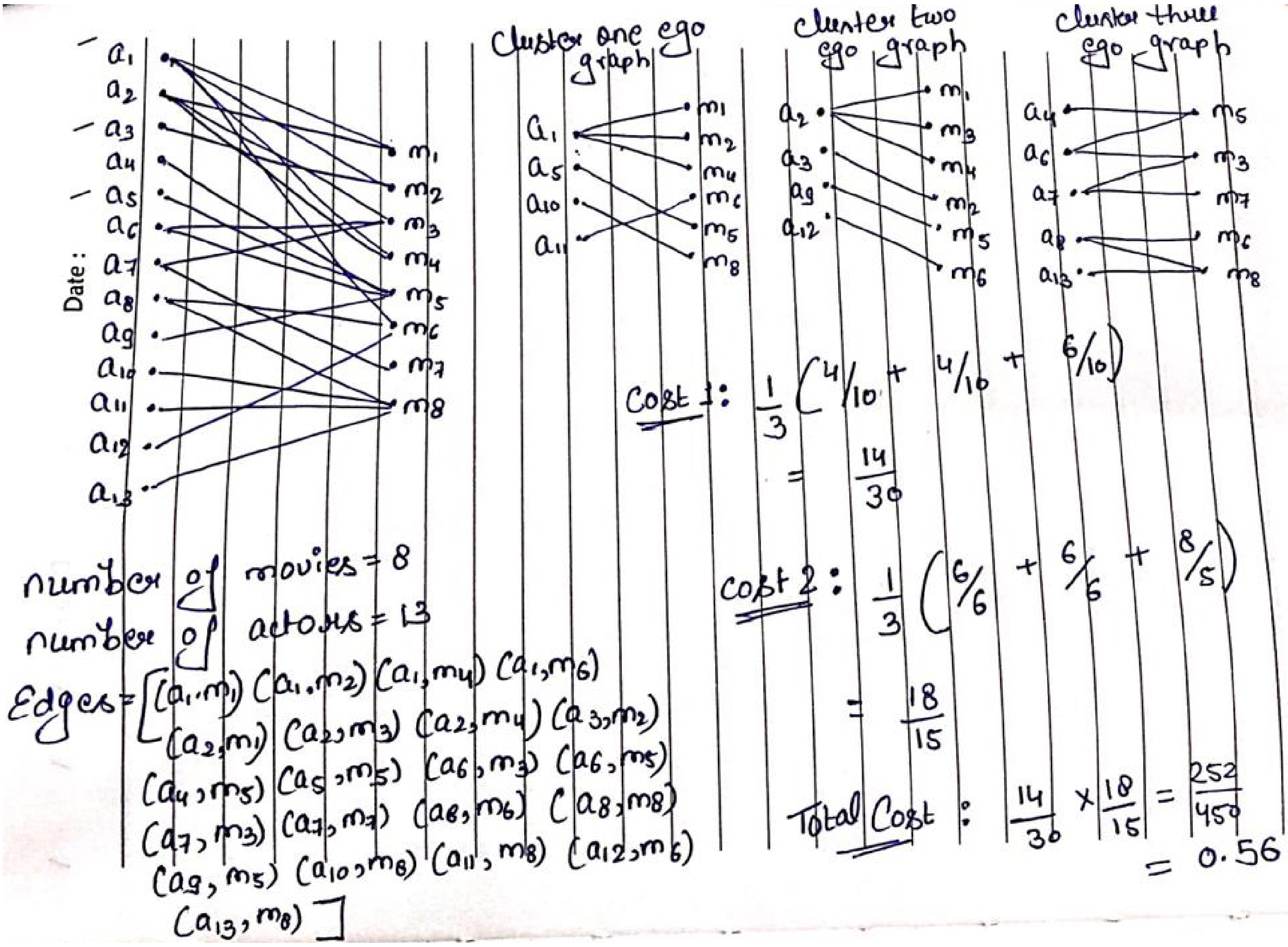
   c. choose the number of clusters for which you have maximum score of $Cost1 * Cost2$

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

   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

```
    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)
        computer the metric Cost = Cost1*Cost2
    return number_of_clusters which have maximum Cost
```

d. Fit the clustering algorithm with the opimal number_of_clusters and get the cluster number for each node
e. Convert the d-dimensional dense vectors of nodes into 2-dimensional using dimensionality reduction techniques (preferably TSNE)
f. 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



these links and function might be usefull while solving this assignment

1. what is bipartite graph: https://en.wikipedia.org/wiki/Bipartite_graph (https://en.wikipedia.org/wiki/Bipartite_graph)
2. Ego graph: https://networkx.github.io/documentation/stable/reference/generated/networkx.generators.ego.ego_graph.h (https://networkx.github.io/documentation/stable/reference/generated/networkx.generators.ego.ego_graph.l
3. Combining two are more graphs: https://stackoverflow.com/a/32652764/4084039 (https://stackoverflow.com/a/32652764/4084039) ex: if you want to merge three graphs which are mentiond in the above image, you can write like this

```
    U=nx.Graph()
    for i in number of clusters:
        if U is empty:
            U.add_edges_from(ith Cluster's graph.edges(data=True))
            U.add_nodes_from(ith Cluster's graph.nodes(data=True))
        else:
            U.add_edges_from(ith Cluster's graph.edges(data=True)+U.edges(d
ata=True))
            U.add_nodes_from(ith Cluster's graph.nodes(data=True)+U.nodes(d
ata=True))
```

4. connected components: https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.comp

(https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.comp

5. Degree of a node: https://networkx.github.io/documentation/stable/reference/classes/generated/networkx.Graph.degree.html (https://networkx.github.io/documentation/stable/reference/classes/generated/networkx.Graph.degree.html)
6. Neighbors of node: https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.Graph.neighbors.html (https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.Graph.neighbors.html)

ACTORS CLUSTERING

In [18]:

```
from sklearn.cluster import KMeans
import numpy as np

#calculation for
overall_cost_1 = []
overall_cost_2 = []
overall_cost   = []
#getting required indexs and datapoints for actor
get_actor_model_target_ids = [ele_indx   for ele_indx,each_index in enumerate(get_cl
get_actor_model_target_ids
get_actor_model_records = get_clustering_model_vectors[get_actor_model_target_ids]
get_actor_model_records.shape
clusters = [3, 5, 10, 30, 50, 100, 200, 500]
#running on different cluster for actor data points
for i in clusters:
    #kmeans model applying
    X = get_actor_model_records
    kmeans = KMeans(n_clusters=i, random_state=0).fit(X)

    #mapping the nodes as keys
    list_of_all_clusters = []
    unique_labels = np.unique(kmeans.labels_)
    dict_of_actor_nodes = {get_clustering_model_ids[indx]:item for indx,item in zip(
    #mapping funciton to actor ndoes
    for each_unique_cluster in unique_labels:
        current_cluster = []
        for node_id,cluster_number in dict_of_actor_nodes.items():
            if cluster_number == each_unique_cluster:
                current_cluster.append(node_id)
        print("{} cluster length is {}".format(str(each_unique_cluster),str(len(curr
        list_of_all_clusters.append(current_cluster)

    # finally calculating cost-1 & cost-2
    cost1 = 0
    cost2 = 0
    total_clusters = len(list_of_all_clusters)
    for each_cluster in list_of_all_clusters:
        G = nx.Graph()
        cost_1_sub_graph_ratio = 0
        cost_2_sub_graph_ratio = 0
        for each_node in each_cluster:
            #print(each_node)
            ego_subgrapgh_object = nx.ego_graph(B,each_node)
            G.add_nodes_from(ego_subgrapgh_object.nodes())
            G.add_edges_from(ego_subgrapgh_object.edges())
        ''' for calculating cost one '''
        #largest connected components
        largest_connect_components =  max(nx.connected_component_subgraphs(G), key=l
        largest_connected_nodes     =  len(largest_connect_components.nodes())
        # total nodes in sub-graph
        total_nodes_in_subgraph     =  nx.number_of_nodes(G)
        cost_1_sub_graph_ratio      +=  largest_connected_nodes/total_nodes_in_subgra

        ''' for calculating cost two '''
        total_edges_in_subgraph     =  nx.number_of_edges(G)
        total_right_side_nodes      =  return_right_side_nodes_count(G.nodes(),each_c
        cost_2_sub_graph_ratio      +=  total_edges_in_subgraph/total_right_side_node
        print(cost_1_sub_graph_ratio, cost_2_sub_graph_ratio)
    cost1 = cost_1_sub_graph_ratio/total_clusters
```

```
    cost2 = cost_2_sub_graph_ratio/total_clusters
    overall_cost_1.append(cost1)
    overall_cost_2.append(cost2)
    overall_cost.append(cost1*cost2)
    print("Final cost1 and cost 2 is {}, {} - overall cost is {}".format(cost1, cost
```

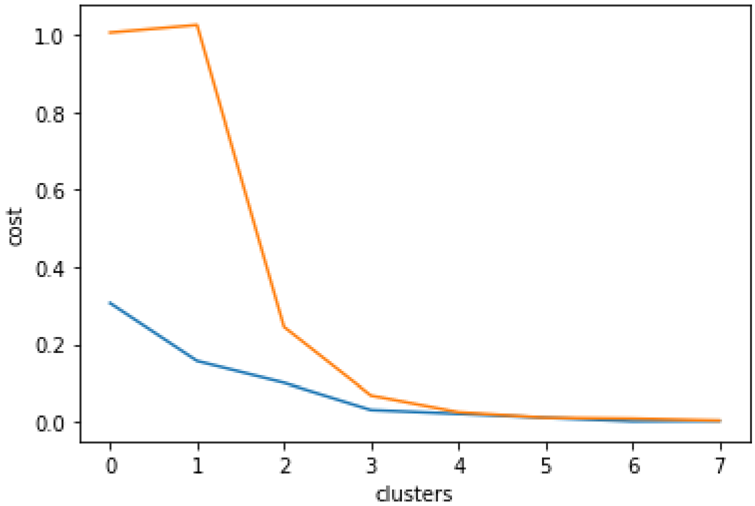
```
1.0 1.8992805755395683
0.8937329700272479 2.0191387559808613
Final cost1 and cost 2 is 0.029791099000908264, 0.06730462519936205 -
overall cost is 0.0020050787525332197
0 cluster length is 243
1 cluster length is 59
2 cluster length is 2
3 cluster length is 55
4 cluster length is 755
5 cluster length is 9
6 cluster length is 10
7 cluster length is 8
8 cluster length is 125
9 cluster length is 98
10 cluster length is 12
11 cluster length is 17
12 cluster length is 230
13 cluster length is 27
14 cluster length is 155
```

In [19]:

```
plt.plot(overall_cost_1)
plt.plot(overall_cost_2)
plt.xlabel('clusters')
plt.ylabel('cost')
```

Out[19]:

```
Text(0, 0.5, 'cost')
```

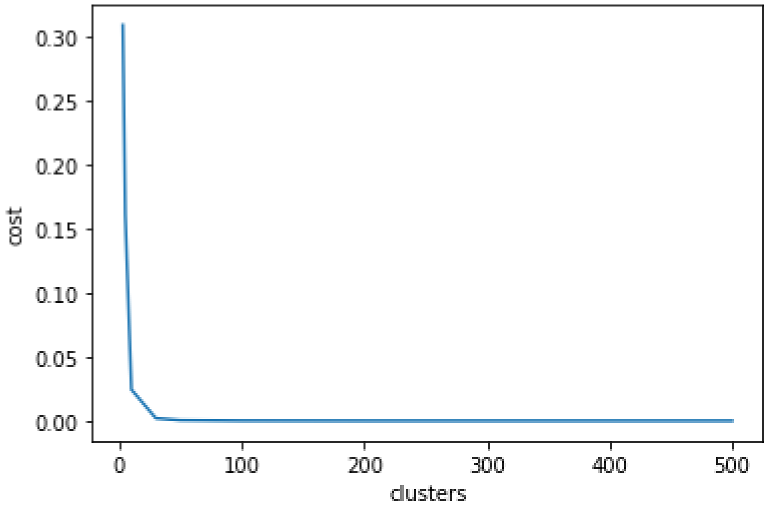


In [20]:

```
plt.plot(clusters,overall_cost)
plt.xlabel('clusters')
plt.ylabel('cost')
```

Out[20]:

```
Text(0, 0.5, 'cost')
```



In [21]:

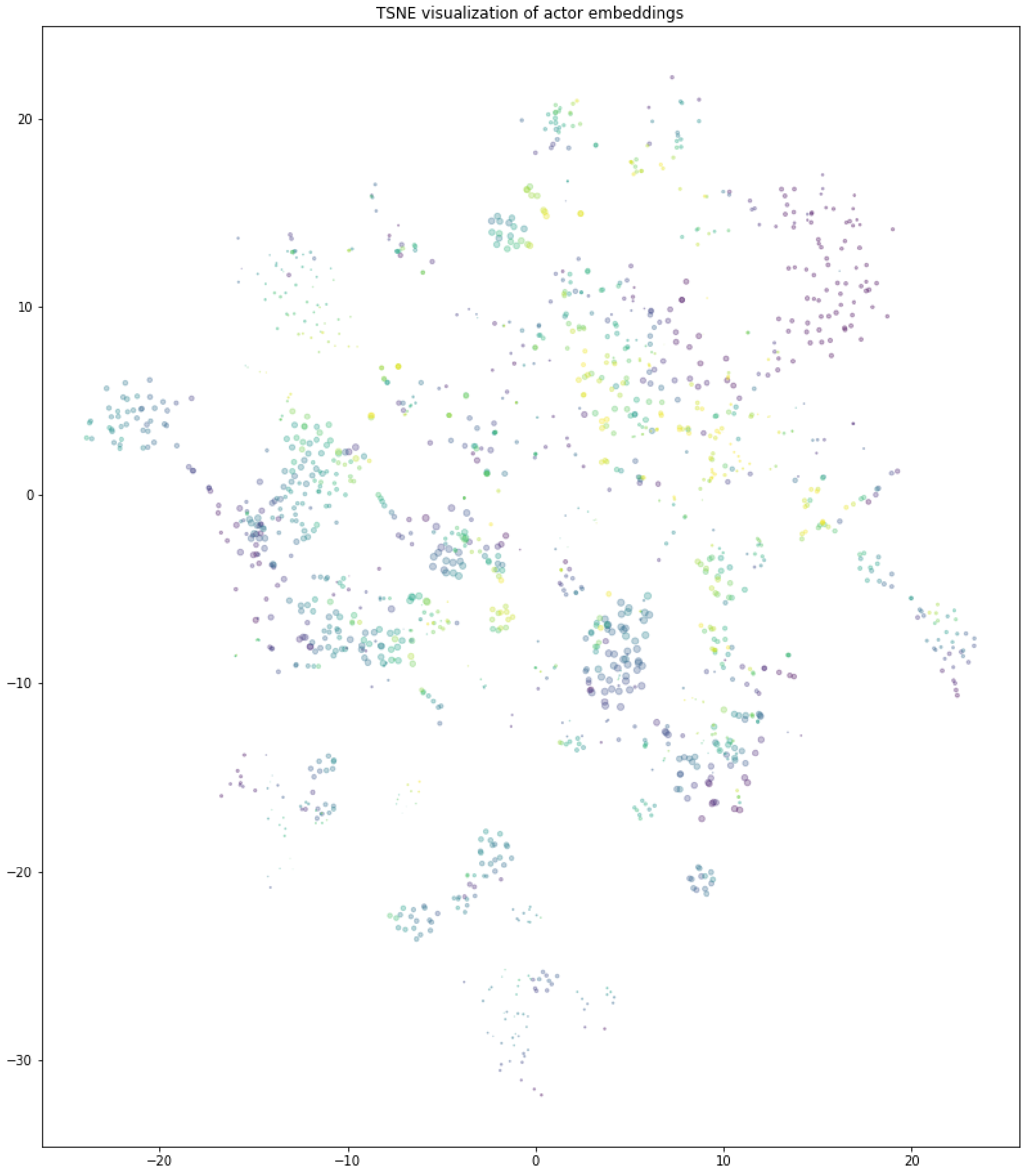
```
print("actor nodes best cost is {} for {} clusters ".format(overall_cost[0],clusters
```

```
actor nodes best cost is 0.3085745926277488 for 3 clusters
```

In [22]:

```
#this for plotting the tsne on actor nodes
from sklearn.manifold import TSNE
import numpy as np
# this code is refrenced from aaic refrence notebook
transform = TSNE #PCA
trans = transform(n_components=3)
node_embeddings_2d = trans.fit_transform(get_actor_model_records)
label_map = { l: i for i, l in enumerate(np.unique(get_actor_model_target_ids))}
node_colours = [ label_map[target] for target in get_actor_model_target_ids]
plt.figure(figsize=(20,16))
plt.axes().set(aspect="equal")
plt.scatter(node_embeddings_2d[:,0],
            node_embeddings_2d[:,1],
            node_embeddings_2d[:,2],
            c=node_colours, alpha=0.3)
plt.title('{} visualization of actor embeddings'.format(transform.__name__))

plt.show()
```



Movies clustering

In [23]:

```
from sklearn.cluster import KMeans
import numpy as np
overall_cost_1 = []
overall_cost_2 = []
overall_cost   = []
get_movie_model_target_ids = [ele_indx   for ele_indx,each_index in enumerate(get_cl
get_movie_model_records   = get_clustering_model_vectors[get_movie_model_target_ids]
#get_movie_model_records.shape
clusters = [3, 5, 10, 30, 50, 100, 200, 500]
for i in clusters:
    #kmeans model applying
    X = get_movie_model_records
    kmeans = KMeans(n_clusters=i, random_state=0).fit(X)

    #mapping the nodes as keys
    list_of_all_clusters = []
    unique_labels = np.unique(kmeans.labels_)
    dict_of_movie_nodes = {get_clustering_model_ids[indx]:item for indx,item in zip(
    for each_unique_cluster in unique_labels:
        current_cluster = []
        #print(each_unique_cluster)
        for node_id,cluster_number in dict_of_movie_nodes.items():
            if cluster_number == each_unique_cluster:
                current_cluster.append(node_id)
        print("{} cluster length is {}".format(str(each_unique_cluster),str(len(curr
        list_of_all_clusters.append(current_cluster)

    # finally calculating cost-1 & cost-2
    cost1 = 0
    cost2 = 0
    total_clusters = len(list_of_all_clusters)
    for each_cluster in list_of_all_clusters:
        G = nx.Graph()
        cost_1_sub_graph_ratio = 0
        cost_2_sub_graph_ratio = 0
        for each_node in each_cluster:
            #print(each_node)
            ego_subgrapgh_object = nx.ego_graph(B,each_node)
            G.add_nodes_from(ego_subgrapgh_object.nodes())
            G.add_edges_from(ego_subgrapgh_object.edges())
        ''' for calculating cost one '''
        #largest connected components
        largest_connect_components =  max(nx.connected_component_subgraphs(G), key=l
        largest_connected_nodes    =  len(largest_connect_components.nodes())
        # total nodes in sub-graph
        total_nodes_in_subgraph    =  nx.number_of_nodes(G)
        cost_1_sub_graph_ratio     +=  largest_connected_nodes/total_nodes_in_subgra

        ''' for calculating cost two '''
        total_edges_in_subgraph    =  nx.number_of_edges(G)
        total_right_side_nodes     =  return_right_side_nodes_count(G.nodes(),each_c
        cost_2_sub_graph_ratio     +=  total_edges_in_subgraph/total_right_side_node
        print(cost_1_sub_graph_ratio, cost_2_sub_graph_ratio)
    cost1 = cost_1_sub_graph_ratio/total_clusters
    cost2 = cost_2_sub_graph_ratio/total_clusters
    overall_cost_1.append(cost1)
    overall_cost_2.append(cost2)
    overall_cost.append(cost1*cost2)
    print("Final cost1 and cost 2 is {}, {} - overall {}".format(cost1, cost2,overal
```

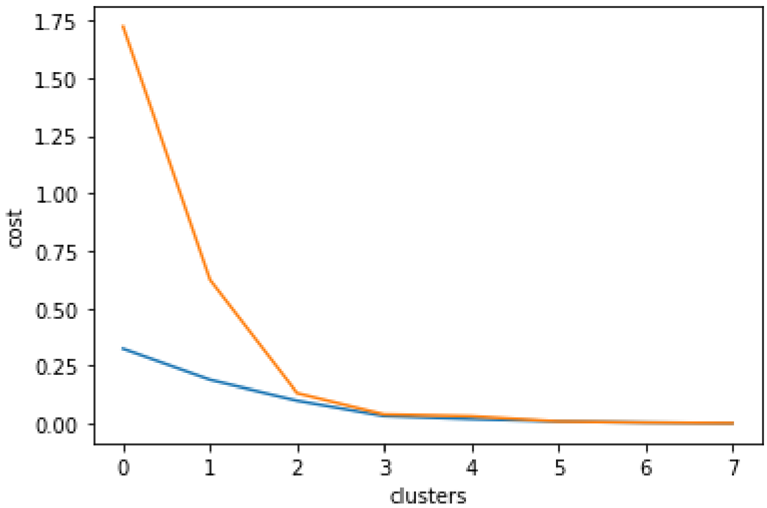
```
0 cluster length is 269
1 cluster length is 230
2 cluster length is 793
1.0 1.9315068493150684
0.9746168582375478 1.3439181916038752
0.9724137931034482 5.171066525871172
Final cost1 and cost 2 is 0.32413793103448274, 1.7236888419570573 - ov
erall [0.5587129349791841]
0 cluster length is 228
1 cluster length is 327
2 cluster length is 141
3 cluster length is 69
4 cluster length is 527
1.0 1.8644906033630069
1.0 5.745011086474501
0.9416846652267818 1.2459935897435896
1.0 1.346938775510204
0.9468838526912181 3.119774011299435
Final cost1 and cost 2 is 0.18937677053824362, 0.6239548022598871 - ov
erall [0.5587129349791841, 0.1181625454128058]
```

In [24]:

```python
plt.plot(overall_cost_1)
plt.plot(overall_cost_2)
plt.xlabel('clusters')
plt.ylabel('cost')
```

Out[24]:

```
Text(0, 0.5, 'cost')
```

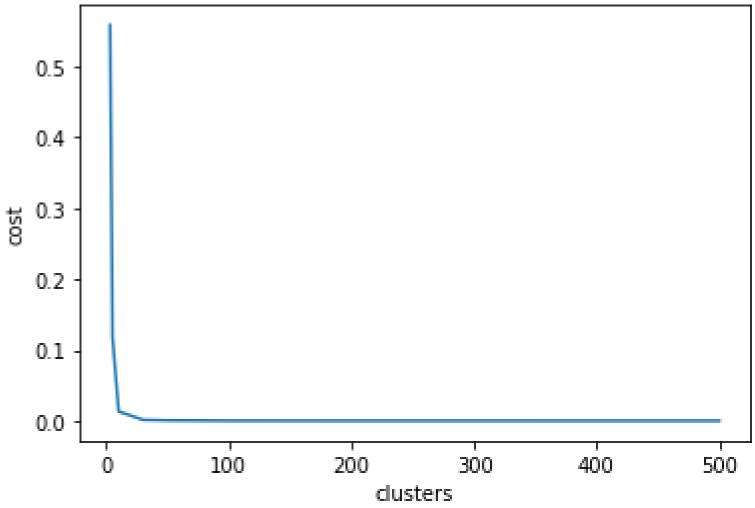


In [25]:

```
plt.plot(clusters,overall_cost)
plt.xlabel('clusters')
plt.ylabel('cost')
```

Out[25]:

```
Text(0, 0.5, 'cost')
```



In [26]:

```
print("movies nodes best cost is {} for {} clusters ".format(overall_cost[0],cluster
```

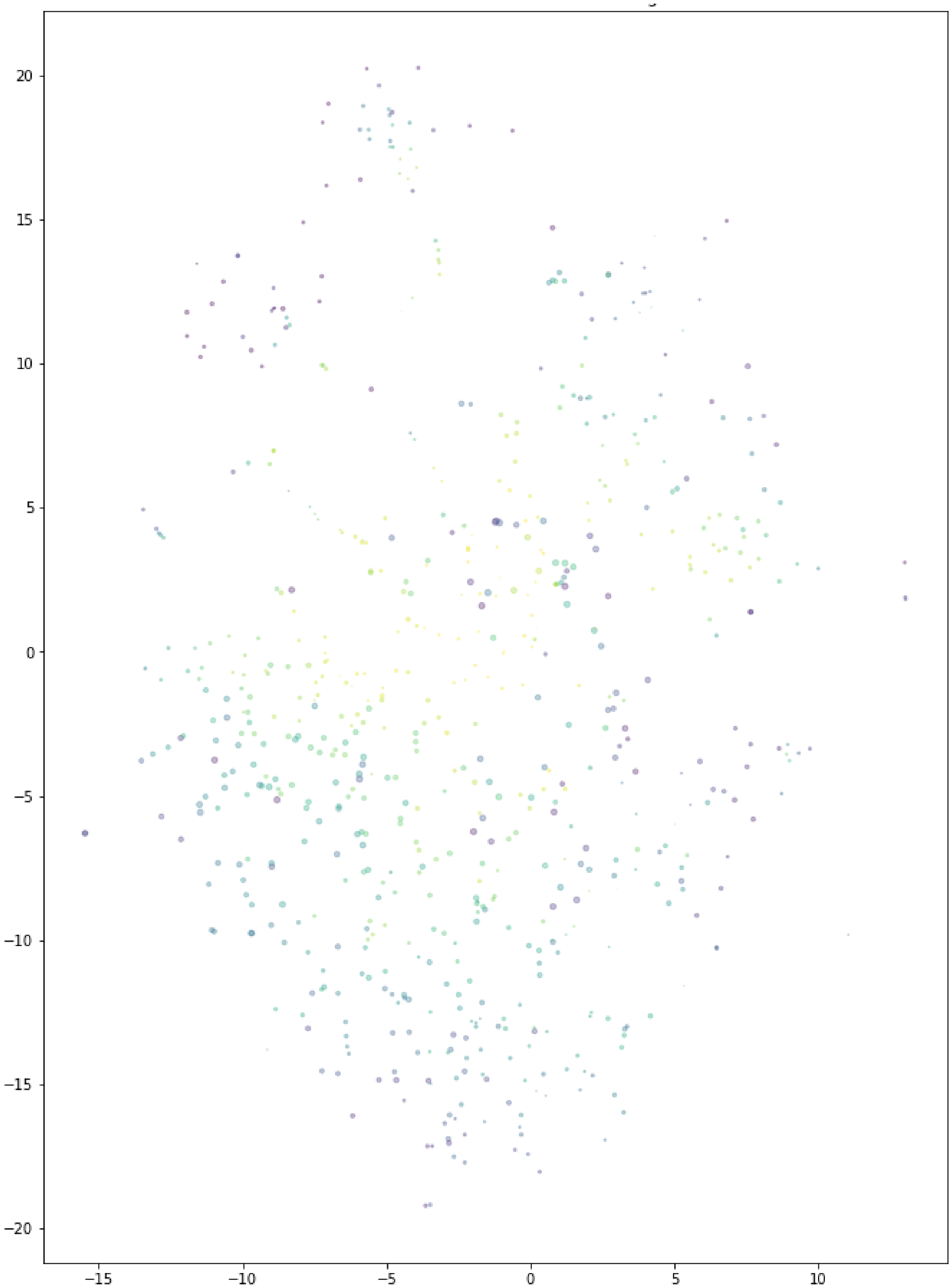
```
movies nodes best cost is 0.5587129349791841 for 3 clusters
```

In [27]:

```
#this for plotting the tsne on actor nodes
from sklearn.manifold import TSNE
import numpy as np
# this code is refrenced from aaic refrence notebook
transform = TSNE #PCA
trans = transform(n_components=3)
node_embeddings_2d = trans.fit_transform(get_movie_model_records)
label_map = { l: i for i, l in enumerate(np.unique(get_movie_model_target_ids))}
node_colours = [ label_map[target] for target in get_movie_model_target_ids]
plt.figure(figsize=(20,16))
plt.axes().set(aspect="equal")
plt.scatter(node_embeddings_2d[:,0],
            node_embeddings_2d[:,1],
            node_embeddings_2d[:,2],
            c=node_colours, alpha=0.3)
plt.title('{} visualization of actor embeddings'.format(transform.__name__))

plt.show()
```

TSNE visualization of actor embeddings



20
15
10
5
0
−5
−10
−15
−20
−15
−10
−5
0
5
10