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 [ ]:

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_rem
ount=True).
```

In [2]:

```
data=pd.read_csv("/content/drive/MyDrive/Clustering Assignment/movie_actor_network.csv", index_col=False, names=['movi
e','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]:

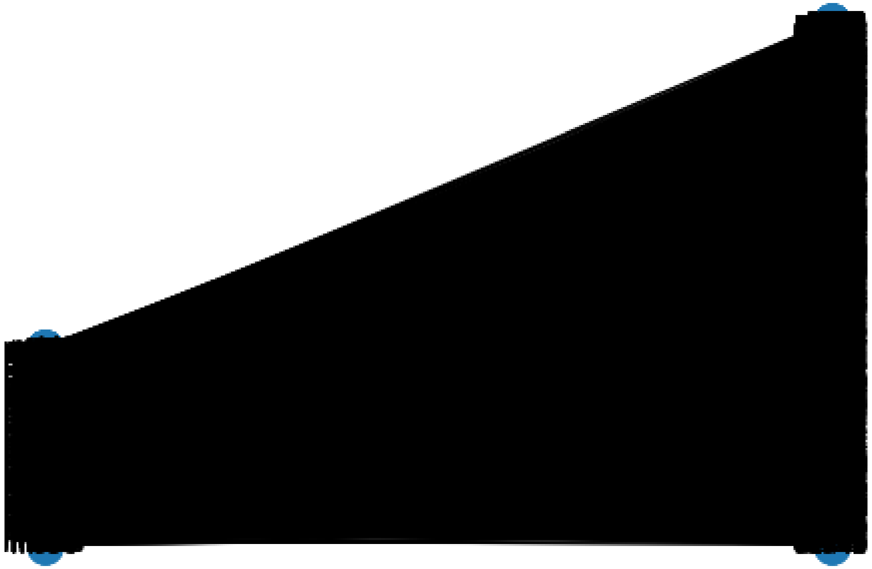
```
A = list(nx.connected_component_subgraphs(B))[0]
```

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 [7]:
```
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 [8]:
```
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 [9]:

```
# 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 [10]:

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

In [11]:

```
model.wv.vectors.shape
```

Out[11]:

```
(4703, 128)
```

In [12]:

```
# 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]
```

In [13]:

```
print(node_ids[0:15])
```

```
['a973', 'a967', 'a964', 'a1731', 'a970', 'a969', 'a1028', 'a1057', 'a965', 'a1003', 'm1094', 'a959', 'm1100', 'a96
6', 'a988']
```

In [14]:

```
print(node_targets[0:15])
```

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

In [21]:

```
def data_split(node_ids,node_targets,node_embeddings):

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

    return actor_nodes,movie_nodes,actor_embeddings,movie_embeddings
```

In [22]:

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

Grader function - 1

In [23]:

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

Out[23]:

```
True
```

Grader function - 2

In [24]:

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

Out[24]:

```
True
```

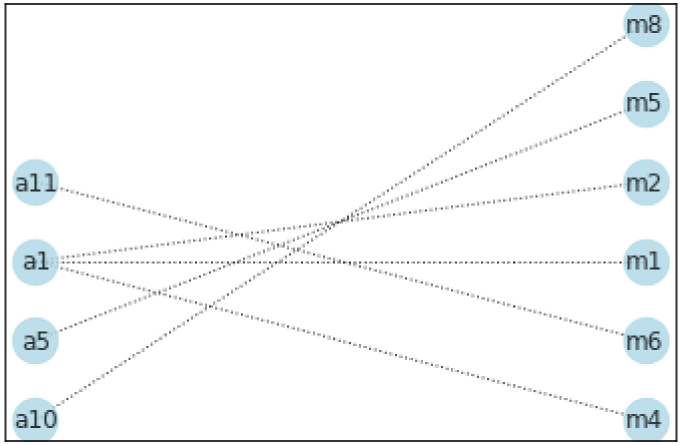
Calculating cost1

In [25]:

```
def cost1(graph,number_of_clusters):


    components = [comp for comp in nx.connected_components(graph)]
    component_size = [len(comp) for comp in components]

    no_nodes_largestCC=max(component_size)

    total_nodes_incluster=graph.number_of_nodes()


    div= (no_nodes_largestCC/total_nodes_incluster)

    cost1=div/number_of_clusters



    return cost1
```

In [26]:

```
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 [27]:

```
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[27]:

```
True
```

Calculating cost2

```
In [28]: def cost2(graph,number_of_clusters):



             degree=[graph.degree(j) for j in graph.nodes() if 'a' in j  ]

             sum_of_degree=np.sum(degree)

             m=[k for k in graph.nodes() if 'm' in k ]
             movies_nodes=len(np.unique(m))



             c2=(sum_of_degree/movies_nodes)

             cost2=(c2/number_of_clusters)



             return cost2
```

Grader function - 4

```
In [29]: 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[29]: True
```

Task 1 : Apply clustering algorithm to group similar actors

Grouping similar actors

In [30]:

```
#get corresponding actor nodes for labels
def getClusterPoints(V, labels):
    clusters = {}
    for l in range(0, max(labels)+1):
        data_points = []
        indices = [i for i, x in enumerate(labels) if x == l]
        for idx in indices:
            data_points.append(V[idx])
        clusters[l] = data_points
    return clusters
```

In [31]:

```
act_embed=np.vstack(actor_embeddings)
from sklearn.cluster import KMeans
metric_cost=[]
list=[3, 5, 10, 30, 50, 100, 200, 500]
for no_of_cluster in list:
  algo=KMeans(n_clusters=no_of_cluster,random_state=5)
  algo.fit(act_embed)
  labels=algo.labels_
  cluster_actor=getClusterPoints(actor_nodes,labels)
  cost11=[]
  cost22=[]
  for j in range(0,no_of_cluster):
    G=nx.Graph()
    for act in cluster_actor[j]:
      sub_graph=nx.ego_graph(B,act)
      G.add_nodes_from(sub_graph.nodes)
      G.add_edges_from(sub_graph.edges())
    cost11.append(cost1(G,no_of_cluster))
    cost22.append(cost2(G,no_of_cluster))
  metric_cost.append((np.sum(cost11))*(np.sum(cost22)))

print("Maximum metric cost is : ",max(metric_cost))
index=metric_cost.index(max(metric_cost))
print("Optimum number of cluster is : ",list[index])
```

```
Maximum metric cost is :  3.713034878683522
Optimum number of cluster is :  3
```

Displaying similar actor clusters

In [32]:

```
#Running KMeans with cluster=3 to get labels
algo=KMeans(n_clusters=3,random_state=5)
algo.fit(act_embed)
labels=algo.labels_
```

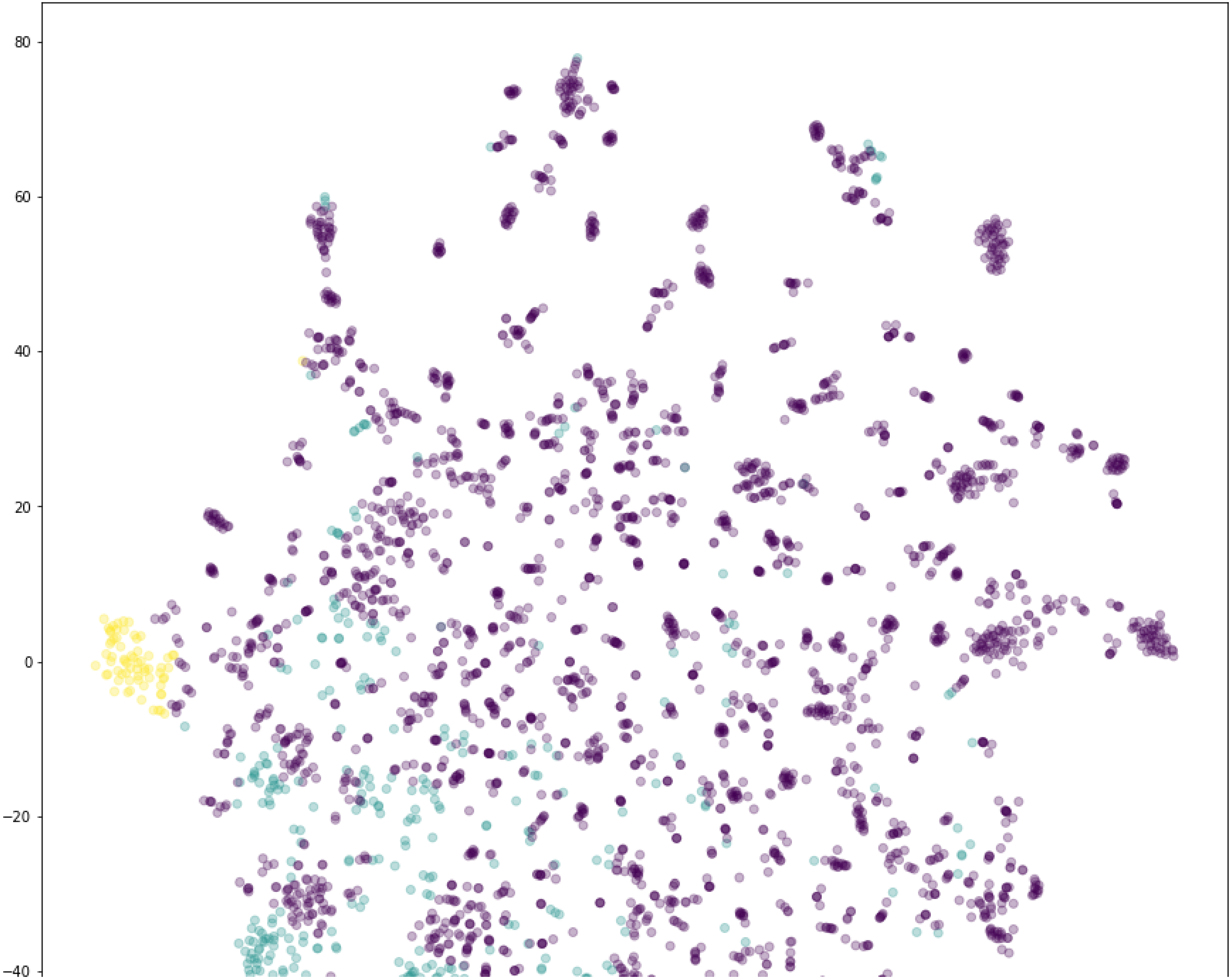
In [33]:

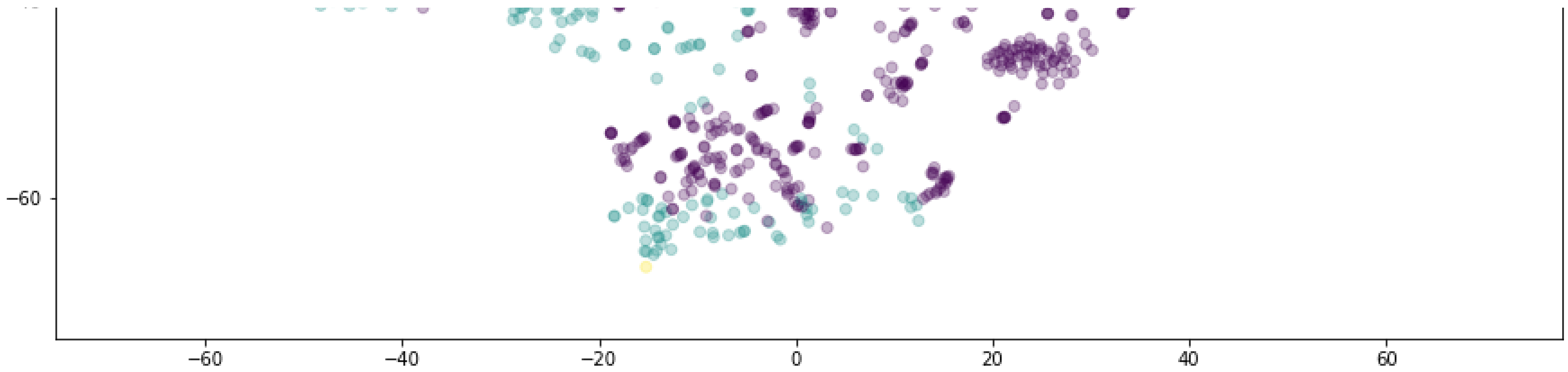
```
#Converting vector to 2d vector using TSNE
from sklearn.manifold import TSNE
transform = TSNE

trans = transform(n_components=2)
actor_embeddings_2d = trans.fit_transform(actor_embeddings)
```

In [34]:

```
#Plotting scatter plot to display actors
import numpy as np
label_map = { l: i for i, l in enumerate(np.unique(labels))}
node_colours = [ label_map[target] for target in labels]
plt.figure(figsize=(20,16))
plt.axes().set(aspect="equal")
plt.scatter(actor_embeddings_2d[:,0],
            actor_embeddings_2d[:,1],
            c=node_colours, alpha=0.3)
plt.title('{} visualization of actor embeddings'.format(transform.__name__))

plt.show()
```

TSNE visualization of actor embeddings





Task 2 : Apply clustering algorithm to group similar movies

Calculating cost1

In [35]:

```
def cost1(graph,number_of_clusters):


    components = [comp for comp in nx.connected_components(graph)]
    component_size = [len(comp) for comp in components]

    no_nodes_largestCC=max(component_size)

    total_nodes_incluster=graph.number_of_nodes()


    div= (no_nodes_largestCC/total_nodes_incluster)

    cost1=div/number_of_clusters



    return cost1
```

Calculating cost1

```
In [36]: def cost2(graph,number_of_clusters):



             degree=[graph.degree(j) for j in graph.nodes() if 'm' in j  ]

             sum_of_degree=np.sum(degree)

             a=[k for k in graph.nodes() if 'a' in k ]
             actors_nodes=len(np.unique(a))



             c2=(sum_of_degree/actors_nodes)

             cost2=(c2/number_of_clusters)



             return cost2
```

Grouping similar movies

```
In [37]: #get corresponding movie nodes for labels
         def getClusterPoints(V, labels):
             clusters = {}
             for l in range(0, max(labels)+1):
                 data_points = []
                 indices = [i for i, x in enumerate(labels) if x == l]
                 for idx in indices:
                     data_points.append(V[idx])
                 clusters[l] = data_points
             return clusters
```

In [56]:

```
movie_embed=np.vstack(movie_embeddings)
from sklearn.cluster import KMeans
metric_cost=[]
list=[3, 5, 10, 30, 50, 100, 200, 500]
for no_of_cluster in list:
  algo=KMeans(n_clusters=no_of_cluster,random_state=5)
  algo.fit(movie_embed)
  labels=algo.labels_
  cluster_movie=getClusterPoints(movie_nodes,labels)
  cost11=[]
  cost22=[]
  for j in range(0,no_of_cluster):
    G=nx.Graph()
    for mov in cluster_movie[j]:
      sub_graph=nx.ego_graph(B,mov)
      G.add_nodes_from(sub_graph.nodes)
      G.add_edges_from(sub_graph.edges())
    cost11.append(cost1(G,no_of_cluster))
    cost22.append(cost2(G,no_of_cluster))
  metric_cost.append((np.sum(cost11))*(np.sum(cost22)))

print("Maximum metric cost is : ",max(metric_cost))
index=metric_cost.index(max(metric_cost))
print("Optimum number of cluster is : ",list[index])
```

```
Maximum metric cost is :  3.085631247043874
Optimum number of cluster is :  3
```

Displaying similar movie clusters

In [61]:

```
#Running KMeans with cluster=3 to get labels
algo=KMeans(n_clusters=3,random_state=5)
algo.fit(movie_embed)
labels=algo.labels_
```

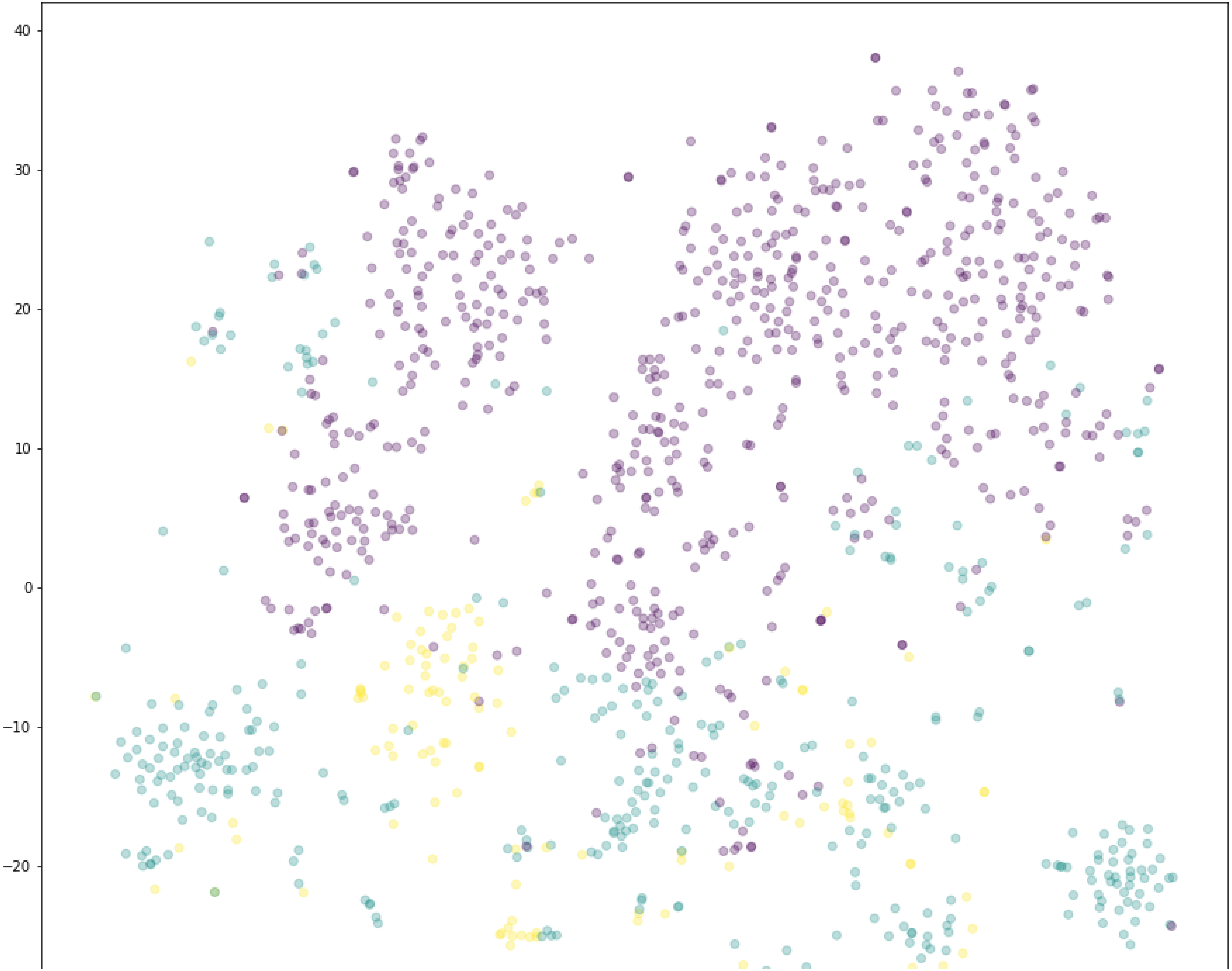
In [62]:

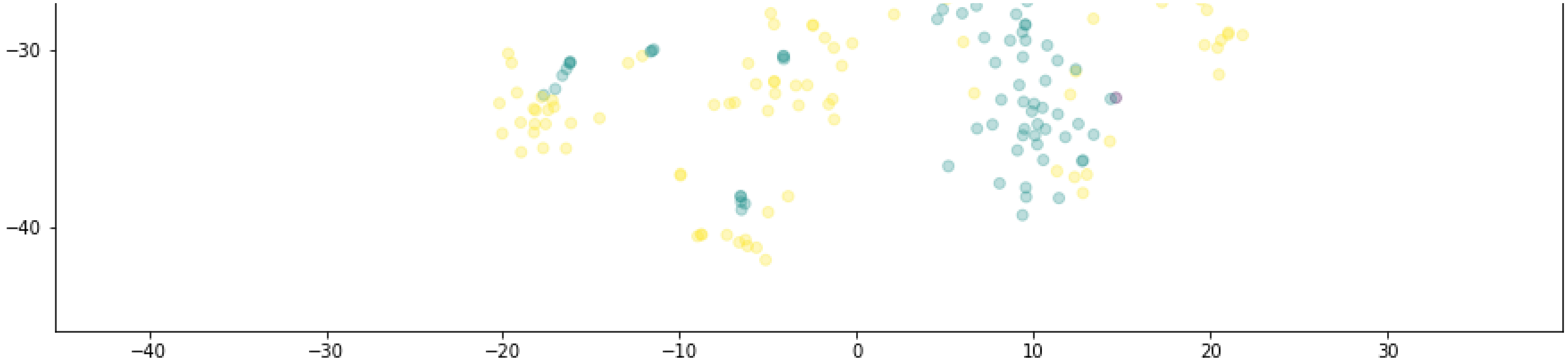
```
#Converting vector to 2d vector using TSNE
from sklearn.manifold import TSNE
transform = TSNE

trans = transform(n_components=2)
movie_embeddings_2d = trans.fit_transform(movie_embeddings)
```

In [63]:

```
#Plotting scatter plot to display movie
import numpy as np
label_map = { l: i for i, l in enumerate(np.unique(labels))}
node_colours = [ label_map[target] for target in labels]
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 movie embeddings'.format(transform.__name__))

plt.show()
```

TSNE visualization of movie embeddings





In [ ]:
```
!jupyter nbconvert --to html Clustering_solve.ipynb
```