Clustering_Assignment_practice

January 13, 2021

1 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())

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

3 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 $Cost1 * Cost2$

   $\text{Cost1} = \frac{1}{N} \sum_{\text{each cluster i}} \frac{\text{(number of nodes in the largest connected component in the graph with the movie nodes and its actor neighbours i}}{\text{(total number of nodes in that cluster i)}}$
   where N= number of clusters (Write your code in def cost1())

3. $\text{Cost2} = \frac{1}{N} \sum_{\text{each cluster i}} \frac{\text{(sum of degress of movie nodes in the graph with the movie nodes and its actor neighbours in cluster i)}}{\text{(number of unique actor nodes in the graph with the movie nodes and its actor neighbours in cluster i)}}$
   where N= number of clusters (Write your code in def cost2())

Algorithm for actor nodes

[1]:
```
def compute_custom_cost(cls,B):
  clusters = list(cls.keys())
  c1,c2=0,0
  for i in clusters:
    a = [actor_nodes[j] for j in cls[i]]
    ll = list()
    for k in a:
      t = list(nx.ego_graph(B,k).edges)
      for ind,val in enumerate(t):
        if 'm' in val[0]:
          ll.append(tuple(reversed(t[ind])))
        else:
          ll.append(t[ind])
    d1 = pd.DataFrame(ll,columns=['actors','movies'])
    edgess = [tuple(x) for x in d1.values.tolist()]
    C = nx.Graph()
    C.add_nodes_from(d1['movies'].unique(), bipartite=0, label='movies')
    C.add_nodes_from(d1['actors'].unique(), bipartite=1, label='actors')
    C.add_edges_from(edgess, label='acted')
    c1 += cost1(C,len(clusters))
    c2 += cost2(C,len(clusters))
  res = c1*c2
  return res
```

[2]:
```
def plot_TSNE(node_embeddings_2d,labels):
  import numpy as np
# draw the points
  node_targets = labels
  label_map = { l: i for i, l in enumerate(np.unique(node_targets))}
  node_colours = [ label_map[target] for target in node_targets]

  plt.figure(figsize=(10,10))
  plt.axes().set(aspect="equal")
  plt.scatter(node_embeddings_2d[:,0],
              node_embeddings_2d[:,1],
              c=node_colours, alpha=0.3)

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

```
    return
```

[3]:
```
def plot_scatter(q):
    x = q['x']
    y = q['y']
    Cluster = q['labels']
    fig = plt.figure()
    ax = fig.add_subplot(111)
    scatter = ax.scatter(x,y,c=Cluster,s=50)

    ax.set_xlabel('x')
    ax.set_ylabel('y')
    plt.colorbar(scatter)
    fig.show()
    return
```

[4]:
```
!pip install networkx==2.3
```

```
Requirement already satisfied: networkx==2.3 in /usr/local/lib/python3.6/dist-
packages (2.3)
Requirement already satisfied: decorator>=4.3.0 in
/usr/local/lib/python3.6/dist-packages (from networkx==2.3) (4.4.2)
```

[5]:
```
!pip install stellargraph
```

```
Requirement already satisfied: stellargraph in /usr/local/lib/python3.6/dist-
packages (1.2.1)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.6/dist-
packages (from stellargraph) (2.3)
Requirement already satisfied: tensorflow>=2.1.0 in
/usr/local/lib/python3.6/dist-packages (from stellargraph) (2.4.0)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.6/dist-
packages (from stellargraph) (1.1.5)
Requirement already satisfied: numpy>=1.14 in /usr/local/lib/python3.6/dist-
packages (from stellargraph) (1.19.5)
Requirement already satisfied: matplotlib>=2.2 in /usr/local/lib/python3.6/dist-
packages (from stellargraph) (3.2.2)
Requirement already satisfied: scikit-learn>=0.20 in
/usr/local/lib/python3.6/dist-packages (from stellargraph) (0.22.2.post1)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.6/dist-
packages (from stellargraph) (1.4.1)
Requirement already satisfied: gensim>=3.4.0 in /usr/local/lib/python3.6/dist-
packages (from stellargraph) (3.6.0)
Requirement already satisfied: decorator>=4.3.0 in
/usr/local/lib/python3.6/dist-packages (from networkx>=2.2->stellargraph)
(4.4.2)
Requirement already satisfied: absl-py~=0.10 in /usr/local/lib/python3.6/dist-
packages (from tensorflow>=2.1.0->stellargraph) (0.10.0)
```

```
Requirement already satisfied: h5py~=2.10.0 in /usr/local/lib/python3.6/dist-
packages (from tensorflow>=2.1.0->stellargraph) (2.10.0)
Requirement already satisfied: flatbuffers~=1.12.0 in
/usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargraph)
(1.12)
Requirement already satisfied: wheel~=0.35 in /usr/local/lib/python3.6/dist-
packages (from tensorflow>=2.1.0->stellargraph) (0.36.2)
Requirement already satisfied: google-pasta~=0.2 in
/usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargraph)
(0.2.0)
Requirement already satisfied: astunparse~=1.6.3 in
/usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargraph)
(1.6.3)
Requirement already satisfied: opt-einsum~=3.3.0 in
/usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargraph)
(3.3.0)
Requirement already satisfied: six~=1.15.0 in /usr/local/lib/python3.6/dist-
packages (from tensorflow>=2.1.0->stellargraph) (1.15.0)
Requirement already satisfied: tensorboard~=2.4 in
/usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargraph)
(2.4.0)
Requirement already satisfied: tensorflow-estimator<2.5.0,>=2.4.0rc0 in
/usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargraph)
(2.4.0)
Requirement already satisfied: wrapt~=1.12.1 in /usr/local/lib/python3.6/dist-
packages (from tensorflow>=2.1.0->stellargraph) (1.12.1)
Requirement already satisfied: grpcio~=1.32.0 in /usr/local/lib/python3.6/dist-
packages (from tensorflow>=2.1.0->stellargraph) (1.32.0)
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.6/dist-
packages (from tensorflow>=2.1.0->stellargraph) (3.12.4)
Requirement already satisfied: keras-preprocessing~=1.1.2 in
/usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargraph)
(1.1.2)
Requirement already satisfied: termcolor~=1.1.0 in
/usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargraph)
(1.1.0)
Requirement already satisfied: gast==0.3.3 in /usr/local/lib/python3.6/dist-
packages (from tensorflow>=2.1.0->stellargraph) (0.3.3)
Requirement already satisfied: typing-extensions~=3.7.4 in
/usr/local/lib/python3.6/dist-packages (from tensorflow>=2.1.0->stellargraph)
(3.7.4.3)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-
packages (from pandas>=0.24->stellargraph) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.6/dist-packages (from pandas>=0.24->stellargraph) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/usr/local/lib/python3.6/dist-packages (from matplotlib>=2.2->stellargraph)
(2.4.7)
```

```
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-
packages (from matplotlib>=2.2->stellargraph) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.6/dist-packages (from matplotlib>=2.2->stellargraph)
(1.3.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-
packages (from scikit-learn>=0.20->stellargraph) (1.0.0)
Requirement already satisfied: smart-open>=1.2.1 in
/usr/local/lib/python3.6/dist-packages (from gensim>=3.4.0->stellargraph)
(4.1.0)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.6/dist-packages (from
tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (2.23.0)
Requirement already satisfied: google-auth<2,>=1.6.3 in
/usr/local/lib/python3.6/dist-packages (from
tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (1.17.2)
Requirement already satisfied: setuptools>=41.0.0 in
/usr/local/lib/python3.6/dist-packages (from
tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (51.1.1)
Requirement already satisfied: werkzeug>=0.11.15 in
/usr/local/lib/python3.6/dist-packages (from
tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (1.0.1)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in
/usr/local/lib/python3.6/dist-packages (from
tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (1.7.0)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in
/usr/local/lib/python3.6/dist-packages (from
tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (0.4.2)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/dist-
packages (from tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (3.3.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-
packages (from
requests<3,>=2.21.0->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.6/dist-packages (from
requests<3,>=2.21.0->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.6/dist-packages (from
requests<3,>=2.21.0->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph)
(2020.12.5)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.6/dist-packages (from
requests<3,>=2.21.0->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (3.0.4)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.6/dist-packages (from google-
auth<2,>=1.6.3->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (0.2.8)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in
/usr/local/lib/python3.6/dist-packages (from google-
```

```
auth<2,>=1.6.3->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (4.2.0)
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3" in
/usr/local/lib/python3.6/dist-packages (from google-
auth<2,>=1.6.3->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (4.6)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.6/dist-packages (from google-auth-
oauthlib<0.5,>=0.4.1->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (1.3.0)
Requirement already satisfied: importlib-metadata; python_version < "3.8" in
/usr/local/lib/python3.6/dist-packages (from
markdown>=2.6.8->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (3.3.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in
/usr/local/lib/python3.6/dist-packages (from pyasn1-modules>=0.2.1->google-
auth<2,>=1.6.3->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.6/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: zipp>=0.5 in /usr/local/lib/python3.6/dist-
packages (from importlib-metadata; python_version <
"3.8"->markdown>=2.6.8->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph)
(3.4.0)
```

[6]:
```python
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
```

[7]:
```python
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_remount=True).
```

[8]:
```python
path = '/content/drive/MyDrive/AAIC/Assignments/14.Clustering on Graph Dataset/
 ↪Practice/movie_actor_network.csv'
data=pd.read_csv(path, index_col=False, names=['movie','actor'])
```

[9]:
```python
data.shape
```

[9]:
```
(9650, 2)
```

[10]:
```python
edges = [tuple(x) for x in data.values.tolist()]
```

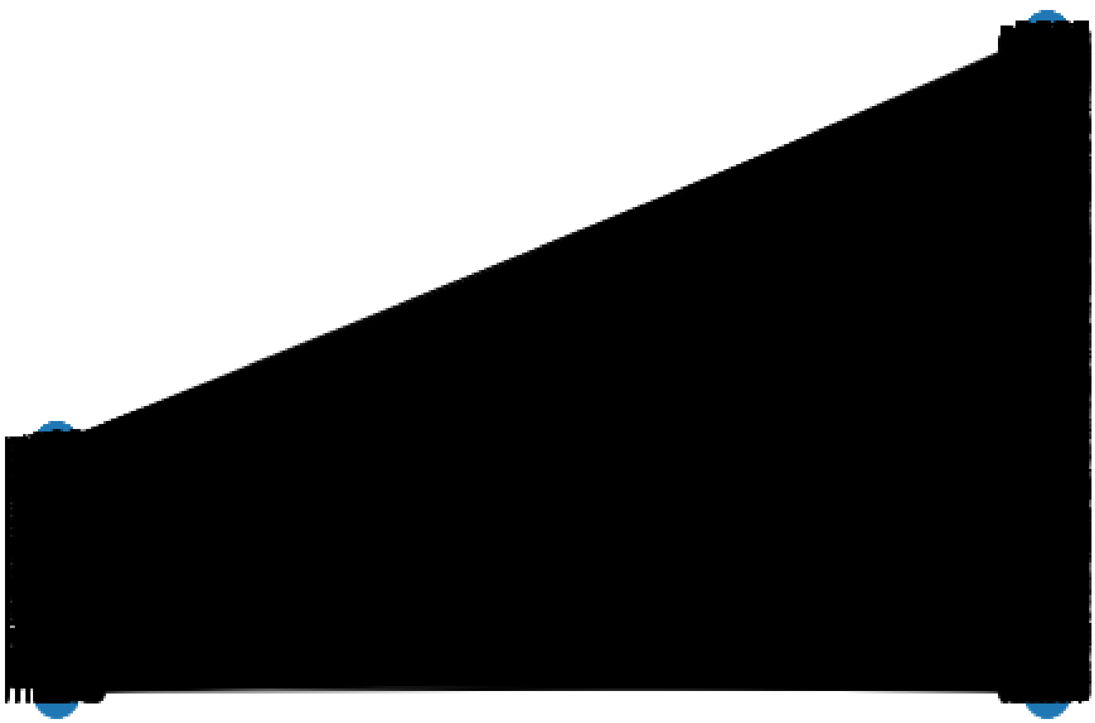
[11]:
```
B = nx.Graph()
B.add_nodes_from(data['movie'].unique(), bipartite=0, label='movie')
B.add_nodes_from(data['actor'].unique(), bipartite=1, label='actor')
B.add_edges_from(edges, label='acted')
```

[12]:
```
A = list(nx.connected_component_subgraphs(B))[0]
```

[13]:
```
print("number of nodes", A.number_of_nodes())
print("number of edges", A.number_of_edges())
```

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

[14]:
```
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()
```



[15]:
```
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
```

[16]:
```
# 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
```

[17]:
```
from gensim.models import Word2Vec
model = Word2Vec(walks, size=128, window=5)
```

[18]:
```
model.wv.vectors.shape  # 128-dimensional vector for each node in the graph
```

[18]:
```
(4703, 128)
```

[19]:
```
# 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]
```

image.png

[20]:
```
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␣
↪and movie embedding
    # By using node_ids and node_targets, we can extract actor_nodes and movie␣
↪nodes

    for ind,val in enumerate(node_ids):
      if 'a' in val:
        actor_nodes.append(val)
        actor_embeddings.append(node_embeddings[ind])
      if 'm' in val:
        movie_nodes.append(val)
        movie_embeddings.append(node_embeddings[ind])

    return actor_nodes,movie_nodes,actor_embeddings,movie_embeddings
```

[21]:
```
actor_nodes,movie_nodes,actor_embeddings,movie_embeddings =␣
↪data_split(node_ids,node_targets,node_embeddings)
```

Grader function - 1

[22]:
```
def grader_actors(data):
    assert(len(data)==3411)
    return True
grader_actors(actor_nodes)
```

[22]: True

Grader function - 2

[23]:
```
def grader_movies(data):
    assert(len(data)==1292)
    return True
grader_movies(movie_nodes)
```

[23]: True

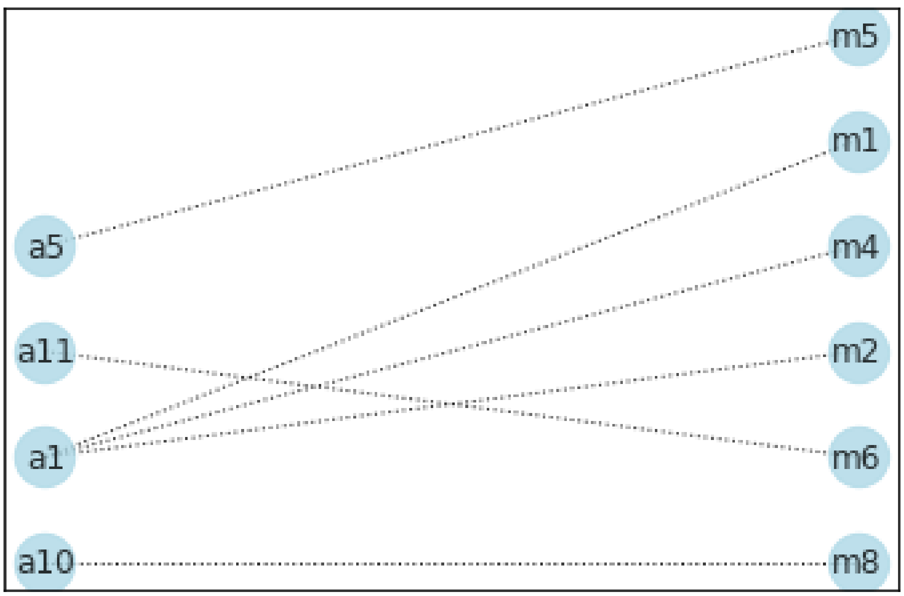
Calculating cost1

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

where N= number of clusters

[24]:
```
def cost1(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''
    g = list(nx.connected_component_subgraphs(graph))
    max_nodes = max([len(i.nodes()) for i in g])
    total_nodes = graph.number_of_nodes()
    cost1 = (1/number_of_clusters)*(max_nodes/total_nodes)
    return cost1
```

[25]:
```
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

[26]:
```
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)
```

[26]: 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

[27]:
```
def cost2(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''
    movie = list()
    d = dict(graded_graph.degree)
    for k,v in d.items():
      if 'm' in k:
        movie.append(k)

    p = list(nx.connected_component_subgraphs(graded_graph))
    actor = list()
    for i in p:
      deg = dict(i.degree)
      [actor.append(value) for key,value in deg.items() if 'a' in key]
    #print(sum(actor),len(movie),sum(actor)/len(movie))
    cost2 = (1/number_of_clusters)*(sum(actor)/len(movie))
    return cost2
```

Grader function - 4

[28]:
```
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)
```

[28]:
```
True
```

Grouping similar actors

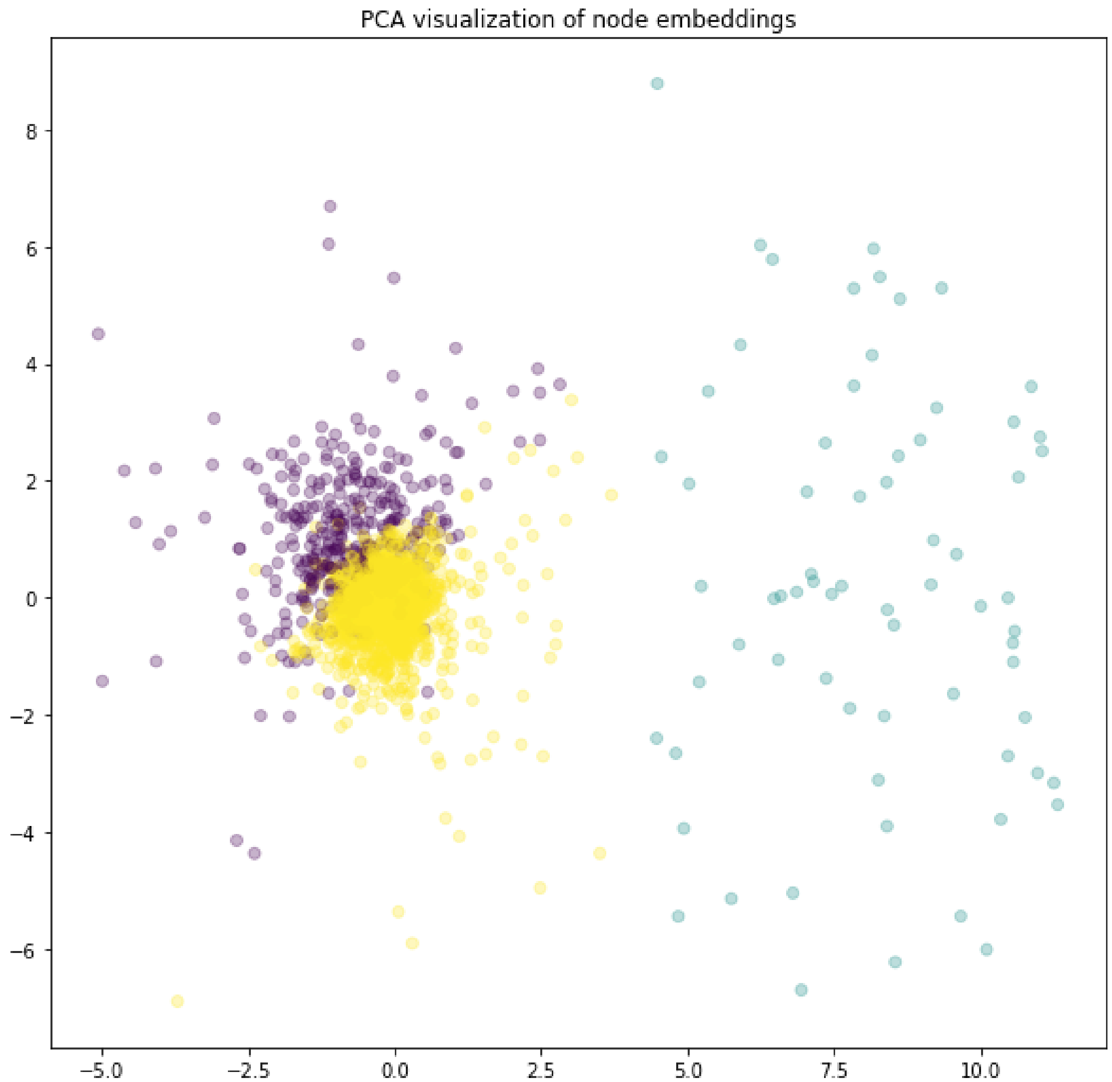
[34]:
```
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import SpectralClustering
from tqdm import tqdm_notebook as tqdm
p = [3, 5, 10, 30, 50, 100, 200, 500]
d = dict()
for number_of_clusters in p:
  algo = KMeans(n_clusters=number_of_clusters)
  algo.fit(actor_embeddings)
  #labels = (algo.labels_)
  cls = {i: np.where(algo.labels_ == i)[0] for i in range(algo.n_clusters)}
  d[number_of_clusters] = (compute_custom_cost(cls,B))

print('optimal number of clusters:',max(d, key = d.get))
final_model = KMeans(n_clusters=max(d, key = d.get))
final_model.fit(actor_embeddings)
```

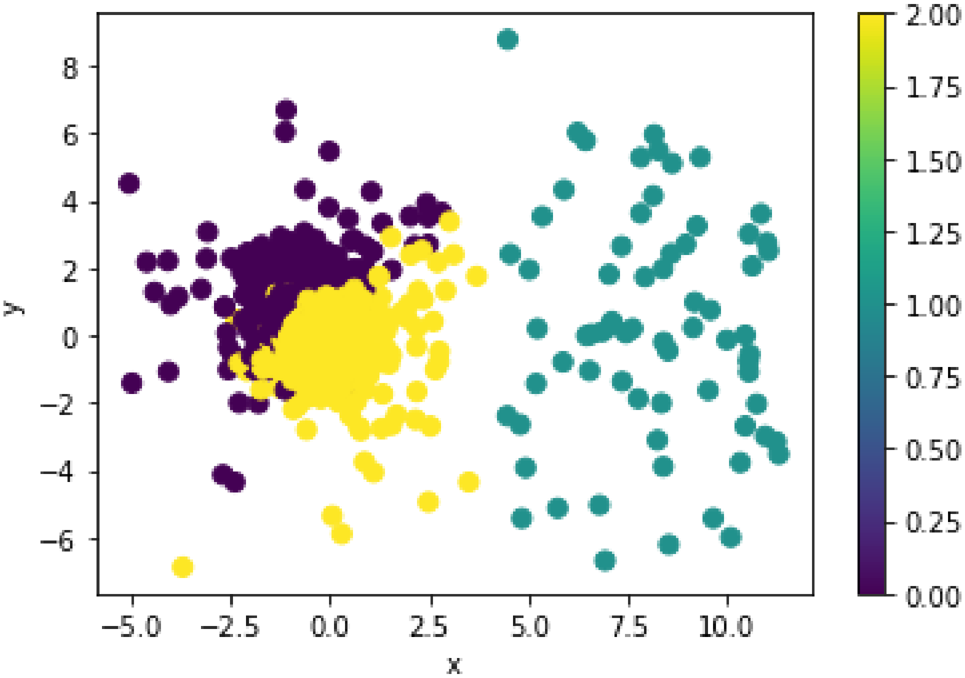
```
optimal number of clusters: 3
```

[34]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

Displaying similar actor clusters

[35]:
```
labels = (final_model.labels_)
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
transform = PCA #TSNE #
trans = transform(n_components=2)
node_embeddings_2d = trans.fit_transform(actor_embeddings)
plot_TSNE(node_embeddings_2d,labels)

q = pd.DataFrame(node_embeddings_2d,columns=['x','y'])
q['labels'] = final_model.labels_
plot_scatter(q)
```





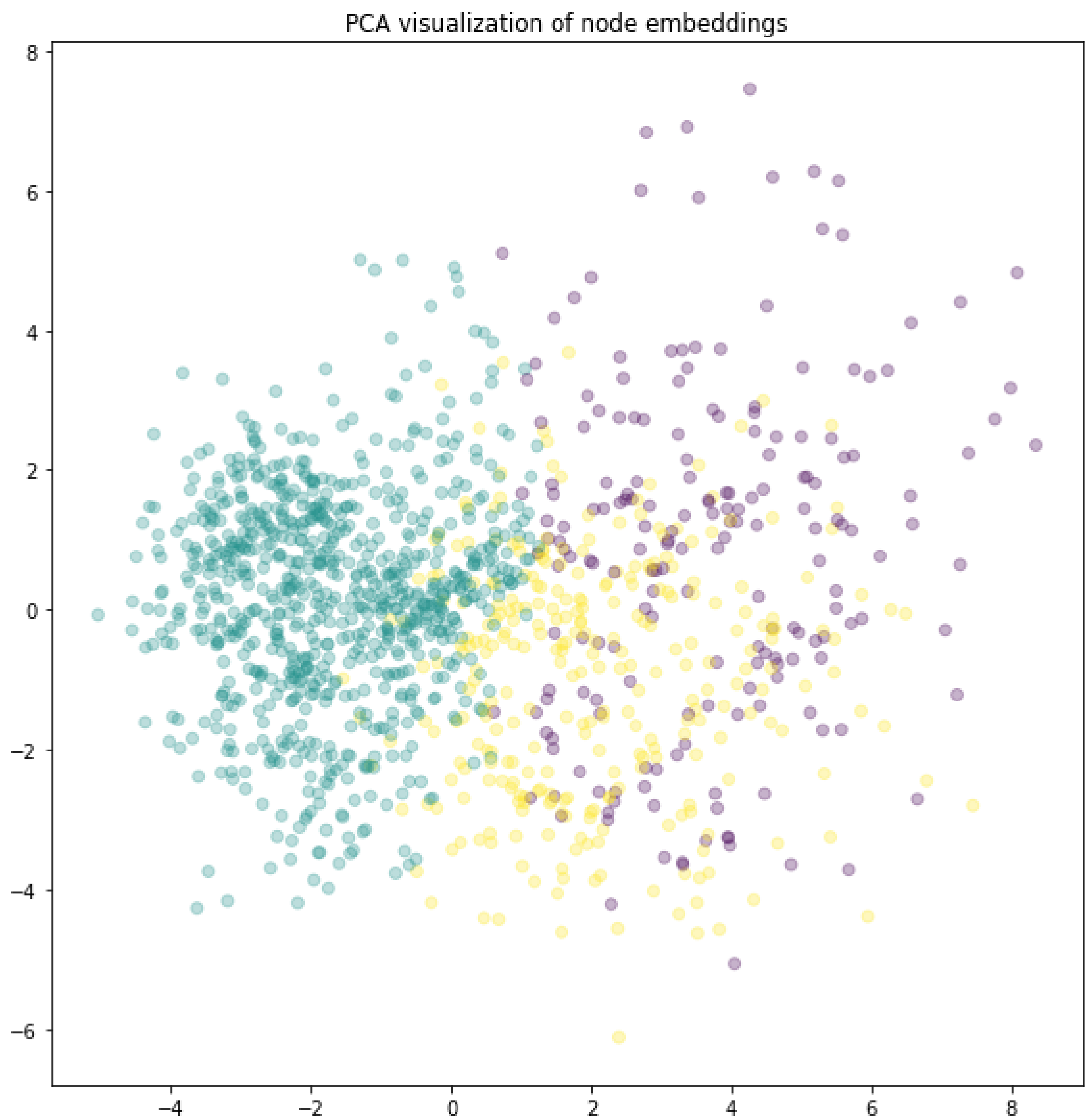
Grouping similar movies

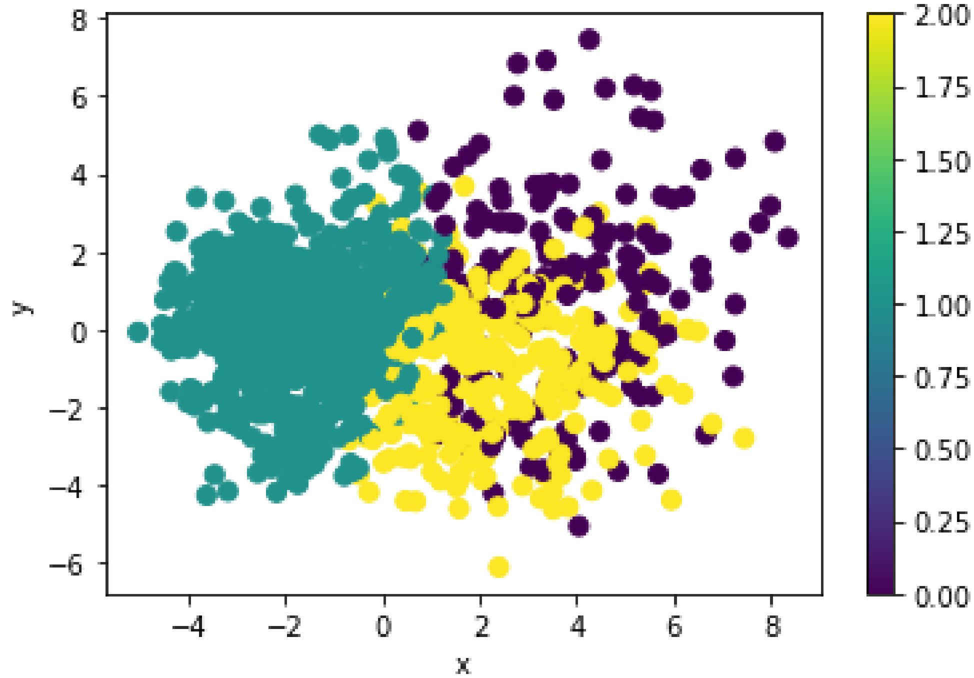
[36]:
```
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import SpectralClustering
from tqdm import tqdm_notebook as tqdm

p = [3, 5, 10, 30, 50, 100, 200, 500]
d = dict()
for number_of_clusters in p:
  algo = KMeans(n_clusters=number_of_clusters)
  algo.fit(movie_embeddings)
  #labels = (algo.labels_)
  cls = {i: np.where(algo.labels_ == i)[0] for i in range(algo.n_clusters)}
  d[number_of_clusters] = (compute_custom_cost(cls,B))

print('optimal number of clusters:',max(d, key = d.get))
final_model = KMeans(n_clusters=max(d, key = d.get), )
final_model.fit(movie_embeddings)
labels = (final_model.labels_)
cls_ = {i: np.where(final_model.labels_ == i)[0] for i in range(final_model.
 ↪n_clusters)}
```

```
optimal number of clusters: 3
```

Displaying similar movie clusters

[37]:
```
labels = (final_model.labels_)
from sklearn.manifold import TSNE
transform = PCA #TSNE
trans = transform(n_components=2)
node_embeddings_2d = trans.fit_transform(movie_embeddings)
plot_TSNE(node_embeddings_2d,labels)

q = pd.DataFrame(node_embeddings_2d,columns=['x','y'])
q['labels'] = final_model.labels_
plot_scatter(q)
```





[32]: