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.

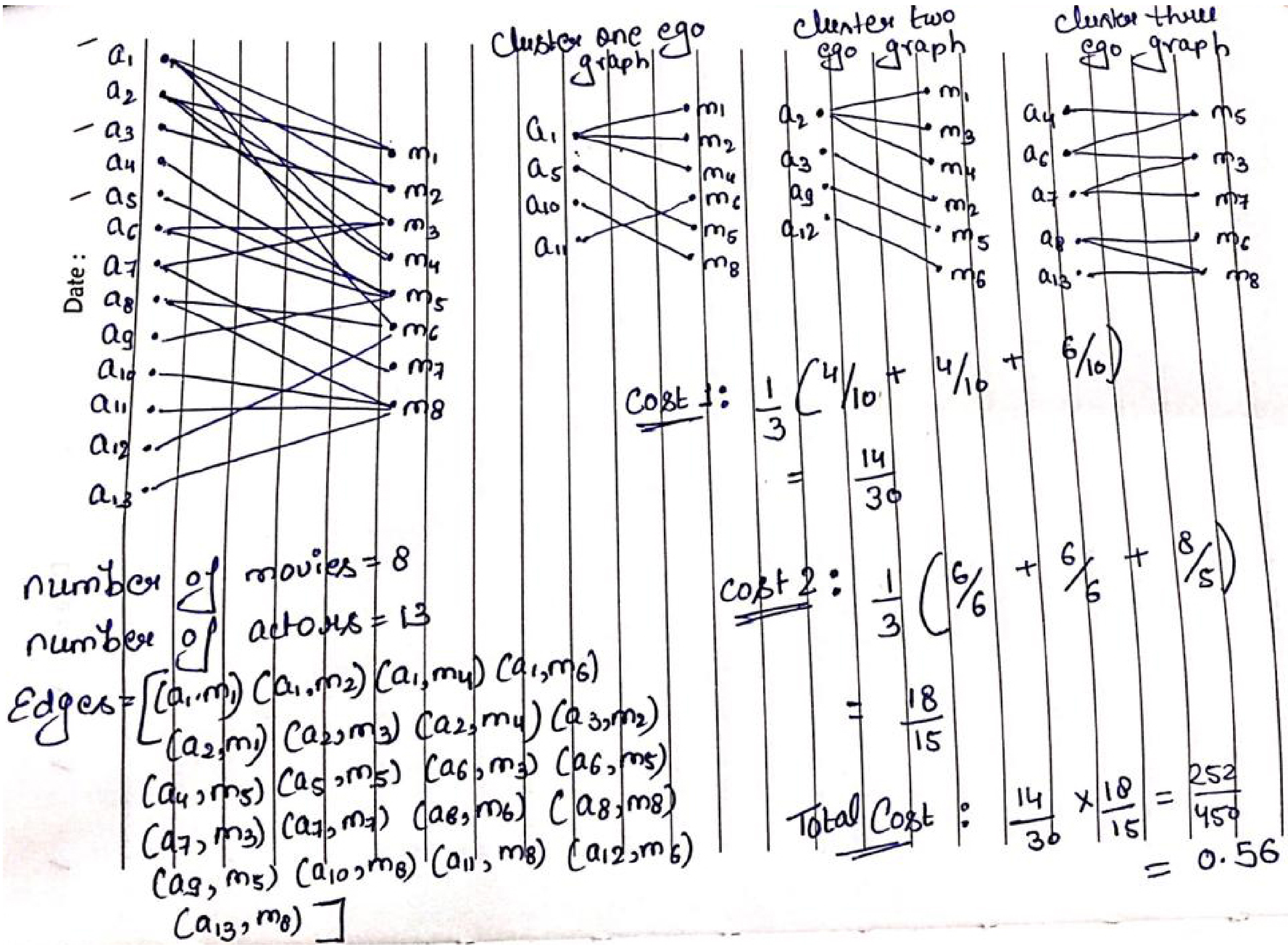
Please check clustering assignment helper functions notebook before attempting this assignment.

- Read graph from the given movie_actor_network.csv (note that the graph is bipartite graph.)
- Using stellergraph 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 =
   $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(number of nodes in the largest connected component in the graph with the actor nodes a}}{\text{(total number of nodes in that cluster i)}}$
   where N= number of clusters
   (Write your code in def cost1())
5. Cost2 =
   $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(sum of degress of actor nodes in the graph with the actor nodes and its movie neighbou}}{\text{(number of unique movie nodes in the graph with the actor nodes and its movie neighbou}}$
   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 =

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

where N= number of clusters

(Write your code in def cost1())

4. Cost2 =

$$\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(sum of degress of movie nodes in the graph with the movie nodes and its actor neighbou}}{\text{(number of unique actor nodes in the graph with the movie nodes and its actor neighbou}}$$

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 i
    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 ar
        cost2=cost2(graph1)+cost2(graph2)+cost2(graph3)
    computer the metric Cost = Cost1*Cost2
return number_of_clusters which have maximum Cost
```

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

```
Collecting networkx==2.3
  Downloading networkx-2.3.zip (1.7 MB)
     |████████████████████████████████| 1.7 MB 2.7 MB/s
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.7/dist-p
Building wheels for collected packages: networkx
  Building wheel for networkx (setup.py) ... done
  Created wheel for networkx: filename=networkx-2.3-py2.py3-none-any.whl size=1556
  Stored in directory: /root/.cache/pip/wheels/44/e6/b8/4efaab31158e9e9ca9ed80b11f
Successfully built networkx
Installing collected packages: networkx
  Attempting uninstall: networkx
    Found existing installation: networkx 2.6.3
    Uninstalling networkx-2.6.3:
      Successfully uninstalled networkx-2.6.3
ERROR: pip's dependency resolver does not currently take into account all the pack
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have imgaug 0.2.9 whi
Successfully installed networkx-2.3
Collecting stellargraph
  Downloading stellargraph-1.2.1-py3-none-any.whl (435 kB)
     |████████████████████████████████| 435 kB 2.8 MB/s
Requirement already satisfied: gensim>=3.4.0 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.7/dist
```

```
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: matplotlib>=2.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: numpy>=1.14 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: tensorflow>=2.1.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/lo
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/di
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dis
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: libclang>=9.0.1 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: flatbuffers<3.0,>=1.12 in /usr/local/lib/python3.7/
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.21.0 in /usr/local/
Requirement already satisfied: wheel<1.0,>=0.32.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: tensorflow-estimator<2.8,~=2.7.0rc0 in /usr/local/l
Requirement already satisfied: gast<0.5.0,>=0.2.1 in /usr/local/lib/python3.7/dist
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: absl-py>=0.4.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: keras<2.8,>=2.7.0rc0 in /usr/local/lib/python3.7/di
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: tensorboard~=2.6 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packag
```

```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
from stellargraph.data import UniformRandomMetaPathWalk
from stellargraph import StellarGraph
```

```python
from google.colab import files
files= files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving movie_actor_network.csv to movie_actor_network.csv
```

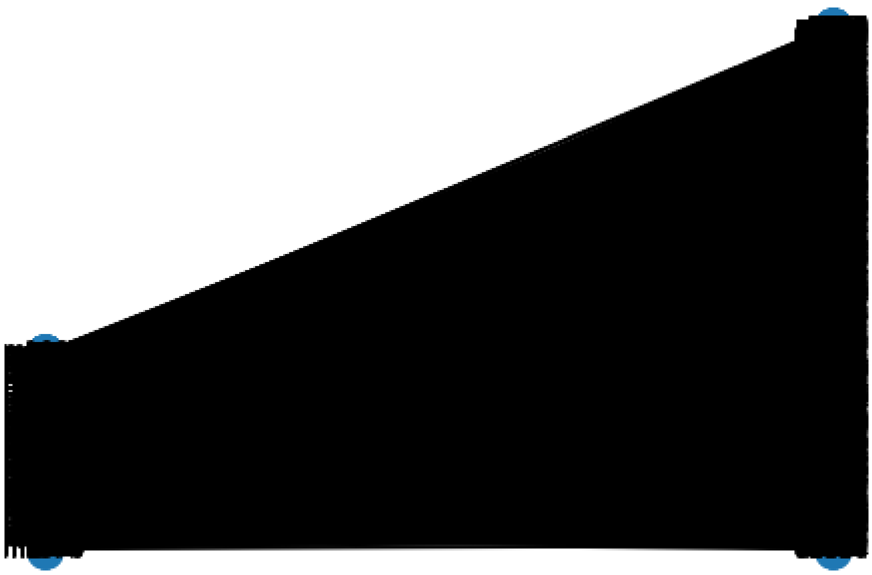
```python
data=pd.read_csv('movie_actor_network.csv', index_col=False, names=['movie','actor'])
```

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

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


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')


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


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

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

```
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()
```



```
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
```

```
# 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
```

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

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

```
(4703, 128)
```

```
# 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 embeddin
node_targets = [ A.node[node_id]['label'] for node_id in node_ids]
```

```
print(node_ids)
```

```
['a973', 'a967', 'a964', 'a1731', 'a970', 'a969', 'a1028', 'a1003', 'a1057', 'a965',
```

```
print(node_embeddings)
```

```
[[-0.36408344  0.6013007   0.19060278 ...  1.070536    0.36474088
  -1.5114822 ]
 [-1.0541456   1.5448879   0.42175627 ... -0.7364733   0.40513664
  -1.0941483 ]
 [-0.9837526   0.75910765 -0.25438234 ...  1.4631729  -0.32913098
  -1.453644  ]
 ...
 [-0.00555333  0.10831326  0.08687561 ... -0.00736847 -0.07330825
   0.03718201]
 [-0.03076893  0.16981243  0.13722724 ...  0.06990413 -0.08223744
  -0.02922636]
 [-0.00268559  0.08356352  0.09306414 ...  0.04964262 -0.08914711
   0.03794672]]
```

```
print(node_targets)
```

```
['actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'ac
```

```
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']
```

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

#In this function, we will split the node embeddings into actor_embeddings , movie_embeddi
#split the node_embeddings into actor_embeddings,movie_embeddings based on node_ids


  actor_embedding = [actor_embeddings.append(x) for i,x in enumerate(node_embeddings) if n
  actor_node = [actor_nodes.append(x) for i,x in enumerate(node_ids) if node_targets[i]=='

  movie_embedding = [movie_embeddings.append(x) for i,x in enumerate(node_embeddings) if n
  movie_node = [movie_nodes.append(x) for i,x in enumerate(node_ids) if node_targets[i]=='

  return actor_nodes, movie_nodes, actor_embeddings, movie_embeddings

  # By using node_embedding and node_targets, we can extract actor_embedding and movie emb
  # By using node_ids and node_targets, we can extract actor_nodes and movie nodes

  # 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 emb
  # By using node_ids and node_targets, we can extract actor_nodes and movie nodes


#L=actor_nodes
#M=movie_nodes
#N=actor_embeddings
#O=movie_embeddings
L,M,N,O = data_split(node_ids,node_targets,node_embeddings)
```

Grader function - 1

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

```
True
```

Grader function - 2

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

```
True
```

Calculating cost1

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

```
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()
    cost1= (num/Total_Nodes)*(1/number_of_clusters)
    return cost1
```

```
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
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'),
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,
```

Grader function - 3

```
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)
```

```
True
```

Calculating cost2

Cost2 =

$$\frac{1}{N}\sum_{\text{each cluster i}} \frac{(\text{sum of degress of actor nodes in the graph with the actor nodes and its movie neighbours in cl}}{(\text{number of unique movie nodes in the graph with the actor nodes and its movie neighbours in c}}$$

where N= number of clusters

```
def cost2(graph,number_of_clusters):
  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

```
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)
```

```
True
```

Grouping similar actors

```
print(node_ids)
```

```
['a973', 'a967', 'a964', 'a1731', 'a970', 'a969', 'a1028', 'a1003', 'a1057', 'a965',
```

```
from sklearn.cluster import KMeans

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

for cluster in cluster_list:
  algo=KMeans(n_clusters=cluster)
  algo.fit(N)
  label=algo.labels_
  dic=dict(zip(L,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,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)
```

```
0.4634181601629053
0.16662392683837254
0.019950886648122394
0.00031256422193422836
3.714072693383038e-05
6.911037011546084e-06
1.2051505892623307e-06
1.7916666666666666e-07
```

```
cost_1=+cost1(G1,cluster)
cost_2=+cost2(G1,cluster)


print(cost_1*cost_2)
Cost.append(cost_1*cost_2)
```

```
1.7916666666666666e-07
```

Displaying similar actor clusters

```
best_cluster=cluster_list[Cost.index(max(Cost))]
```

```
best_cluster
```

```
3
```

```
from sklearn.cluster import KMeans
k_means=KMeans(n_clusters=best_cluster)
k_means.fit(N)
print(k_means.cluster_centers_)
```

```
[[ 9.54607460e-02  5.99532485e-01  3.79215107e-01 -3.73679529e-01
  -8.46609806e-02 -1.69769979e-01 -2.29452921e-02  2.32799534e-01
  -2.51182783e-01  6.27938510e-04 -1.21437545e-01  4.03055856e-01
  -2.18338233e-01  1.31905728e-02  2.00726148e-01  2.07471038e-02
   9.80367482e-02 -2.35662888e-02 -5.26130794e-01  8.05199324e-02
  -2.89087371e-01 -3.04105332e-01  7.30368155e-02  1.51088809e-01
  -2.39291492e-01  9.07596538e-02  4.16188146e-01 -1.81508196e-02
  -4.17059869e-03 -2.64620921e-02  1.25675689e-01 -1.66350179e-01
   2.59854644e-01  2.89322731e-03 -4.74327062e-01  1.76162189e-02
  -4.58555997e-01 -1.33361751e-01 -1.07807377e-01  9.45527920e-02
  -4.77117593e-01 -2.55761125e-01  1.40315816e-01  6.47080711e-01
   1.97436974e-01 -4.38139587e-01  5.22430153e-02 -2.69946514e-01
   6.58499389e-02  2.80031521e-01  4.95228825e-02 -1.67023130e-01
  -3.48707768e-02  3.58042567e-01 -1.57436243e-01 -1.19424823e-01
  -1.79844404e-01 -5.30140052e-01 -6.05677192e-01  4.06630844e-01
  -3.45966840e-01 -1.67793805e-01  2.01237118e-01 -5.99831368e-02
   4.39903450e-01 -6.06832108e-02  6.87800305e-03  9.36709298e-02
   2.94726386e-01  1.00217688e-01  3.12673019e-01  2.65999708e-02
   7.91443934e-02 -4.76955867e-01  1.65353533e-01  2.35375012e-01
   3.32330581e-01  4.48218167e-01 -7.05333926e-02 -2.00634310e-01
   2.94484738e-01  6.35555409e-02  4.55050350e-03 -1.00483002e-01
  -4.19793731e-01 -7.04811531e-02  1.69441704e-01  1.45572140e-01
   5.52562323e-01  1.70096902e-01  1.97841252e-02  5.92959913e-02
   1.27183242e-01 -2.29883641e-01  4.63110228e-01  1.24007381e-01
   1.10746320e-01 -2.70541103e-01 -2.27366139e-01  6.46262111e-02
   3.09664222e-02  3.40435240e-01  3.72050814e-02  3.85155263e-02
   1.78080546e-01 -1.03371588e-01  1.78322454e-01 -1.94053375e-01
  -2.63810347e-01 -4.24888186e-02  8.39282917e-02  4.32903381e-01
  -1.14975044e-01  4.44468337e-02 -3.54368857e-01  1.84932964e-01
  -4.56275630e-01  3.28409568e-01  5.89688360e-02  2.11379130e-01
   2.32970879e-01  3.19835342e-01 -2.56398154e-01  1.21393596e-01
   5.98505870e-02 -1.35271464e-01 -3.59245594e-01  1.08579749e-01]
 [-2.54349681e-02  2.12812602e-01  1.34941253e-01 -1.73470729e-01
  -1.16199025e-01 -1.26948730e-02 -1.08043927e-02  5.92070253e-02
  -1.15467562e-01 -8.00720359e-02 -2.85396533e-02  2.35770578e-01
   4.67164679e-04  9.72136450e-02  1.26728897e-01 -2.16806369e-02
  -1.25390573e-01 -3.63990547e-02 -2.53915558e-01  1.04701673e-01
  -1.31128900e-01 -2.83695531e-02 -4.49540636e-02  9.39394226e-02
  -5.77488003e-02  4.11110505e-02  1.29851544e-01 -1.63154994e-01
  -6.43635004e-02 -4.36374752e-02  6.42113277e-02 -6.86156946e-03
   1.99648731e-02 -8.64580639e-02 -1.85852427e-01  5.53942700e-02
```

```
-1.89649203e-01 -1.89368487e-01 -4.12125463e-02 -1.09342361e-01
-1.57809449e-01 -1.28001154e-01  4.71356070e-02  2.85659557e-01
-2.11438632e-02 -1.52971984e-01 -6.05785009e-02 -2.63945839e-02
-7.89081221e-02  4.20205887e-02 -7.46095936e-03 -1.30745491e-01
 9.46664424e-02  2.29885433e-01 -1.50134687e-01 -1.04264122e-01
-2.03363611e-01 -3.34972736e-01 -2.09151332e-01  2.05653254e-01
-1.67168319e-01 -7.87531168e-02  7.41168779e-02  3.49661330e-02
 1.55055839e-01  3.55264268e-02 -3.23949571e-02  4.58281644e-02
 1.35328098e-01 -1.14270311e-01  1.56760972e-01  1.63541414e-01
-4.93908453e-02 -1.57643255e-01  9.94587389e-02  7.12324954e-02
 1.30532950e-01  2.35057447e-01 -1.02198133e-01 -1.07740772e-01
 1.86643498e-01  3.77191677e-02  9.59592163e-02 -7.57240116e-02
-1.19631018e-01 -1.62591817e-01  2.04314457e-01 -6.27329430e-03
 1.44724656e-01  1.24520987e-01 -1.68235727e-02 -4.82857708e-02
 1.30417273e-01 -1.11421189e-02  1.81791116e-01  9.53360371e-02
 6.92596827e-02 -4.84978866e-02 -1.79146838e-01 -3.82482790e-02
 1.29939380e-01  2.50652045e-01 -5.15113595e-02  3.78116159e-02
```

```
print(k_means.labels_)
```

```
[2 2 2 ... 1 1 1]
```

```
from sklearn.manifold import TSNE
```

```
#dimension data for actor node:
dimension_data_for_actor_node = N

dimension_data_for_actor_node_array=np.asarray([dimension_data_for_actor_node])
dimension_data_for_actor_node_array.shape
```

```
(1, 3411, 128)
```

```
dimension_data_for_actor_node_final=np.reshape(dimension_data_for_actor_node_array,(3411,1
dimension_data_for_actor_node_final.shape
```

```
(3411, 128)
```

```
#step2:apply kmeans algorithm on data using n_cluster
from sklearn.cluster import KMeans
#here we are considering n_clusters=3
kmeans= KMeans(n_clusters=3)
kmeans.fit(dimension_data_for_actor_node_final)
#now Kmeans model contain 3 clusters and each cluster contain similar actor nodes
predicted_cluster=kmeans.predict(dimension_data_for_actor_node_final)

#Step3:

from sklearn.manifold import TSNE
#TSNE_model = TSNE
```

```
TSNE_model = TSNE(n_components=2, random_state=0)

#apply TSNE model on the "dimension data for actor node" to reduce 128 dimensions to 2 dim
two_dimensional_data = TSNE_model.fit_transform(dimension_data_for_actor_node_final)

#now 2 dimensional data contains 3411 rows and 2 dimensions
two_dimensional_data_shape= two_dimensional_data.shape

#step4: Perform Verticle Stacking
#By using vstack() function which is present inside the numpy module, we are going to perf
#Taking Transpose:
transpose_predicted_cluster = predicted_cluster.T
transpose_two_dimensional_data = two_dimensional_data.T

required_data = np.vstack((transpose_predicted_cluster, transpose_two_dimensional_data))

#Now shape of required data is (3, 3411)
#step5:
#use DataFrame() function present in pandas module to convert the transposed_required_data
import pandas as pd
import seaborn as sn

final_data = pd.DataFrame(data= required_data.T, columns= ["Col_1","Col_2","Col_3"])
#now final_data is a DataFrame, which contain 3411 rows and 3 columns

#Ploting the result of tsne

sn.FacetGrid(final_data, hue="Col_1", size=6).map(plt.scatter, 'Col_3', 'Col_2')
plt.title('Visualization for similar actor clusters With perplexity = 2')
plt.show()
```

Grouping similar movies

```
from sklearn.cluster import KMeans

cluster_list5=[3,5,10,30,50,100,200,500]
Cost5=[]

for cluster in cluster_list5:
  algo5=KMeans(n_clusters=cluster)
  algo5.fit(O)
  label=algo5.labels_
  dic=dict(zip(M,label))
  cost_15=0
  cost_25=0
  for i in label:
    ac_node5 = [k for k,v in dic.items() if v == i]
    G15=nx.Graph()
    for n in ac_node5:

      sub_graph15 = nx.ego_graph(A,n)
      G15.add_nodes_from(sub_graph15.nodes)
      G15.add_edges_from(sub_graph15.edges())

    cost_15=+cost1(G15,cluster)
    cost_25=+cost2(G15,cluster)

  print(cost_15*cost_25)
  Cost5.append(cost_15*cost_25)
```

```
0.7064356136212424
0.21454450955332527
0.023392802865827185
0.001840238704177323
0.0006264024826927669
8.159830177755859e-05
1.0045422781271837e-05
4.0710059171597637e-07
```

Displaying similar movie clusters

```
best_cluster1=cluster_list5[Cost5.index(max(Cost5))]
```

```
best_cluster1
```

```
3
```

```
cost_15=+cost1(G15,cluster)
```

```
cost_25=+cost2(G15,cluster)

print(cost_15*cost_25)
Cost5.append(cost_15*cost_25)
```

```
4.0710059171597637e-07
```

```
from sklearn.cluster import KMeans
k_means5=KMeans(n_clusters=best_cluster1)
k_means5.fit(O)
print(k_means.cluster_centers_)
```

```
 -5.77488003e-02  4.11110505e-02  1.29851544e-01 -1.63154994e-01
 -6.43635004e-02 -4.36374752e-02  6.42113277e-02 -6.86156946e-03
  1.99648731e-02 -8.64580639e-02 -1.85852427e-01  5.53942700e-02
 -1.89649203e-01 -1.89368487e-01 -4.12125463e-02 -1.09342361e-01
 -1.57809449e-01 -1.28001154e-01  4.71356070e-02  2.85659557e-01
 -2.11438632e-02 -1.52971984e-01 -6.05785009e-02 -2.63945839e-02
 -7.89081221e-02  4.20205887e-02 -7.46095936e-03 -1.30745491e-01
  9.46664424e-02  2.29885433e-01 -1.50134687e-01 -1.04264122e-01

 -2.03363611e-01 -3.34972736e-01 -2.09151332e-01  2.05653254e-01
 -1.67168319e-01 -7.87531168e-02  7.41168779e-02  3.49661330e-02
  1.55055839e-01  3.55264268e-02 -3.23949571e-02  4.58281644e-02
  1.35328098e-01 -1.14270311e-01  1.56760972e-01  1.63541414e-01
 -4.93908453e-02 -1.57643255e-01  9.94587389e-02  7.12324954e-02
  1.30532950e-01  2.35057447e-01 -1.02198133e-01 -1.07740772e-01
  1.86643498e-01  3.77191677e-02  9.59592163e-02 -7.57240116e-02
 -1.19631018e-01 -1.62591817e-01  2.04314457e-01 -6.27329430e-03
  1.44724656e-01  1.24520987e-01 -1.68235727e-02 -4.82857708e-02
  1.30417273e-01 -1.11421189e-02  1.81791116e-01  9.53360371e-02
  6.92596827e-02 -4.84978866e-02 -1.79146838e-01 -3.82482790e-02
  1.29939380e-01  2.50652045e-01 -5.15113595e-02  3.78116159e-02
  5.51504942e-03  6.59905866e-02 -4.12064623e-02 -1.09674110e-01
 -1.75619215e-01  1.78051749e-01  6.25471678e-02  1.01410793e-01
 -1.29898016e-01  4.04469804e-02 -1.67707794e-01  1.46538096e-01
 -2.37994509e-01  1.50411334e-01  3.72793086e-02  2.53133146e-01
  1.60495319e-02  1.44359859e-01 -1.23009899e-01  1.82586891e-01
  1.88174382e-01  9.44932487e-02 -1.48378647e-01  1.49408322e-02]
[-3.73964297e-01  3.29730182e-01  4.32758440e-01  1.61591820e+00
 -9.52933828e-01  1.24605928e-01  5.92688663e-01 -1.02157270e+00
  1.84998843e-01  7.06643909e-01  8.77156510e-02  3.21520147e-01
 -6.15334861e-01  5.53349564e-01  3.42831766e-01  5.99067235e-01
  7.00454169e-01 -7.77054579e-01  4.55620214e-01 -6.73362716e-01
  8.07452488e-01 -1.29957713e-01 -9.23774530e-02 -6.36223676e-04
  4.09922561e-01 -8.00451584e-01  7.96903223e-01 -1.10460797e+00
  6.58207253e-02 -1.74918596e-02  1.74495676e-01  7.82324880e-02
 -9.10586734e-01 -1.38556187e+00  3.19999811e-01 -3.49063062e-01
 -4.37788227e-01 -1.19666749e+00 -1.17384881e+00 -2.26194760e-01
  7.56546555e-01 -1.84409472e+00 -3.48742266e-01 -1.25626321e-01
 -6.80978898e-01 -2.54663944e-01 -6.46603936e-01 -1.24542154e+00
 -1.92096982e-01 -7.09020816e-01 -2.49511236e-01 -1.20710050e+00
  7.27903621e-01  9.75389400e-01 -5.45433947e-01  1.26564168e+00
 -6.60060577e-01 -1.05096959e+00  5.48695692e-01 -5.70247636e-01
 -1.39865031e+00  3.42316990e-01 -4.13082001e-01  5.99081641e-02
  1.83884035e-01 -3.08431205e-02  6.60632413e-01 -5.55440246e-01
  1.25723628e-03 -5.29158077e-01  6.31734932e-01  1.24151801e+00
 -1.87508945e+00  8.11915236e-01  9.09469237e-02 -7.97689991e-01
 -7.78323143e-01  1.73357898e-01 -2.41624335e-01  8.04416065e-01
```

```
 -5.66698716e-01  9.82825698e-02 -3.14527004e-02 -1.79756863e+00
 -1.22718392e+00 -1.55002018e+00  1.63572381e-01 -7.18288128e-01
 -2.26352738e-01  4.86608947e-01 -1.11741529e+00 -5.11314666e-01
  8.42685447e-01  8.53931082e-01 -1.25920163e+00  6.85780845e-01
  5.51616082e-01 -2.47969338e-01  3.94633008e-01 -3.97431763e-01
  1.13887967e+00  1.31987398e+00  5.77958881e-01 -7.70017459e-01
 -3.78646943e-01  8.06627109e-02 -7.85700081e-01  8.59214668e-01
  6.21636513e-01  4.62352629e-01  2.91618379e-01 -7.65181963e-01
 -1.66174502e-01 -1.47473835e+00 -5.77785479e-02 -9.64427073e-01
 -1.33522899e-01  6.62439771e-01  6.86899287e-01  4.95839658e-01
  1.79901471e-01 -9.00134378e-01 -2.01147709e-01 -2.31797147e-02
  3.32848323e-01  4.39565334e-01 -1.86615514e-01 -7.26353323e-01]]
```

```
print(k_means5.labels_)
```

```
[0 0 0 ... 1 1 1]
```

```
from sklearn.manifold import TSNE
```

```
#dimension data for movie node:
dimension_data_for_movie_node = O

dimension_data_for_movie_node_array=np.asarray([dimension_data_for_movie_node])
dimension_data_for_movie_node_array.shape
```

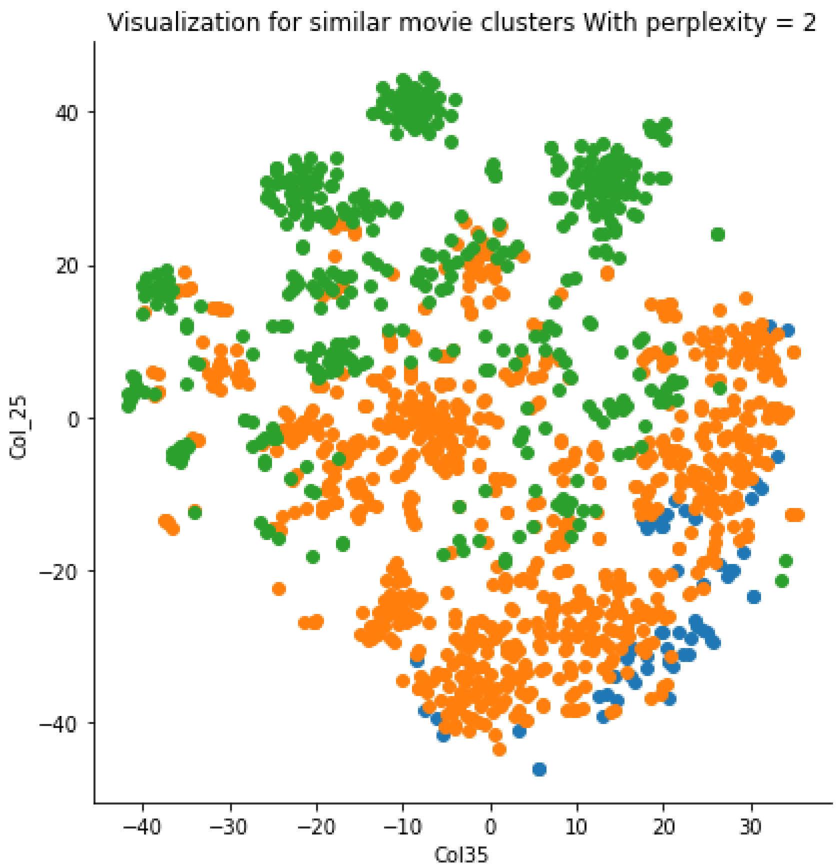
```
(1, 1292, 128)
```

```
dimension_data_for_movie_node_final=np.reshape(dimension_data_for_movie_node_array,(1292,1
dimension_data_for_movie_node_final.shape
```

```
(1292, 128)
```

```
#step2:apply kmeans algorithm on data using n_cluster
from sklearn.cluster import KMeans
#here we are considering n_clusters=5
kmeans5= KMeans(n_clusters=5)

kmeans5.fit(dimension_data_for_movie_node_final)
#now Kmeans model contain five clusters and each cluster contain similar movie nodes
predicted_cluster5=kmeans.predict(dimension_data_for_movie_node_final)

#Step3:

from sklearn.manifold import TSNE
#TSNE_model = TSNE

TSNE_model5 = TSNE(n_components=2)

#apply TSNE model on the "dimension data for actor node" to reduce 128 dimensions to 2 dim
two_dimensional_data5 = TSNE_model5.fit_transform(dimension_data_for_movie_node_final)

#now 2 dimensional data contains 3411 rows and 2 dimensions
two_dimensional_data5_shape= two_dimensional_data5.shape
```

```
#step4: Perform Verticle Stacking
#By using vstack() function which is present inside the numpy module, we are going to perf
#Taking Transpose:
transpose_predicted_cluster5 = predicted_cluster5.T
transpose_two_dimensional_data5 = two_dimensional_data5.T

required_data5 = np.vstack((transpose_predicted_cluster5,transpose_two_dimensional_data5))

#Now shape of required data is (3, 1292)
#step5:
#use DataFrame() function present in pandas module to convert the transposed_required_data
import pandas as pd
import seaborn as sn

final_data5 = pd.DataFrame(required_data5.T, columns= ["Col_15","Col_25","Col35"])
#now final_data is a DataFrame, which contain 1292 rows and 3 columns

#Ploting the result of tsne

sn.FacetGrid(final_data5, hue="Col_15", size=6).map(plt.scatter, 'Col35', 'Col_25')
plt.title('Visualization for similar movie clusters With perplexity = 2')
plt.show()
```



✓ 33s completed at 2:24 AM