Clustering_on_Graph_Dataset_Assignment_updated

April 14, 2021

1 Clustering Assignment

2 Task 1 : Apply clustering algorithm to group similar actors

[1]:
```
!pip install networkx==2.3
```

```
Collecting networkx==2.3
  Downloading https://files.pythonhosted.org/packages/85/08/f20aef11d4c343
b557e5de6b9548761811eb16e438cee3d32b1c66c8566b/networkx-2.3.zip (1.7MB)
     |                 | 1.8MB 17.0MB/s
Requirement already satisfied: decorator>=4.3.0 in
/usr/local/lib/python3.7/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-any.whl
size=1556408
sha256=bc465c77feb4a9b3649c55ff339aa5d99868c6b10ba215c57778f56e88cb06f8
  Stored in directory: /root/.cache/pip/wheels/de/63/64/3699be2a9d0ccdb37c7f1632
9acf3863fd76eda58c39c737af
Successfully built networkx
ERROR: albumentations 0.1.12 has requirement imgaug<0.2.7,>=0.2.5, but

you'll have imgaug 0.2.9 which is incompatible.
Installing collected packages: networkx
  Found existing installation: networkx 2.5.1
    Uninstalling networkx-2.5.1:
      Successfully uninstalled networkx-2.5.1
Successfully installed networkx-2.3
```

[2]:
```
!pip install stellargraph
```

```
Collecting stellargraph
  Downloading https://files.pythonhosted.org/packages/74/78/16b23ef04cf6fb
24a7dea9fd0e03c8308a56681cc5efe29f16186210ba04/stellargraph-1.2.1-py3-none-
any.whl (435kB)
     |                 | 440kB 17.3MB/s
Requirement already satisfied: numpy>=1.14 in
/usr/local/lib/python3.7/dist-packages (from stellargraph) (1.19.5)
```

```
Requirement already satisfied: scikit-learn>=0.20 in
/usr/local/lib/python3.7/dist-packages (from stellargraph) (0.22.2.post1)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.7/dist-
packages (from stellargraph) (1.1.5)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.7/dist-
packages (from stellargraph) (2.3)
Requirement already satisfied: tensorflow>=2.1.0 in
/usr/local/lib/python3.7/dist-packages (from stellargraph) (2.4.1)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-
packages (from stellargraph) (1.4.1)
Requirement already satisfied: gensim>=3.4.0 in /usr/local/lib/python3.7/dist-
packages (from stellargraph) (3.6.0)
Requirement already satisfied: matplotlib>=2.2 in /usr/local/lib/python3.7/dist-
packages (from stellargraph) (3.2.2)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-
packages (from scikit-learn>=0.20->stellargraph) (1.0.1)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas>=0.24->stellargraph) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-
packages (from pandas>=0.24->stellargraph) (2018.9)
Requirement already satisfied: decorator>=4.3.0 in
/usr/local/lib/python3.7/dist-packages (from networkx>=2.2->stellargraph)
(4.4.2)
Requirement already satisfied: termcolor~=1.1.0 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph)
(1.1.0)
Requirement already satisfied: tensorboard~=2.4 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph)
(2.4.1)
Requirement already satisfied: six~=1.15.0 in /usr/local/lib/python3.7/dist-
packages (from tensorflow>=2.1.0->stellargraph) (1.15.0)
Requirement already satisfied: wheel~=0.35 in /usr/local/lib/python3.7/dist-
packages (from tensorflow>=2.1.0->stellargraph) (0.36.2)
Requirement already satisfied: wrapt~=1.12.1 in /usr/local/lib/python3.7/dist-
packages (from tensorflow>=2.1.0->stellargraph) (1.12.1)
Requirement already satisfied: tensorflow-estimator<2.5.0,>=2.4.0 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph)
(2.4.0)
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.7/dist-
packages (from tensorflow>=2.1.0->stellargraph) (3.12.4)
Requirement already satisfied: h5py~=2.10.0 in /usr/local/lib/python3.7/dist-
packages (from tensorflow>=2.1.0->stellargraph) (2.10.0)
Requirement already satisfied: flatbuffers~=1.12.0 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph)
(1.12)
Requirement already satisfied: astunparse~=1.6.3 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph)
(1.6.3)
```

```
Requirement already satisfied: opt-einsum~=3.3.0 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph)
(3.3.0)
Requirement already satisfied: grpcio~=1.32.0 in /usr/local/lib/python3.7/dist-
packages (from tensorflow>=2.1.0->stellargraph) (1.32.0)
Requirement already satisfied: keras-preprocessing~=1.1.2 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph)
(1.1.2)
Requirement already satisfied: gast==0.3.3 in /usr/local/lib/python3.7/dist-
packages (from tensorflow>=2.1.0->stellargraph) (0.3.3)
Requirement already satisfied: google-pasta~=0.2 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph)
(0.2.0)
Requirement already satisfied: absl-py~=0.10 in /usr/local/lib/python3.7/dist-
packages (from tensorflow>=2.1.0->stellargraph) (0.12.0)
Requirement already satisfied: typing-extensions~=3.7.4 in
/usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph)
(3.7.4.3)
Requirement already satisfied: smart-open>=1.2.1 in
/usr/local/lib/python3.7/dist-packages (from gensim>=3.4.0->stellargraph)
(5.0.0)
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: kiwisolver>=1.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib>=2.2->stellargraph)
(1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-
packages (from matplotlib>=2.2->stellargraph) (0.10.0)
Requirement already satisfied: werkzeug>=0.11.15 in
/usr/local/lib/python3.7/dist-packages (from
tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (1.0.1)
Requirement already satisfied: setuptools>=41.0.0 in
/usr/local/lib/python3.7/dist-packages (from
tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (54.2.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-
packages (from tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (3.3.4)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in
/usr/local/lib/python3.7/dist-packages (from
tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (1.8.0)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.7/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/python3.7/dist-packages (from
tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (1.28.1)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in
/usr/local/lib/python3.7/dist-packages (from
tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (0.4.4)
Requirement already satisfied: importlib-metadata; python_version < "3.8" in
/usr/local/lib/python3.7/dist-packages (from
markdown>=2.6.8->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (3.10.0)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from
requests<3,>=2.21.0->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from
requests<3,>=2.21.0->tensorboard~=2.4->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~=2.4->tensorflow>=2.1.0->stellargraph)
(2020.12.5)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from
requests<3,>=2.21.0->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (2.10)
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3.6" in
/usr/local/lib/python3.7/dist-packages (from google-
auth<2,>=1.6.3->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (4.7.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.7/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/python3.7/dist-packages (from google-
auth<2,>=1.6.3->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph) (4.2.1)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.7/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: zipp>=0.5 in /usr/local/lib/python3.7/dist-
packages (from importlib-metadata; python_version <
"3.8"->markdown>=2.6.8->tensorboard~=2.4->tensorflow>=2.1.0->stellargraph)
(3.4.1)
Requirement already satisfied: pyasn1>=0.1.3 in /usr/local/lib/python3.7/dist-
packages (from rsa<5,>=3.1.4; python_version >= "3.6"->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/python3.7/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)
Installing collected packages: stellargraph
Successfully installed stellargraph-1.2.1
```

[3]:

```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
```

[4]:

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

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

[5]:

```python
data=pd.read_csv('/content/drive/My Drive/movie_actor_network.csv',␣
 ↪index_col=False, names=['movie','actor'])
```

[6]:

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

[7]:

```python
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')
```

[8]:

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

[9]:

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

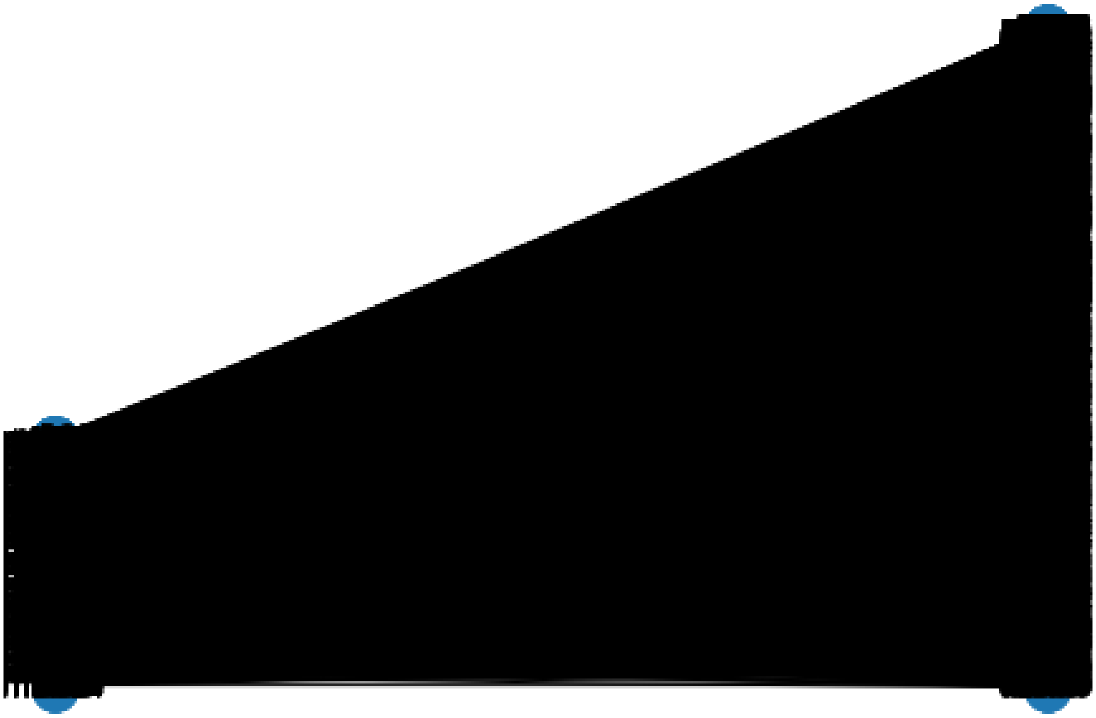
```
number of nodes 4703
number of edges 9650
```

[10]:

```python
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()
```



[11]:
```
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
```

[12]:
```
# 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
```

[13]:
```
from gensim.models import Word2Vec
model = Word2Vec(walks, size=128, window=5)
```

[14]:
```
model.wv.vectors.shape  # 128-dimensional vector for each node in the graph
```

[14]:
```
(4703, 128)
```

[15]:
```
# 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]
```

[16]:
```
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=[],[]
    for index,node in enumerate(node_ids):
        if node_targets[index]=='actor':
            actor_nodes.append(node)
        else:
            movie_nodes.append(node)

    for index,embeddings in enumerate(node_embeddings):
        if node_targets[index]=='actor':
            actor_embeddings.append(embeddings)
        else:
            movie_embeddings.append(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

    return actor_nodes,movie_nodes,actor_embeddings,movie_embeddings
```

```
actor_nodes,movie_nodes,actor_embeddings,movie_embeddings=data_split(node_ids,node_targets,nod
```

Graded function - 1

[17]:
```
def grader_actors(data):
    assert(len(data)==3411)
    return True
grader_actors(actor_nodes)
```

[17]:
```
True
```

Graded function - 2

[18]:
```
def grader_movies(data):
    assert(len(data)==1292)
    return True
grader_movies(movie_nodes)
```

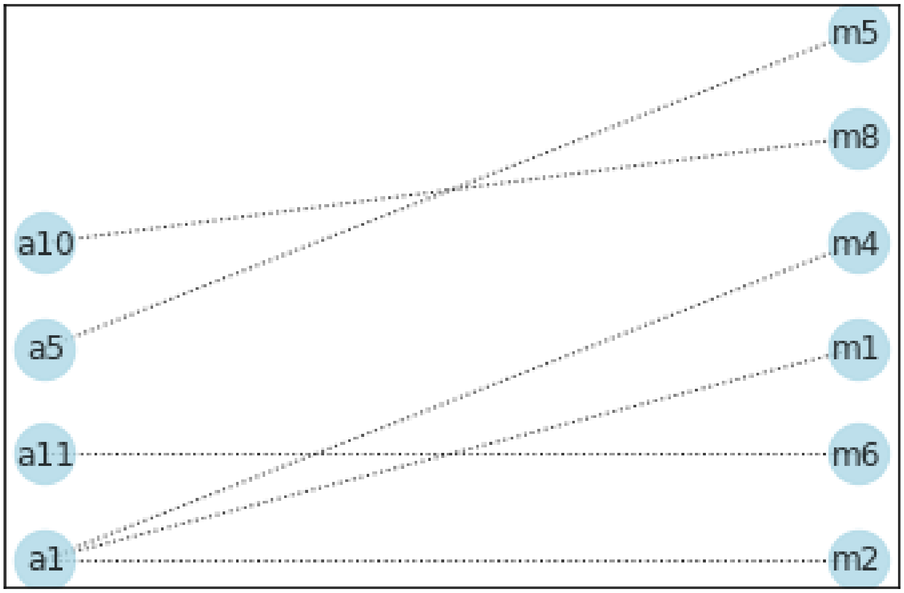
[18]:
```
True
```

Calculating cost1

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

[19]:
```
def cost1(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''
    cost1= (1/number_of_clusters)*((max(nx.
 ↪connected_component_subgraphs(graph),key=len)).number_of_nodes()/graph.
 ↪number_of_nodes())# calculate cost1
    return cost1
```

[20]:
```
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)
```



Graded function - 3

```
[21]: 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)
```

```
[21]: True
```

Calculating cost2

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

```
[22]: def cost2(graph,number_of_clusters):
          '''In this function, we will calculate cost2'''
          nodes=graph.nodes()
          actor=[]
          movie=[]
          for node in nodes:
            if 'a' in node:
              actor.append(node)
            else:
              movie.append(node)
          no_of_uniquemovienodes=len(list(set(movie)))
          sum_of_degree=sum(d for i,d in graph.degree(actor))
```

```
    cost2=(1/number_of_clusters)*(sum_of_degree/no_of_uniquemovienodes) #␣
↪calculate cost2

    return cost2
```

Graded function - 4

[23]:

```
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)
```

[23]:

```
True
```

Grouping similar actors

[24]:

```
max_cost=0
for number_of_clusters in [3, 5, 10, 30, 50, 100, 200, 500]:
  kmeans=KMeans(n_clusters=number_of_clusters)
  kmeans.fit(actor_embeddings)
  cluster_labels=kmeans.labels_
  dict_of_actor_nodes=dict(zip(actor_nodes,cluster_labels))
  list_of_clusters=[]
  for cluster_number in np.unique(cluster_labels):
    cluster=[]
    for node,label in dict_of_actor_nodes.items():
      if label==cluster_number:
        cluster.append(node)
    list_of_clusters.append(cluster)
  cost_1=0
  cost_2=0
  for cluster in list_of_clusters:
    G=nx.Graph()
    for actornode in cluster:
      sub_graph=nx.ego_graph(A,actornode)
      G.add_nodes_from(sub_graph.nodes())
      G.add_edges_from(sub_graph.edges())
    cost_1+=cost1(G,number_of_clusters)
    cost_2+=cost2(G,number_of_clusters)
  metric_cost=cost_1*cost_2
  if metric_cost>max_cost:
    max_cost=metric_cost
    optimal_no_of_clusters=number_of_clusters
```

[25]:

```
optkmeans=KMeans(n_clusters=optimal_no_of_clusters)
optkmeans.fit(actor_embeddings)
lables=optkmeans.labels_
```

Displaying similar actor clusters

[26]:

```
from sklearn.manifold import TSNE
transform = TSNE #PCA

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

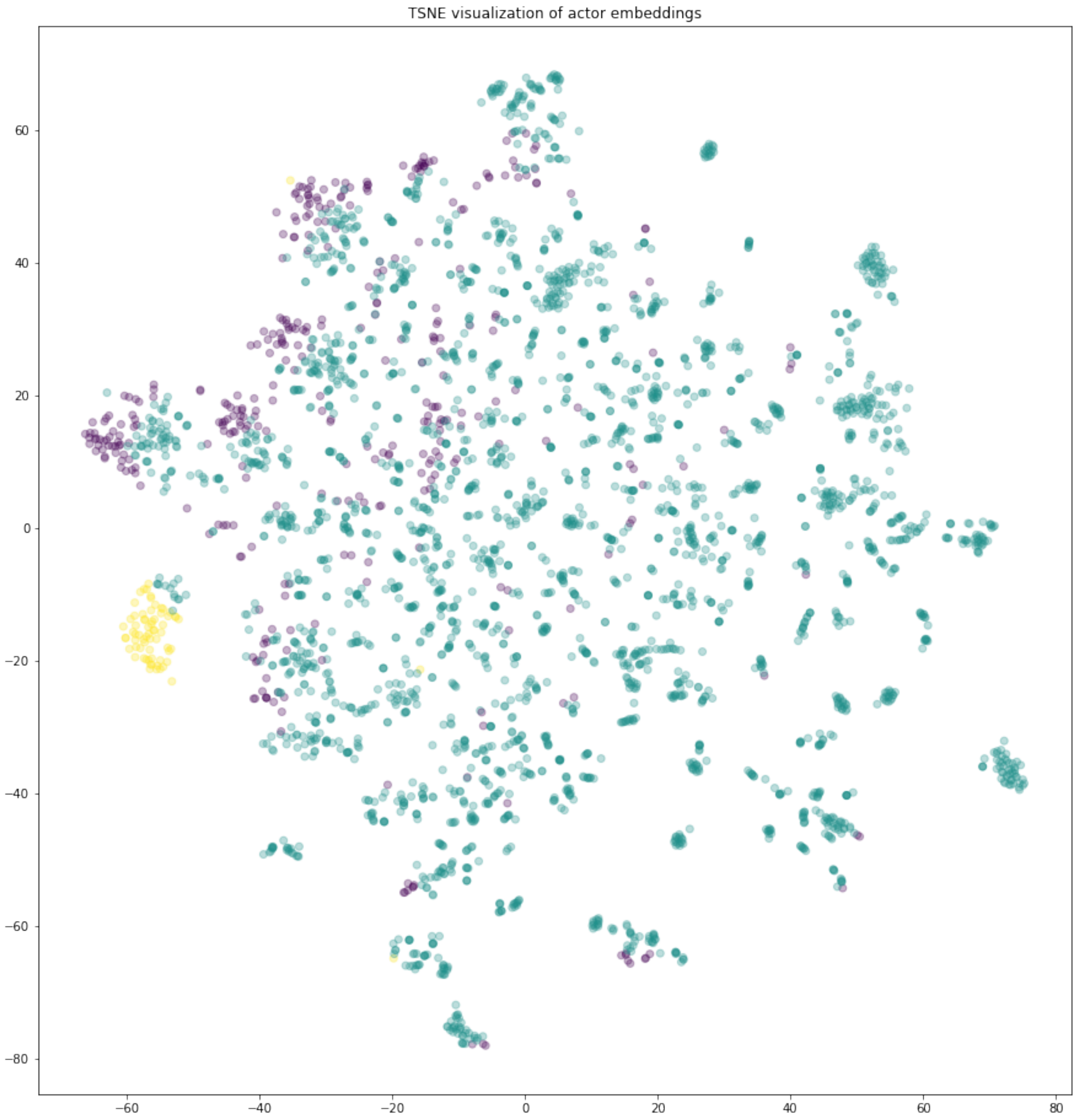
[27]:

```
import numpy as np
# draw the points

label_map = { l: i for i, l in enumerate(np.unique(lables))}
node_colours = [ label_map[target] for target in lables]

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()
```



3 Task 2 : Apply clustering algorithm to group similar movies

Grouping similar movies

[28]:
```
def cost2m(graph,number_of_clusters):
    '''In this function, we will calculate cost2'''
    nodes=graph.nodes()
    actor=[]
    movie=[]
    for node in nodes:
      if 'a' in node:
```

```
    actor.append(node)
   else:
    movie.append(node)
  no_of_uniqueactornodes=len(list(set(actor)))
  sum_of_degree=sum(d for i,d in graph.degree(movie))
  cost2=(1/number_of_clusters)*(sum_of_degree/no_of_uniqueactornodes) #␣
↪calculate cost2

  return cost2
```

[29]:

```
max_cost=0
for number_of_clusters in [3, 5, 10, 30, 50, 100, 200, 500]:
 kmeans=KMeans(n_clusters=number_of_clusters)
 kmeans.fit(movie_embeddings)
 cluster_labels=kmeans.labels_
 dict_of_movie_nodes=dict(zip(movie_nodes,cluster_labels))
 list_of_clusters=[]
 for cluster_number in np.unique(cluster_labels):
  cluster=[]
  for node,label in dict_of_movie_nodes.items():
   if label==cluster_number:
    cluster.append(node)
  list_of_clusters.append(cluster)
 cost_1=0
 cost_2=0
 for cluster in list_of_clusters:
  G=nx.Graph()
  for movienode in cluster:
   sub_graph=nx.ego_graph(A,movienode)
   G.add_nodes_from(sub_graph.nodes())
   G.add_edges_from(sub_graph.edges())
  cost_1+=cost1(G,number_of_clusters)
  cost_2+=cost2m(G,number_of_clusters)
 metric_cost=cost_1*cost_2
 if metric_cost>max_cost:
  max_cost=metric_cost
  optimal_no_of_clusters=number_of_clusters
```

[30]:

```
optkmeans=KMeans(n_clusters=optimal_no_of_clusters)
optkmeans.fit(movie_embeddings)
lables=optkmeans.labels_
```

Displaying similar movie clusters

[31]:

```
from sklearn.manifold import TSNE
transform = TSNE #PCA
```

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

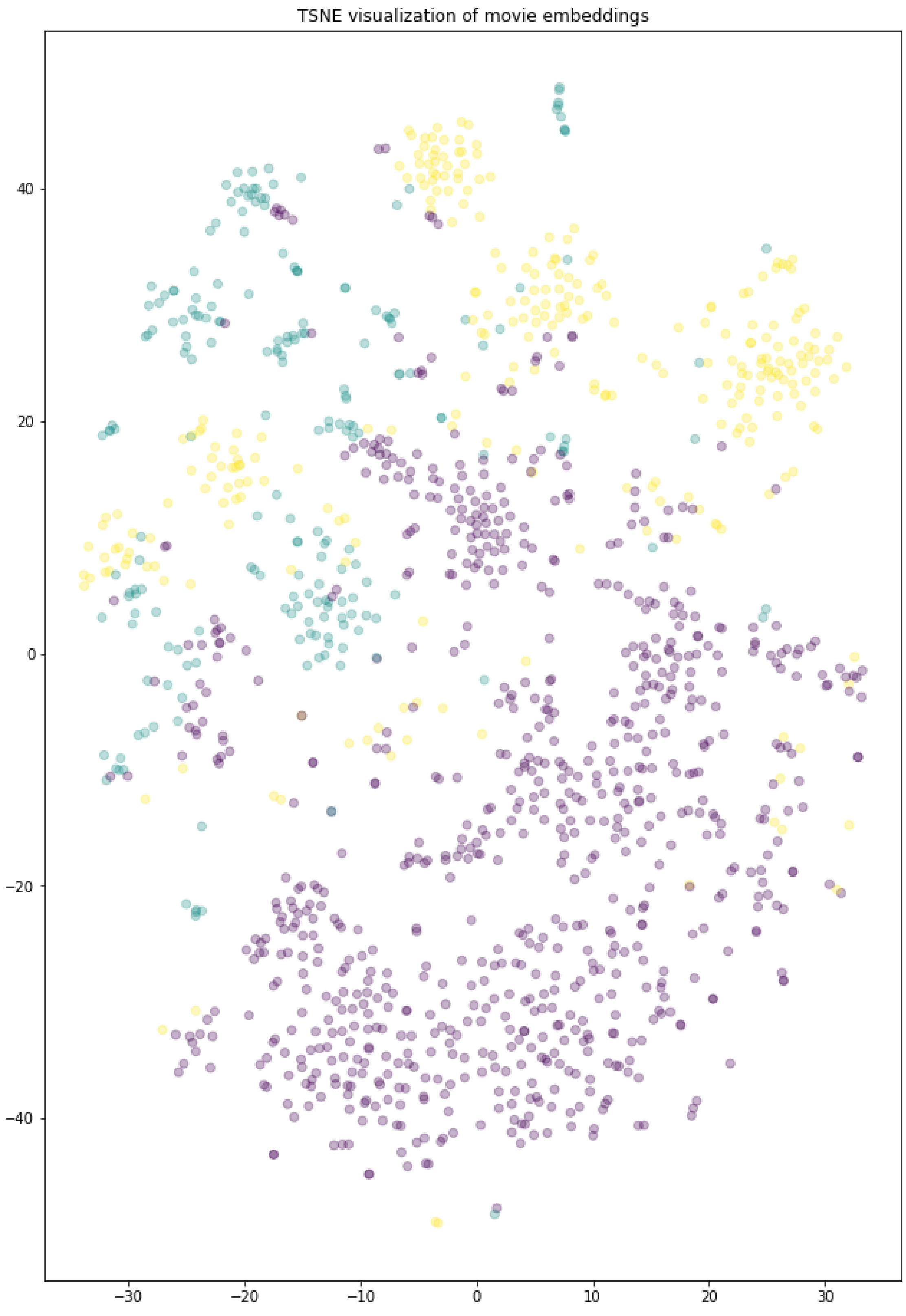
[32]:

```
import numpy as np
# draw the points

label_map = { l: i for i, l in enumerate(np.unique(lables))}
node_colours = [ label_map[target] for target in lables]

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
40
20
0
−20
−40
−30
−20
−10
0
10
20
30