Cluster_Assignment

December 19, 2020

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

[2]:
```
from sklearn.cluster import KMeans

import pandas as pd
import numpy as np
import networkx as nx

from networkx.algorithms import bipartite
import matplotlib.pyplot as plt

from stellargraph.data import UniformRandomMetaPathWalk
from stellargraph import StellarGraph

import warnings
warnings.filterwarnings("ignore")
```

[3]:
```
data = pd.read_csv('/content/drive/MyDrive/KMeans/movie_actor_network.csv',␣
↪index_col=False, names=['movie','actor'])
```

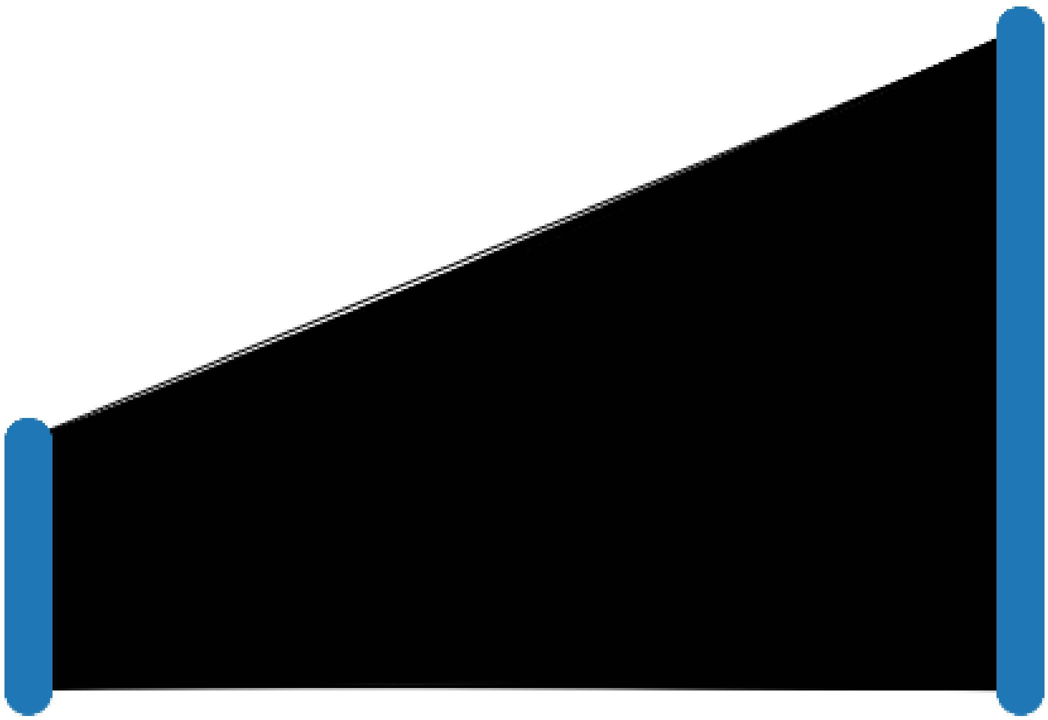
[4]:
```
edges = [tuple(x) for x in data.values.tolist()]

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')
```

[5]:
```
A = (B.subgraph(c) for c in nx.connected_components(B))
A = list(A)[0]
```

[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
```

[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, label=True)
plt.show()
```



[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
```

[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
```

[10]:
```
from gensim.models import Word2Vec
model = Word2Vec(walks, size=128, window=5)
```

[11]:
```
model.wv.vectors.shape  # 128-dimensional vector for each node in the graph
```

[11]:
```
(4703, 128)
```

[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.nodes[node_id]['label'] for node_id in node_ids]
```

[13]:
```
print(node_ids[:15], end='\n')
print(node_targets[:15], end='\n')
print(node_embeddings.shape, end='\n')
```

```
['a973', 'a967', 'a964', 'a1731', 'a970', 'a969', 'a1028', 'a965', 'a1003',
'a1057', 'm1094', 'a959', 'm67', 'm1100', 'a966']
['actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor',
'actor', 'actor', 'movie', 'actor', 'movie', 'movie', 'actor']
(4703, 128)
```

[14]:
```
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=[],[]
```

```python
    actors_nodes_index = []
    movies_nodes_index = []

    for index, value in enumerate(node_targets):
        if value == 'actor':
            actors_nodes_index.append(index)
        elif value == 'movie':
            movies_nodes_index.append(index)

    actor_embeddings = node_embeddings[actors_nodes_index]
    movie_embeddings = node_embeddings[movies_nodes_index]
    actor_nodes = np.asarray(node_ids)[actors_nodes_index]
    movie_nodes = np.asarray(node_ids)[movies_nodes_index]

    return actor_nodes, movie_nodes, actor_embeddings, movie_embeddings
```

[15]:
```python
actor_nodes, movie_nodes, actor_embeddings, movie_embeddings = ␣
↪data_split(node_ids,node_targets,node_embeddings)
```

Grader function - 1

[16]:
```python
def grader_actors(data):
    assert(len(data)==3411)
    return True
grader_actors(actor_nodes)
```

[16]: True

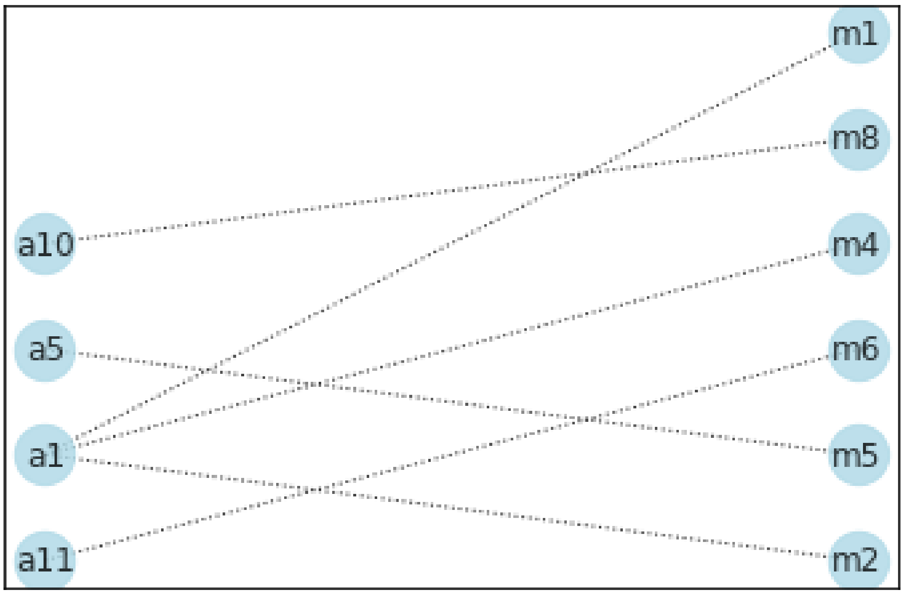
Grader function - 2

[17]:
```python
def grader_movies(data):
    assert(len(data)==1292)
    return True
grader_movies(movie_nodes)
```

[17]: True

[18]:
```python
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)
```



1 Task 1 : Apply clustering algorithm to group similar actors

1.0.1 1.1 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):
    tot_nodes = graph.number_of_nodes()
    largest_comp = max([len(list(graph.subgraph(c))) \
    for index, c in enumerate(nx.connected_components(graph))])

    return (1/number_of_clusters) *(largest_comp / tot_nodes)
```

Grader function - 3

[20]:
```
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)
```

[20]: True

1.0.2 1.2 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

[21]:
```
def cost2(graph,number_of_clusters):
    unique_movies = set()

    sum_of_degree = [j if 'a' in i else unique_movies.add(i)
    for subgraph in nx.connected_component_subgraphs(graded_graph)
    for i,j in dict(subgraph.degree).items()]

    sum_of_degree = sum([i for i in sum_of_degree if i])

    return (1 / number_of_clusters) * (sum_of_degree / len(unique_movies))
```

Grader function - 4

[22]:
```
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)
```

[22]: True

1.0.3 1.3 HyperParameter Tuning

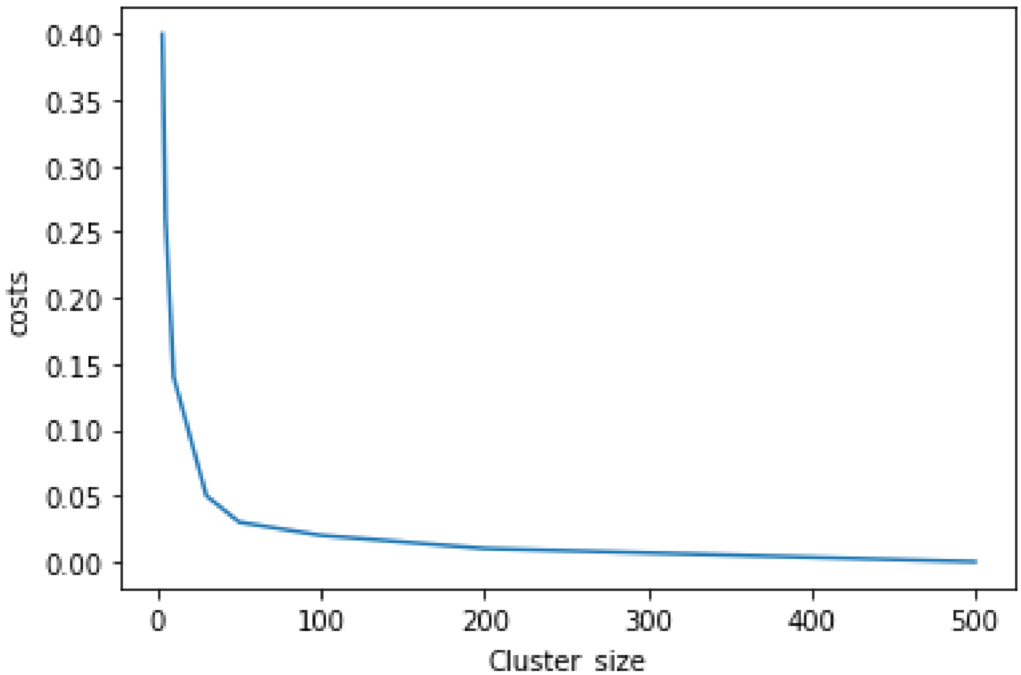
[24]:
```
clusters = [3, 5, 10, 30, 50, 100, 200, 500]
cost = dict()

for k in clusters:
    kmeans = KMeans(n_clusters=k, random_state=0).fit(actor_embeddings)

    cost_1 = []
    cost_2 = []

    for cluster_index, label in enumerate(np.unique(kmeans.labels_)):
```

```
        filter_arr = kmeans.labels_ == label
        cluster = np.where(filter_arr)[0].tolist() #ith cluster index

        actor_list = actor_nodes[cluster].tolist()
        edges = []
        movie_list = []

        for actor_ in actor_nodes[cluster]: #ith cluster actors
            movie_list.extend(list(A.neighbors(actor_))) #ith cluster movies
            edges.extend([(actor_, movie) for movie in movie_list]) #ith␣
↪cluster edges

        actor_list = list(set(actor_list))
        edges = list(set(edges))
        movie_list = list(set(movie_list))

        graph= nx.Graph()
        graph.add_nodes_from(actor_nodes, bipartite=0) # Add the node attribute␣
↪"bipartite"
        graph.add_nodes_from(movie_nodes, bipartite=1)
        graph.add_edges_from(edges) #get graph

        cost_1.append(cost1(graph, k))#ith cluster cost1
        cost_2.append(cost2(graph, k))#ith cluster cost2

    cost[str(k)] = sum(cost_1) * sum(cost_2) # product of total cost
```

[25]:
```
cost
```

[25]:
```
{'10': 0.14235594301509671,
 '100': 0.019876674463108666,
 '200': 0.010747395279608772,
 '3': 0.4016585158409525,
 '30': 0.05658090580480542,
 '5': 0.264597065702743,
 '50': 0.0364150542207102,
 '500': 0.004497980012757825}
```

[26]:
```
cluster = [3, 5, 10, 30, 50, 100, 200, 500]
costs = [0.40, 0.26, 0.14, 0.05, 0.03, 0.02, 0.01, 0.00]

plt.plot(cluster, costs)
plt.xlabel('Cluster_size')
plt.ylabel('costs')
plt.show();
```



1.0.4 1.4 KMeans Clustering

[27]:
```
kmeans = KMeans(n_clusters=3, random_state=0).fit(actor_embeddings)
kmeans.labels_.shape, actor_embeddings.shape
```

[27]: ((3411,), (3411, 128))

1.0.5 1.5 t-SNE Dimensionality Reduction

[28]:
```
from sklearn.manifold import TSNE
transform = TSNE #PCA

trans = transform(n_components=2)
node_embeddings_2d = trans.fit_transform(actor_embeddings)
```

[29]:
```
df = np.column_stack((node_embeddings_2d, kmeans.labels_))
df = pd.DataFrame(data=df, columns=['dim_0', 'dim_1', 'label'])
df.head()
```

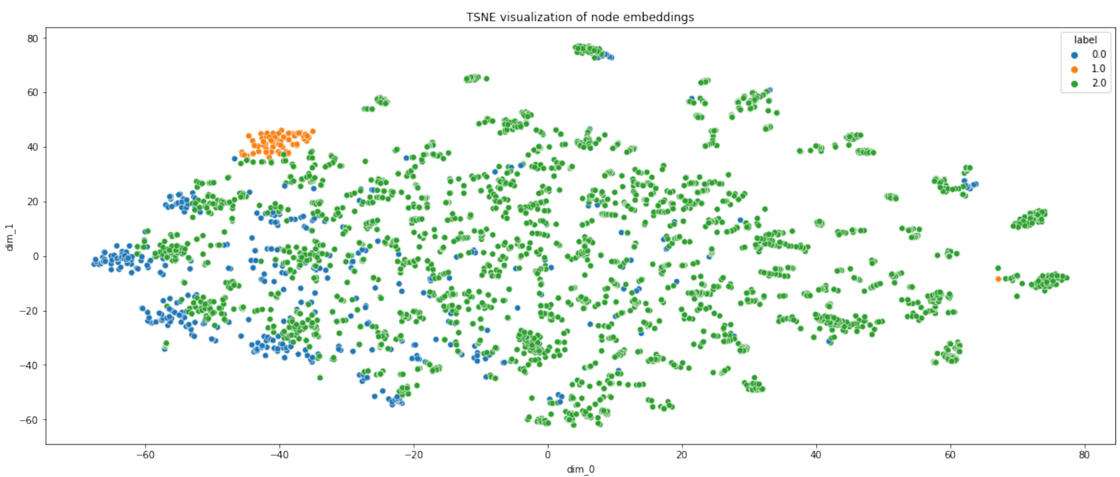
[29]:

	dim_0	dim_1	label
0	-38.006275	44.858200	1.0
1	-42.704880	44.824699	1.0
2	-38.377029	44.984688	1.0
3	-46.646774	35.692848	0.0

4 -39.660126 46.048958   1.0

1.0.6 1.6 EDA - Group Similar Actors

[30]:
```
import seaborn as sns
import random

number_of_colors = 3

color = ["#"+''.join([random.choice('0123456789ABCDEF') for j in range(6)])
             for i in range(number_of_colors)]

plt.figure(figsize=(20,8))

sns.scatterplot(data=df, x='dim_0', y='dim_1',hue='label', palette=sns.
 ↪color_palette("tab10", 3), s=40);
plt.title('{} visualization of node embeddings'.format(transform.__name__))
plt.show();
```



2 Task 2 : Apply clustering algorithm to group similar movies

2.0.1 2.1 Calculating COST1

[31]:
```
def cost1(graph, number_of_clusters):
    tot_nodes = graph.number_of_nodes()
    largest_comp = max([len(list(graph.subgraph(c))) \
    for index, c in enumerate(nx.connected_components(graph))])

    return (1/number_of_clusters) *(largest_comp / tot_nodes)
```

2.0.2 2.2 Calculating COST2

[32]:
```
def cost2(graph, number_of_clusters):
    unique_actors = set()

    sum_of_degree = [j if 'm' in i else unique_actors.add(i)
    for c in nx.connected_components(graded_graph) \
    for i,j in dict(graded_graph.subgraph(c).degree).items()]

    sum_of_degree = sum([i for i in sum_of_degree if i])

    return (1 / number_of_clusters) * (sum_of_degree / len(unique_actors))
```

2.0.3 2.3 HyperParameter Tuning

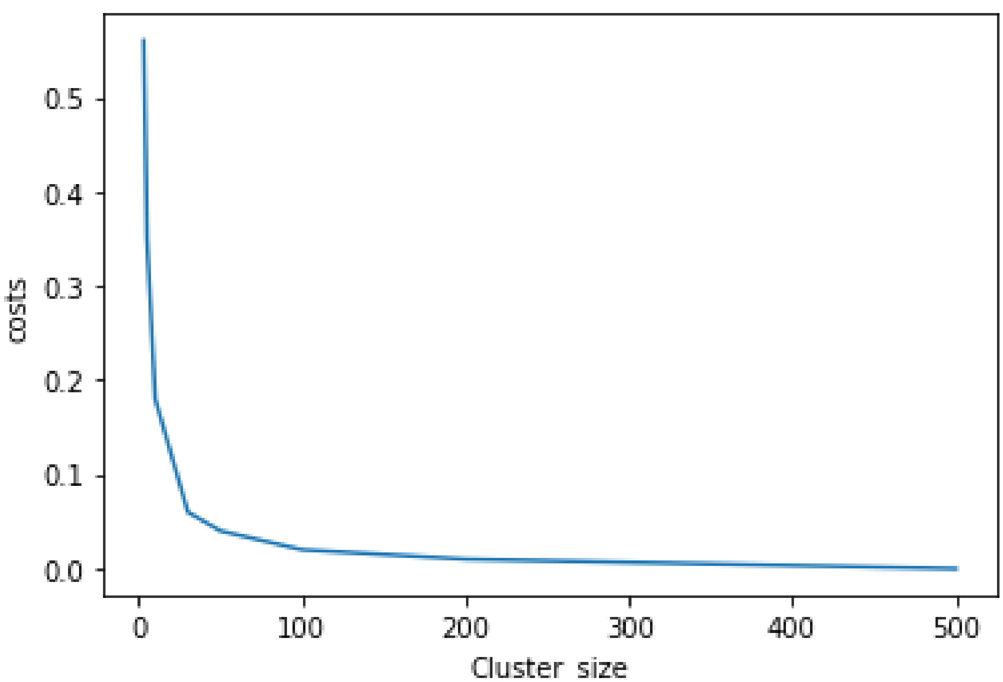
[33]:
```
from networkx.algorithms import bipartite

clusters = [3, 5, 10, 30, 50, 100, 200, 500]
cost = dict()

for k in clusters:

    kmeans = KMeans(n_clusters=k, random_state=0).fit(movie_embeddings)

    cost_1 = []
    cost_2 = []

    for cluster_index, label in enumerate(np.unique(kmeans.labels_)):
        filter_arr = kmeans.labels_ == label
        cluster = np.where(filter_arr)[0].tolist() #ith cluster index

        movie_list = movie_nodes[cluster].tolist()
        edges = []
        actor_list = []

        for movie_ in movie_nodes[cluster]: #ith cluster actors
            actor_list.extend(list(A.neighbors(movie_))) #ith cluster movies
            edges.extend([(movie_, actor) for actor in actor_list]) #ith␣
↪cluster edges

        actor_list = list(set(actor_list))
        edges = list(set(edges))
        movie_list = list(set(movie_list))

        graph= nx.Graph()
        graph.add_nodes_from(actor_nodes, bipartite=0) # Add the node attribute␣
↪"bipartite"
```

```
        graph.add_nodes_from(movie_nodes, bipartite=1)
        graph.add_edges_from(edges) #get graph

        cost_1.append(cost1(graph, k))#ith cluster cost1
        cost_2.append(cost2(graph, k))#ith cluster cost2

    cost[str(k)] = sum(cost_1) * sum(cost_2) # product of total cost
```

[34]:
```
cost
```

[34]:
```
{'10': 0.18626408675313633,
 '100': 0.021343823091643604,
 '200': 0.011394322772698303,
 '3': 0.5599617265575165,
 '30': 0.06590474165426326,
 '5': 0.35409313204337667,
 '50': 0.04126514990431643,
 '500': 0.005256857325111576}
```

[35]:
```
cluster = [3, 5, 10, 30, 50, 100, 200, 500]
costs = [0.56, 0.35, 0.18, 0.06, 0.04, 0.02, 0.01, 0.00]

plt.plot(cluster, costs)
plt.xlabel('Cluster_size')
plt.ylabel('costs')
plt.show();
```



2.0.4 2.4 KMeans Clustering

[36]:
```
kmeans = KMeans(n_clusters=3, random_state=0).fit(movie_embeddings)
kmeans.labels_.shape, movie_embeddings.shape
```

[36]: ((1292,), (1292, 128))

2.0.5 2.5 t-SNE Dimensionality Reduction

[37]:
```
from sklearn.manifold import TSNE
transform = TSNE #PCA

trans = transform(n_components=2)
node_embeddings_2d = trans.fit_transform(movie_embeddings)
```

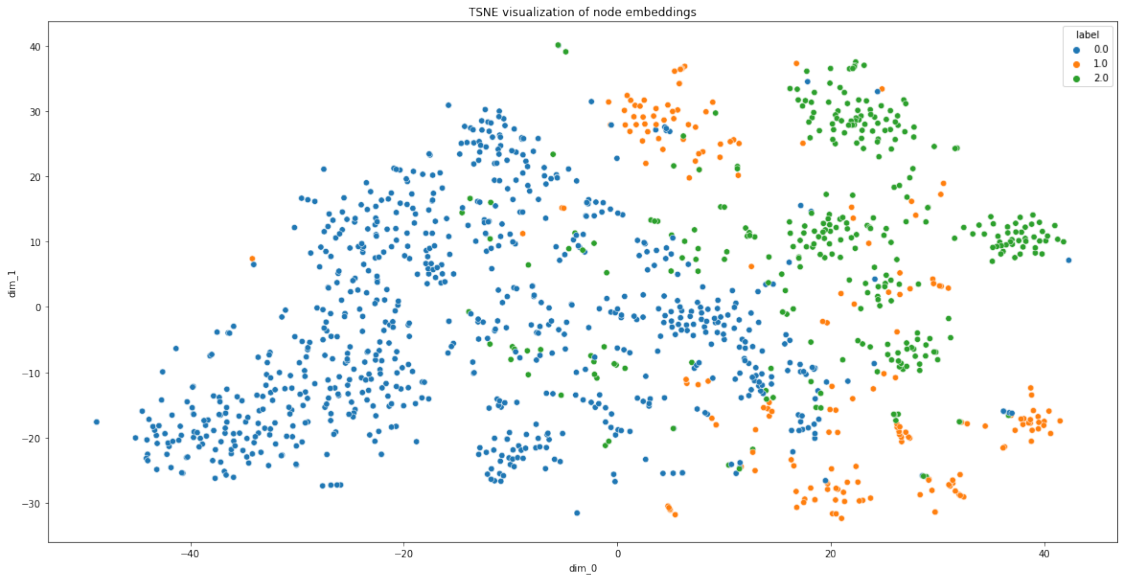
[38]:
```
df = np.column_stack((node_embeddings_2d, kmeans.labels_))
df = pd.DataFrame(data=df, columns=['dim_0', 'dim_1', 'label'])
df.head()
```

[38]:

	dim_0	dim_1	label
0	16.820282	-30.699364	1.0
1	16.795731	37.309212	1.0
2	21.020885	-32.380444	1.0
3	28.384100	-28.754696	1.0
4	20.167734	-31.655575	1.0

2.0.6 2.6 EDA - Group Similar Movies

[39]:
```
import seaborn as sns
import random

plt.figure(figsize=(20,10))

sns.scatterplot(data=df, x='dim_0', y='dim_1',hue='label', palette=sns.
 ↪color_palette("tab10", 3), s=40);
plt.title('{} visualization of node embeddings'.format(transform.__name__))
plt.show();
```


TSNE visualization of node embeddings
label
0.0
1.0
2.0
dim_1
dim_0
40
30
20
10
0
−10
−20
−30
−40
−20
0
20
40