Program -3 DBSCAN

Dataset: – This dataset contains 8580 text records in sparse format. In this dataset labels are not provided.

Objective : The objective of this assignment is to use Kmeans and variation of DBSCAN to assign each text record to a cluster ranging from 1 to k > 100

Approach –

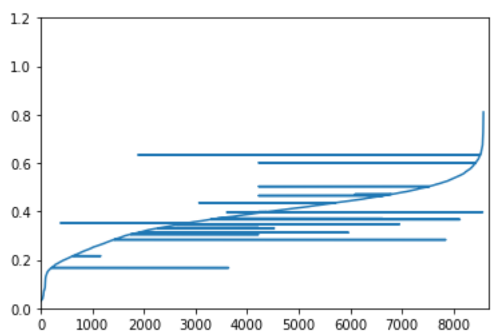
1) All the records are read using **csr_read** function and was saved into mat1 variable

2) mat1 is transformed in to a count matrix to a normalized tf or tf-idf representation using **TfidfTransformer().fit_transform** module. Library to **import from sklearn.feature_extraction.text import TfidfTransformer**

3)Performed Dimensionality Reduction using SVD by keeping the following parameters **n_components=200,n_iter=7, random_state=42.** Resultant matrix is saved into **mat_dr.**

4) After that each value is normalized to unit scale using **normalizer.fit**_transform and saved into **points.** Also, a copy of points is made as points_or for future reference. Variable **explained_variance** gives the Explained Variance of SVD and I got it as **61 Percent.**

5) Epsilon value is computed using K-nearest neighbor and graph is plotted for the same. Parameter used **n_neighbors = 30, metric='cosine'**



6) This matrix saved is then given to KMeans to find clusters . Number of clusters used is 500. After that centroids obtained for all the cluster is saved in **Cluster_center** and labels for the respective clusters are saved in **cluster_label.**

7)In order to make cluster of clusters I decided to use center points obtained through Kmeans and find core points , border points and noise points for it. I tried with different values of eps and min points but combination of **eps = .60** and **minpts = 3** worked best for me. I got **282 core points** with this , **90 border points , and 128 Noise points**

8) G = nx.Graph() was used to Invoke graph instance to visualize the cluster

9) Number of clusters obtained were 7

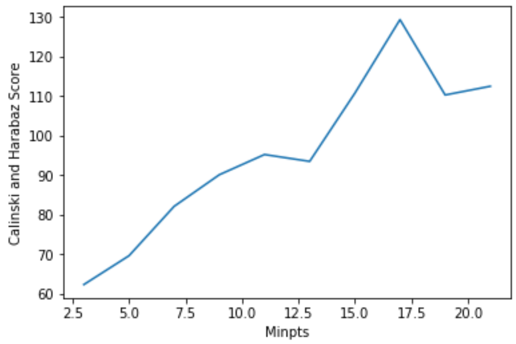
10) In order to make the prediction cluster_label is again copied into variable named KmeansLabels. A dictionary names cluster_dict was made and all the labels obtained through kmeans was converted into the corresponding DBSCAN labels . Also an additional cluster was made to adjust all the noise points. Final prediction was saved into clustpred file.

11)For the purpose of inter cluster evaluation I have used calinski harabaz metrics. I ran the code for different episilon and k values and got the following table. I plotted the graph for epsilon value 0.7.

Rank. -15

NMI – 0.4781

k\eps	0.7
3	62.346865
5	69.6394353
7	82.1397973
9	90.1328761
11	95.2451444
13	93.5108298
15	110.755648
17	129.380118
19	110.314061
21	112.519327




```python
import numpy as np
import pandas as pd
import random
from collections import defaultdict
from scipy.sparse import csr_matrix
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import Normalizer
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
import numpy


# In[849]:


#reading csr file and returning csr matrix by calculating ind,ptr and values in the file.
def csr_read(fname, ftype="csr", nidx=1):

    with open(fname) as f:
        lines = f.readlines()
        nrows = len(lines)
        ncols = 0
        nnz = 0
        for i in range(nrows):
            p = lines[i].split()
            if len(p) % 2 != 0:
                raise ValueError("Invalid CSR matrix")
            nnz += len(p)/2
            nnz=int(nnz)
            for j in range(0, len(p), 2):
                cid = int(p[j]) - nidx
                if cid+1 > ncols:
                    ncols = cid+1

    val = np.zeros(nnz, dtype=np.float)
    ind = np.zeros(nnz, dtype=np.int)
    ptr = np.zeros(nrows+1, dtype=np.long)
    n = 0
    for i in range(nrows):
        p = lines[i].split()
        for j in range(0, len(p), 2):
            ind[n] = int(p[j]) - nidx
            val[n] = float(p[j+1])
            n += 1
        ptr[i+1] = n

    assert(n == nnz)

    return csr_matrix((val, ind, ptr), shape=(nrows, ncols), dtype=np.float)
```

```python

#performing dimensionality reduction using SVD to 150 components and normalising the output to
inhibit impact of document length on distance computation
print("Performing dimensionality reduction using LSA")
svd = TruncatedSVD(n_components=200,n_iter=7, random_state=42)
normalizer = Normalizer(copy=False)
mat_dr=svd.fit_transform(mat1)
points=normalizer.fit_transform(mat_dr)
points_or=points
explained_variance = svd.explained_variance_ratio_.sum()
print(explained_variance)


# In[854]:


#Computing eps for minpts using K-distance graph and elbow point in it.
nbrs = NearestNeighbors(n_neighbors=30,metric='cosine').fit(points)
distances, indices = nbrs.kneighbors(points)
t=distances[:,-1]
i=indices[:,0]
a=np.sort(t)
plt.axis([0, 8680, 0, 1.2])
plt.plot(i,a)


# In[855]:


import sklearn
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import scale

import sklearn.metrics as sm
from sklearn.metrics import confusion_matrix,classification_report


# In[856]:


clustering = KMeans(n_clusters = 500, random_state = 42)
clustering.fit(points)


# In[857]:



print(clustering.labels_)
cluster_label= clustering.labels_
# cluster_len
#cluster_len
X1=pd.Series(clustering.labels_)
X1.value_counts()


# In[858]:


#Plotting the data points again on the graph and visualize how the data has been clustered
plt.scatter(mat_dr[:,0],mat_dr[:,1], c=clustering.labels_, cmap='rainbow')


# In[859]:


#Plotting the data points again on the graph and visualize the centroids in black
plt.scatter(mat_dr[:,0], mat_dr[:,1], c=clustering.labels_, cmap='rainbow')
plt.scatter(clustering.cluster_centers_[:,0] ,clustering.cluster_centers_[:,1], color='black')


# In[860]:


Cluster_center = np.array(clustering.cluster_centers_)


# In[861]:


Cluster_center.shape

```

```
#DBSCAN tryout using NetworkX
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx


# In[880]:


points = Cluster_center
eps = .60
minPts = 3


# In[881]:


# Find core points with minPts as 5
neighborhoods = []
core = []
border = []
noise = []

for i in range(len(points)):
    neighbors = []
    for p in range(0, len(points)):
        # If the distance is below eps, p is a neighbor
        if np.linalg.norm(points[i] - points[p]) < eps:
            neighbors.append(p)
    neighborhoods.append(neighbors)
    # If neighborhood has at least minPts, i is a core point
    if len(neighbors) > minPts :
        core.append(i)

print("core: ", core)


# In[882]:


print(len(core))


# In[883]:


# Find border points
for i in range(len(points)):
    neighbors = neighborhoods[i]
    # Look at points that are not core points
    if len(neighbors) <= minPts:
        for j in range(len(neighbors)):
            # If one of its neighbors is a core, it is also in the core point's neighborhood,
            # thus it is a border point rather than a noise point
            if neighbors[j] in core:
                border.append(i)
                # Need at least one core point...
                break

print("border: ", border)
#print(len(border))


# In[889]:


print(len(border))


# In[884]:


#Find noise points
for i in range(len(points)):
    if i not in core and i not in border:
        noise.append(i)

print("noise", noise)
#print("Length of noise points :", len(noise))


# In[890]:


print("Length of noise points :", len(noise))
```

```


# Invoke graph instance to visualize the cluster
G = nx.Graph()


# In[892]:


# Add nodes -- core points + border points
nodes = core+border
G.add_nodes_from(nodes)


# In[893]:


# Create neighborhood
for i in range(len(nodes)):
    for p in range(len(nodes)):
        # If the distance is below the threshold, add a link in the graph.
        if p != i and np.linalg.norm(points[nodes[i]] - points[nodes[p]]) <= eps:
            G.add_edges_from([(nodes[i], nodes[p])])


# In[894]:


# List the connected components / clusters
clusters = list(nx.connected_components(G))
print("# clusters:", len(clusters))
print("clusters: ", clusters)


# In[878]:


# Visualise the graph
plt.subplot(111)
nx.draw_circular(G, with_labels=True, font_weight='bold')
plt.show()


# In[879]:



kmeansLabels = cluster_label

cluster_dict = {}
for i, cluster in enumerate(clusters):
    for cluster_center in cluster:
        cluster_dict[cluster_center] = i
#np.array(clusters)

for cluster_center in noise:
    cluster_dict[cluster_center] = -1

#  To convert kmeans labels to dbscan labels
labels_final = [cluster_dict[x] for x in kmeansLabels]

labels_final = np.array(labels_final)
labels_final[labels_final==-1] = max(labels_final)+1
labels_final[labels_final==0] = max(labels_final)+1

print(pd.Series(labels_final).value_counts())


# In[895]:


fh = open("clustpred.dat", "w")
for x in labels_final:
    fh.write("{}\n".format(x))
fh.close()

```

```
from sklearn import metrics


# In[897]:


s=metrics.calinski_harabaz_score(points_or, labels_final)

print(s)


# In[898]:


#Precomputed scores
score=
[62.34686498,69.6394,82.13979732,90.13287605,95.24514444,93.51082975,110.755648,129.3801181,110.3140
608,112.5193268]

k=[3,5,7,9,11,13,15,17,19,21]
plt.xlabel('Minpts')
plt.ylabel('Calinski and Harabaz Score')
plt.plot(k,score)
```