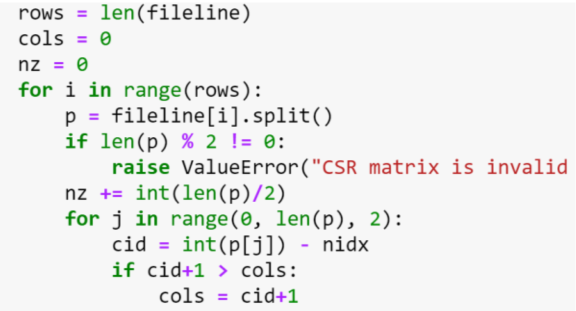
PR1: Text Clustering

Mamidi Veena

015226198

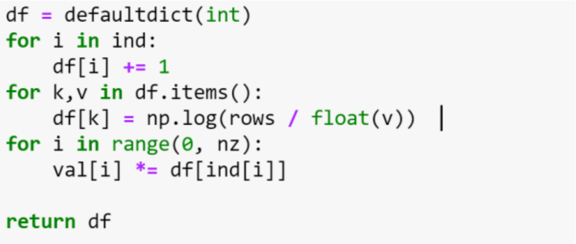
Code for Bisecting K-Means:

Step 1: Reading the CSR matrix from a text file.



```
rows = len(fileline)
cols = 0
nz = 0
for i in range(rows):
    p = fileline[i].split()
    if len(p) % 2 != 0:
        raise ValueError("CSR matrix is invalid
    nz += int(len(p)/2)
    for j in range(0, len(p), 2):
        cid = int(p[j]) - nidx
        if cid+1 > cols:
            cols = cid+1
```

Step 2: Build a function for scaling a given CSR matrix by Inverse Document Frequency and return document frequency.

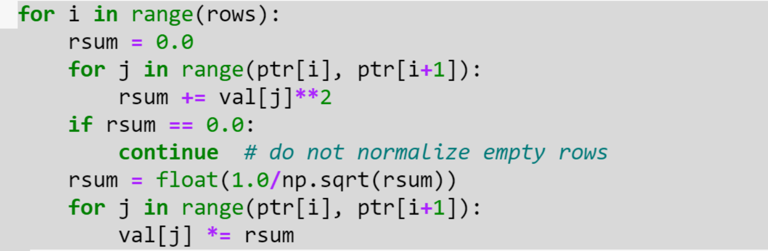


```
df = defaultdict(int)
for i in ind:
    df[i] += 1
for k,v in df.items():
    df[k] = np.log(rows / float(v))
for i in range(0, nz):
    val[i] *= df[ind[i]]

return df
```

Step 3: Performing L2 Normalization

The below is the code for normalization of the values in the matrix.

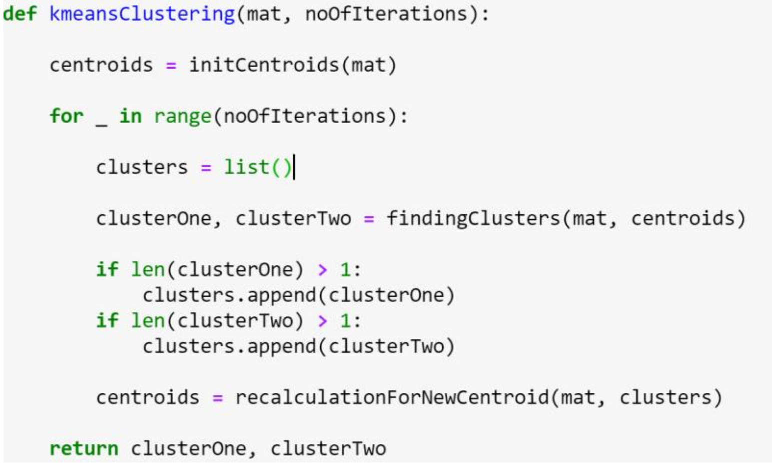
Empty Rows are skipped and the normalization is continued.



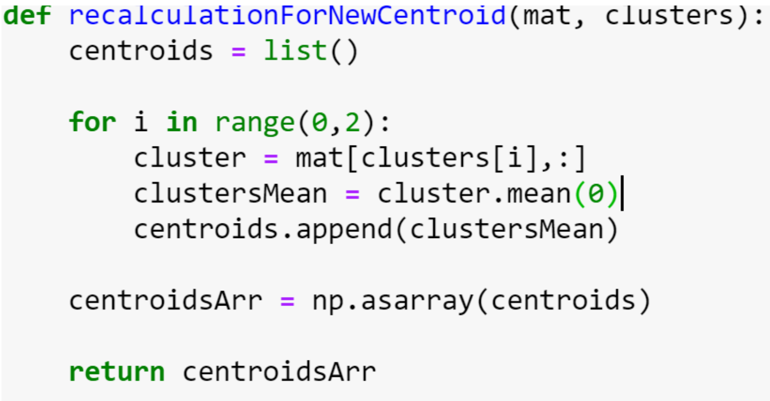
```
for i in range(rows):
    rsum = 0.0
    for j in range(ptr[i], ptr[i+1]):
        rsum += val[j]**2
    if rsum == 0.0:
        continue  # do not normalize empty rows
    rsum = float(1.0/np.sqrt(rsum))
    for j in range(ptr[i], ptr[i+1]):
        val[j] *= rsum
```

PR1: Text Clustering

Step 4: In K means Clustering, we keep recalculating for a new centroid by calculating mean and keep assigning new centroid



```
def kmeansClustering(mat, noOfIterations):

    centroids = initCentroids(mat)

    for _ in range(noOfIterations):

        clusters = list()

        clusterOne, clusterTwo = findingClusters(mat, centroids)

        if len(clusterOne) > 1:
            clusters.append(clusterOne)
        if len(clusterTwo) > 1:
            clusters.append(clusterTwo)

        centroids = recalculationForNewCentroid(mat, clusters)

    return clusterOne, clusterTwo
```



```
def recalculationForNewCentroid(mat, clusters):
    centroids = list()

    for i in range(0,2):
        cluster = mat[clusters[i],:]
        clustersMean = cluster.mean(0)
        centroids.append(clustersMean)

    centroidsArr = np.asarray(centroids)

    return centroidsArr
```

Step 5: Bisecting K means Clustering

Firstly we try to determine the cluster to be dropped and for calculating it we use the below function where we return the Index of the cluster to be dropped.

PR1: Text Clustering

```
def calSSE(mat, clusters):

    list_SSE = list()
    array_SSE = []

    for clu in clusters:
        members = mat[clu,:]
        SSE = np.sum(np.square(members - np.mean(members)))
        list_SSE.append(SSE)

    array_SSE = np.asarray(list_SSE)
    dropCluIndex = np.argsort(array_SSE)[-1]

    return dropCluIndex
```

Secondly, we divide dataset into 2 clusters using K means clustering and keep recalculating the clusters for the number of iterations mentioned.

```
while len(clusters) < k:

    dropCluIndex = calSSE(mat, clusters)
    droppedCluster = clusters[dropCluIndex]

    clusterOne, clusterTwo = kmeansClustering(mat[droppedCluster,:], noOfIterations)
    del clusters[dropCluIndex]

    actualClusterOne = list()
    actualClusterTwo = list()
    for index in clusterOne:
        actualClusterOne.append(droppedCluster[index])

    for index in clusterTwo:
        actualClusterTwo.append(droppedCluster[index])

    clusters.append(actualClusterOne)
    clusters.append(actualClusterTwo)

labels = [0] * mat.shape[0]
```

PR1: Text Clustering

For number of iterations: 8

```
For K= 3 Calinski Harabasz Score is 55.717248
```

For number of iterations: 10

For K= 3 Calinski Harabasz Score is 63.080976

For K= 5 Calinski Harabasz Score is 47.311063

For K= 7 Calinski Harabasz Score is 46.035433

For K= 9 Calinski Harabasz Score is 43.216868

For K= 11 Calinski Harabasz Score is 38.217769

For number of iterations: 12

For K= 3 Calinski Harabasz Score is 58.776345

For K= 5 Calinski Harabasz Score is 50.399152

For K= 7 Calinski Harabasz Score is 48.510865

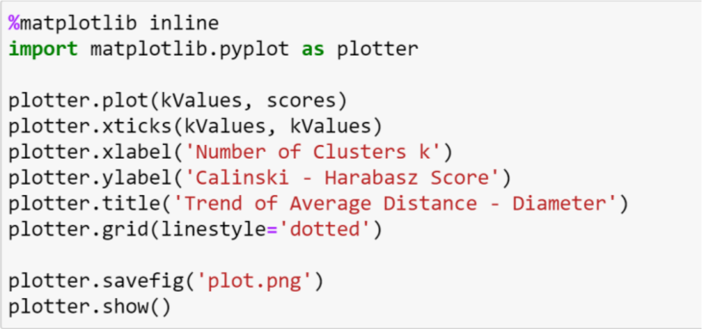
For K= 9 Calinski Harabasz Score is 41.896178

For K= 11 Calinski Harabasz Score is 39.016886

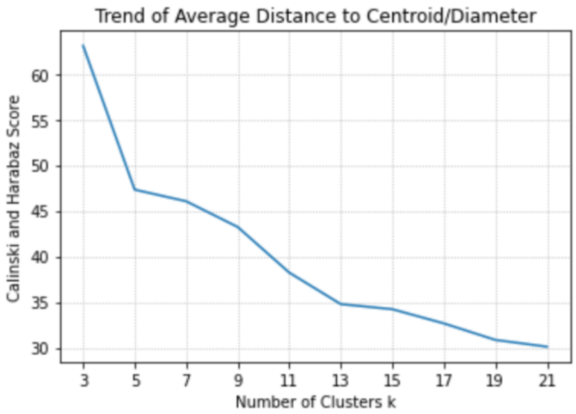
On observation, I got a better Calinski and Harabasz with number of iterations as 10.

PR1: Text Clustering

We can then plot a graph of number of clusters against Calinski- Harbasz Score to study the trend of average distance to diameter of the cluster



```
%matplotlib inline
import matplotlib.pyplot as plotter

plotter.plot(kValues, scores)
plotter.xticks(kValues, kValues)
plotter.xlabel('Number of Clusters k')
plotter.ylabel('Calinski - Harabasz Score')
plotter.title('Trend of Average Distance - Diameter')
plotter.grid(linestyle='dotted')

plotter.savefig('plot.png')
plotter.show()
```


Trend of Average Distance to Centroid/Diameter
Calinski and Harabaz Score
Number of Clusters k