**APPROACH**

**Problem Statement**: Input is provided in the form of three text files. They are 1) CORPUS\_ONE.txt, 2) CORPUS\_TWO.txt and 3) TEST\_CORPUS.txt

Each file consists of three columns:

1. COLUMN 1 - UNIQUE ID
2. COLUMN 2 - TEXT referred to as 'title' of individual documents (rows) in the corpus
3. COLUMN 3 - TEXT referred to as 'abstract' of individual documents (rows) in the corpus

**Goal**

1. Treat each file as individual corpus and run a clustering algorithm on each of the corpus

2. Test it against 'TEST\_CORPUS.txt'

3. You may combine both 'CORPUS\_ONE.txt' and 'CORPUS\_TWO.txt' as one corpus and repeat the task

4. You may please employ any pre-processing on the text in all CORPUS files before training or employing a model (There are embedded xml tags like <strong>, </strong> etc., which may be cleaned)

5. Report your findings against different algorithms (at least two different algorithms) and an inference of which worked best with these corpus and why?

**Approach**:

1. Initially cleaning of all the embedded xml tags was done for all the three text files.
2. Performed the analysis and tokenization of the text using NLP and NLTK.
3. Carried out modelling of the data using various clustering algorithms.

List of clustering algorithms used:

1. K-Means Clustering Algorithm: It is an iterative algorithm that partitions the dataset into K pre-defined distinct non-overlapping clusters where each data point belongs to **only one group.**
2. Mini-Batch K-Means Clustering Algorithm: It is a variant of the [K-Means](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans) algorithm which uses mini-batches to reduce the computation time, while still attempting to optimise the same objective function.
3. DBSCAN Clustering Algorithm: It views clusters as areas of high density separated by areas of low density.
4. Mean Shift Clustering Algorithm: It aims to discover blobs in a smooth density of samples. It is a centroid based algorithm, which works by updating candidates for centroids to be the mean of the points within a given region.
5. Spectral Clustering Algorithm: It performs a low-dimension embedding of the affinity matrix between samples, followed by clustering.

**Steps**:

1. Testing on CORPUS\_ONE.txt

**1] Performing manual tokenization of the text from the files.**

1. Loading of the text file on Jupyter notebook.

corpus\_one = open("CORPUS\_ONE.txt","r+")

corpus\_one = open("CORPUS\_ONE.txt","rt")

1. Split the words inside the text file using whitespace

Cleaning the text can be done by listing the words and in python with split() function does this.

words=text.split()

1. **Select the words after splitting**

import re

words=re.split(r'\W+',text)

1. **Remove any punctuation marks if they are present inside the file**

[maketrans()](https://docs.python.org/3/library/stdtypes.html#str.maketrans) to create a mapping table. Inside the empty mapping table, third function lists all the characters to be removed at the time of the translation process.

import strings

corpus\_new=str.maketrans('','',string.punctuation)

Python has a function called [translate()](https://docs.python.org/3/library/stdtypes.html#str.translate) that will map one set of characters to another.

stripped=[w.translate(corpus\_new) for w in words]

1. **Normalizing Case**

This converts all the words into the lowercase.

words=[word.lower() for word in words]

## 2] Performing Tokenization and Cleaning with NLTK of the text from the files.

## a) Splitting into the sentences

## In this step we split the text into sentences.

## NLTK has in-built function called sent\_tokenize() to split the text into sentences.

## from nltk import sent\_tokenize

## sent\_tokens=sent\_tokenize(text)

## b) Splitting the strings into words (tokens)

## NLTK provides the function word\_tokenize() to split the strings into tokens.

## It also automatically splits the tokens based on whitespaces and punctuation marks.

## from nltk.tokenize import word\_tokenize

## tokens = word\_tokenize(text)

c) **Filter out punctuation**

isalpha() function can be used for filtering out the unwanted tokens.

words=[word for word in tokens if word.isalpha()]

d) **Segregating the stop words**

from nltk.corpus import stopwords

stop\_words=stopwords.words('english')

e) **Stemming**: This process reduces each word to its root or base

e.g, “scintillating”, “scintillations” gets reduced to the stem “scintillat”.

The syntax for this process is as follows:

from nltk.stem.porter import PorterStemmer

porter = PorterStemmer()

stemmed\_words = [porter.stem(word) for word in tokens]

**Data Modelling on CORPUS\_ONE.txt using different clustering algorithms**

1. Importing the necessary packages
2. Loading the dataset in the form of dataframe
3. Performing Exploratory Data Analysis (EDA) steps like finding the shape, size, data types, columns, describing the data, finding the information of data, checking for the null values.
4. There are 3 columns in the dataset: 1. UNIQUE\_ID, 2. TITLE, 3.ABSTRACT
5. Assign two columns TITLE and ABSTRACT to two new variables as:

documentA=data['TITLE']

documentB=data['ABSTRACT']

1. Using K-Means Clustering Algorithm

TF-IDF Vectorizer:  It is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document and the inverse document frequency of the word across a set of documents.

1. Import the required packages

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.cluster import KMeans

from sklearn.metrics import accuracy\_score,precision\_score,recall\_score,f1\_score,classification\_report

1. Converting the document into vector

vectorizer1 = TfidfVectorizer()

X1 = vectorizer1.fit\_transform(documentA)

X2 = vectorizer2.fit\_transform(documentB)

1. Defining the model

model=KMeans(n\_clusters=2,init='k-means++',n\_init=10,max\_iter=300,tol=0.0001,precompute\_distances='auto',

verbose=0,random\_state=10,copy\_x=True,n\_jobs=5,algorithm='auto')

model.fit(X1)

1. Identifying the cluster centers

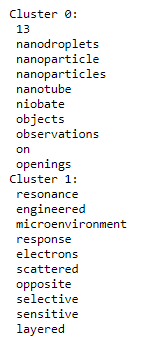
centroid=model.cluster\_centers\_.argsort()[:,:]

1. Getting the feature names of the Vectorizer

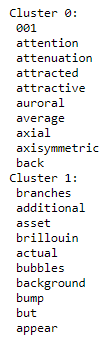
terms1=vectorizer1.get\_feature\_names()

terms2=vectorizer2.get\_feature\_names()

1. Clusters of TITLE values



1. Clusters of ABSTRACT values



Later, we find out the number of clusters and the features present inside one cluster.

1. Predicting the text sentence

X2=vectorizer1.transform(["Acoustic damping is another relevant factor in AEH applications as it influences the quality of the acoustic resonator. "])

y\_predict=model.predict(X2)

The following code results in the output as

Prediction

[1]

It means that the sentence is present in cluster 1.

**Conclusion**: Using this algorithm, we get two clusters for TITLE and ABSTRACT values namely cluster 0 and cluster 1.

Also, the predicted sentence is present in the first cluster.

1. Using Mini-Batch K-Means Clustering Algorithm
2. Import the required packages

from sklearn.cluster import MiniBatchKMeans

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

1. Transform the vectorize into features

tfidfvectorizer1 = TfidfVectorizer()

tfidfvectorizer1.fit(data.ABSTRACT.values)

features1 = tfidfvectorizer1.transform(data.ABSTRACT.values)

1. Define and predict the model

model = MiniBatchKMeans(n\_clusters=5, random\_state=10)

model.fit(features1)

model.predict(features1)

1. Finding the labels inside the model

model.labels\_

1. Reducing the number of dimensions using Principal Component Analysis (PCA)

pca=PCA(random\_state=10,n\_components=2)

trimmed\_features1=pca.fit\_transform(features1.toarray())

trimmed\_features1

1. Define the cluster centres

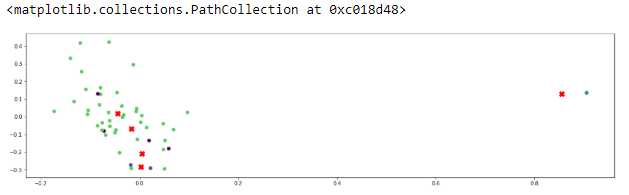
model.cluster\_centers\_

1. Perform PCA on cluster centers

trim\_cluster\_centers=pca.transform(model.cluster\_centers\_)

trim\_cluster\_centers

1. Plot the graph for the clusters after dimensionality reduction



**Fig 1: Scatter plot for Mini-Batch K-Means Clustering Algorithm**

The red marks represent the centroids and blue, green points represent the features inside the cluster.

1. Calculating the homogeneity and silhouette scores

**Silhouette Score**:

Compute the mean Silhouette Coefficient of all samples.

The Silhouette Coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. The Silhouette Coefficient for a sample is

(b - a) / max(a, b)

Code:

from sklearn.metrics import silhouette\_score

print(" Silhouette score of CORPUS ONE = ", silhouette\_score(features1, labels=model.predict(features1))\*100)

**Homogeneity Score**: Homogeneity metric of a cluster labelling given a ground truth.

A clustering result satisfies homogeneity if all of its clusters contain only data points which are members of a single class.

Code:

from sklearn.metrics import homogeneity\_score

print(" Homogeneity score of CORPUS ONE = ", homogeneity\_score(data.ABSTRACT, model.predict(features1))\*100)

**Conclusion**: Mini-Batch K-means clustering algorithm does not group the features into clusters as per the plot for CORPUS\_ONE data. So it is not recommended algorithm on this data.

1. Using DBSCAN (Density-Based Spatial Clustering of Applications with Noise) Clustering Algorithm
2. Import the required packages

from sklearn.cluster import DBSCAN

from sklearn import metrics

from sklearn.datasets import make\_blobs

from sklearn.preprocessing import StandardScaler

1. Make the blobs

It generates isotropic Gaussian blobs for clustering.

X2, true\_label = make\_blobs(n\_samples=50, cluster\_std=0.4,random\_state=0)

1. Scale the data

X2= StandardScaler().fit\_transform(X2)

1. Define the model

dbscan\_clustering = DBSCAN(eps=3, min\_samples=10).fit(X2)

core\_samples\_mask = np.zeros\_like(dbscan\_clustering.labels\_, dtype=bool)

core\_samples\_mask[dbscan\_clustering.core\_sample\_indices\_] = True

1. Label the clusters

dbscan\_clustering.labels\_

label=dbscan\_clustering.labels\_

n\_clusters=len(set(label)) - (1 if -1 in label else 0)

n\_noise = list(label).count(0)

1. Calculating the parameters

print('Estimated number of clusters: %d' % n\_clusters)

print('Estimated number of noise points: %d' % n\_noise)

print("Homogeneity: %0.3f" % metrics.homogeneity\_score(true\_label, label))

print("Completeness: %0.3f" % metrics.completeness\_score(true\_label, label))

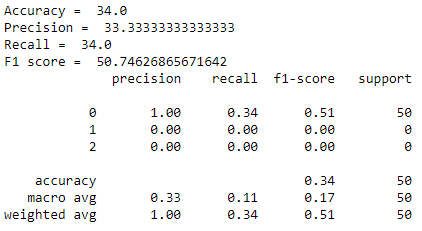
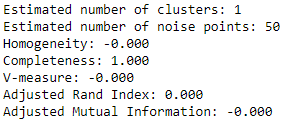
print("V-measure: %0.3f" % metrics.v\_measure\_score(true\_label, label))

print("Adjusted Rand Index: %0.3f"

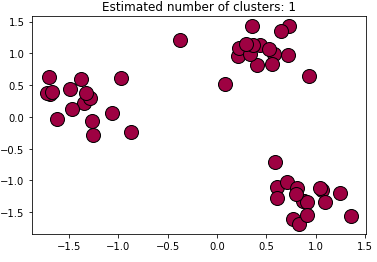
% metrics.adjusted\_rand\_score(true\_label, label))

print("Adjusted Mutual Information: %0.3f"

% metrics.adjusted\_mutual\_info\_score(true\_label, label))



1. Plotting the graph



**Fig 2: Plot representing the DBSCAN Clustering Algorithm**

**Conclusion**: DBSCAN clustering algorithm generates only one cluster. Also many points are not coming together but are deviated. There is no single group of feature points as per the plot.

1. Using Mean Shift Clustering Algorithm
2. Import the package and generate blob

from sklearn.cluster import MeanShift

X2,true\_label=make\_blobs(n\_samples=150,cluster\_std=1.0)

1. Define the model

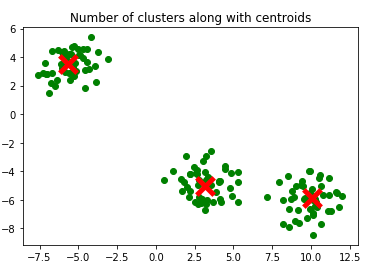
mean\_shift\_clustering = MeanShift(bandwidth=2).fit(X2)

1. Define the cluster centres

centers=mean\_shift\_clustering.cluster\_centers\_

mean\_shift\_clustering.labels\_

1. Plotting the graph



**Fig 3: Plot representing Mean Shift Clustering Algorithm**

**Conclusion**: Mean Shifting Clustering algorithm generates proper clusters along with marking of centroids. So it is highly recommended to use this algorithm.

1. Using Spectral Clustering Algorithm
2. Importing the required packages

from sklearn.cluster import SpectralClustering

from sklearn.preprocessing import StandardScaler, normalize

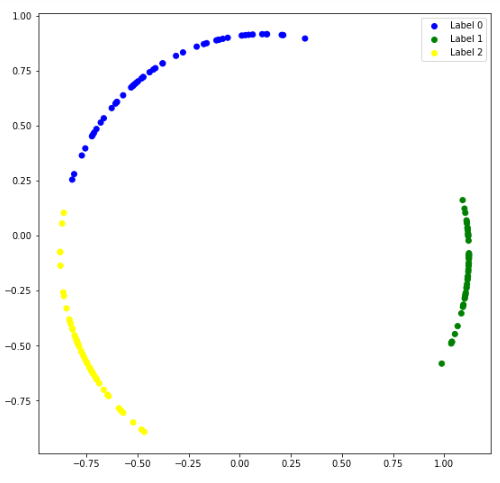
from sklearn.decomposition import PCA

from sklearn.metrics import silhouette\_score

1. Using affinity=’rbf’ (Radial basis function)

rbf\_spectral\_model= SpectralClustering(n\_clusters = 3, affinity ='rbf')

rbf\_labels = rbf\_spectral\_model.fit\_predict(X\_principal)



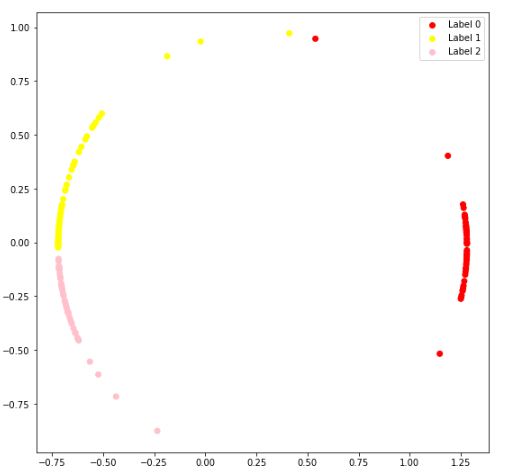
**Fig 4: Plot representing Spectral Clustering Algorithm for rbf**

In this method, feature labels of a particular cluster are placed very close to each other.

1. Using affinity=’nearest\_neighbors’

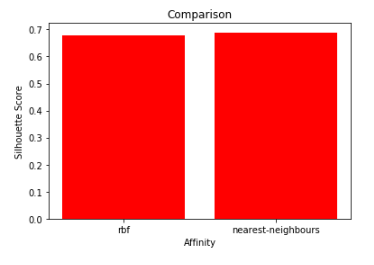
nn\_spectral\_model= SpectralClustering(n\_clusters = 3, affinity =’nearest\_neighbors’)

nn\_labels = nn\_spectral\_model.fit\_predict(X\_principal)



**Fig 5: Plot representing Spectral Clustering Algorithm for nearest\_neighbors**

In this method, few feature labels of label 0 and label 1 are away from their respective clusters.



**Fig 6: Bar plot representing for comparing BF and nearest\_neighbours**

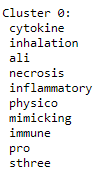
**Conclusion**: In spectral clustering algorithm, ‘**nearest\_neighbors**’ affinity has more Silhouette score than the ‘**rbf’** affinity in CORPUS\_ONE text file.

Perform the similar activities for CORPRUS\_TWO.txt, combination of CORPUS\_ONE.txt and CORPRUS\_TWO.txt and TEST\_CORPUS.txt.

**Results for CORPRUS\_TWO.txt**

1. Using K-Means Clustering Algorithm

Clusters for TITLE and ABSTRACT values

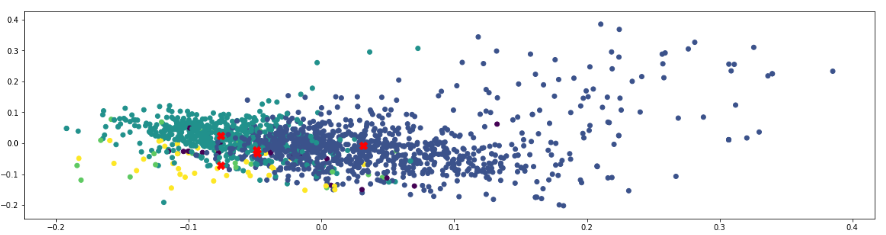
 

Predicted sentence is present in the cluster 0.



**Conclusion**: It generates only one cluster from the CORPUS\_TWO text file. It is not recommended to use K-Means clustering algorithm for this text file.

1. Using Mini-Batch K-Means Clustering Algorithm



**Fig 7: Scatter plot for Mini-Batch K-Means Clustering Algorithm**

In this figure, 4 clusters are generated. They are in light blue, dark blue, yellow and violet colours.

Calculating the parameters:

Homogeneity score of CORPUS TWO = 11.566181910148494

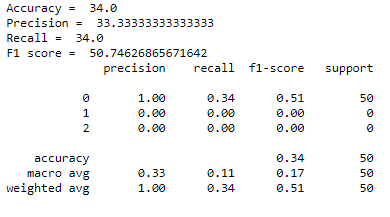
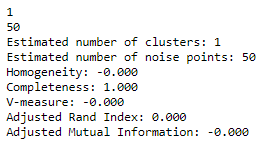
Silhouette score of CORPUS TWO = 0.3624960321288239

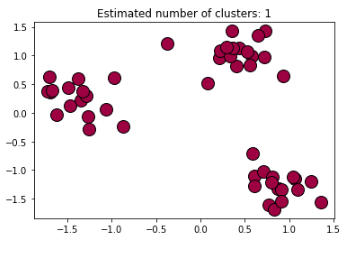
**Conclusion**: Using Mini-Batch K-Means Clustering algorithm, 4 clusters are generated.

Also the centroid is marked exactly in between the clusters in CORPUS\_TWO file. This change is observed in this file as compared to CORPUS\_ONE. It is recommended to use this algorithm for CORPUS\_TWO text file.

1. Using DBSCAN Clustering Algorithm

Calculating the parameters

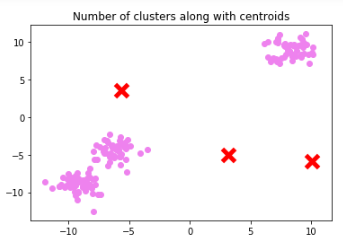




**Fig 8: Plot representing the DBSCAN Clustering Algorithm**

**Conclusion**: DBSCAN algorithm generates one cluster for CORPUS\_TWO text file like that of CORPUS\_ONE text file.

1. Using Mean Shift Clustering Algorithm

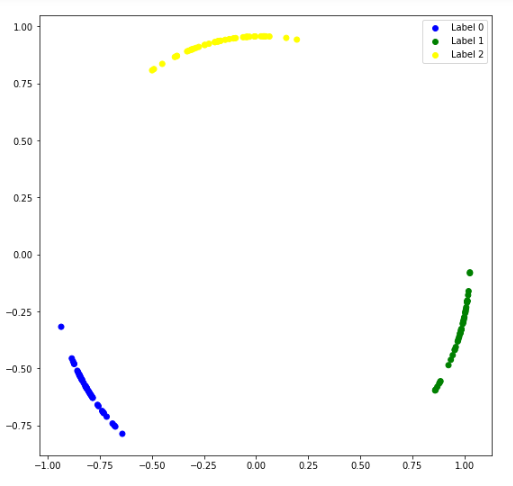


**Fig 9: Plot representing Mean Shift Clustering Algorithm**

**Conclusion**: Mean Shift Clustering Algorithm generated 3 clusters but the centroid is away from cluster. So for the text file CORPUS\_TWO it is not recommended to use this algorithm unlike CORPUS\_ONE.

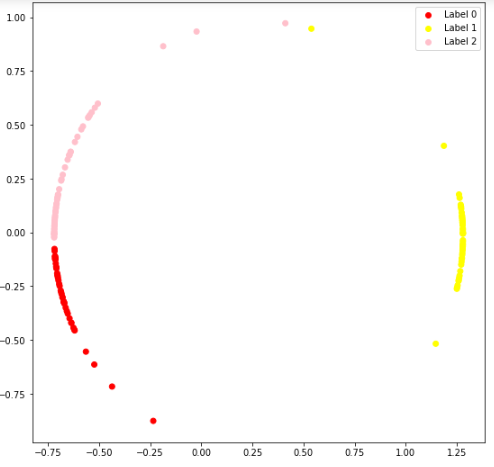
1. Using Spectral Clustering Algorithm

5.1) Using affinity=’rbf

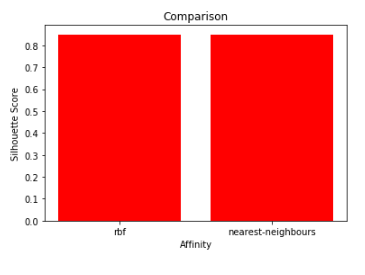


**Fig 10: Plot representing Spectral Clustering Algorithm for rbf**

5.2) Using affinity=’nearest\_neighbors’



**Fig 11: Plot representing Spectral Clustering Algorithm for nearest\_neighbours**



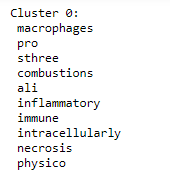
**Fig 12: Bar plot representing for comparing BF and nearest\_neighbours**

**Conclusion**: In CORPUS\_TWO text file, both the methods of spectral clustering algorithm i.e. rbf and nearest neighbours have same silhouette scores as compared to the ones in CORPUS\_ONE text file. So this algorithm can be used for this corpus.

**Results for combination of CORPUS\_ONE.txt and CORPRUS\_TWO.txt**

1. Using K-Means Clustering Algorithm

Cluster values for ABSTRACT

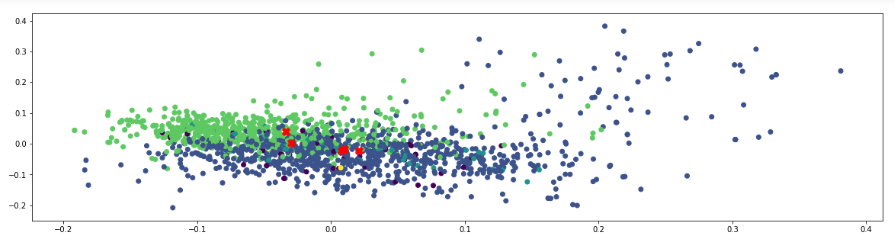


Predicted sentence is present in cluster 0.



**Conclusion**: This algorithm generated only one cluster after we combine both the corpuses.

1. Using Mini-Batch K-Means Clustering Algorithm



**Fig 13: Scatter plot for Mini-Batch K-Means Clustering Algorithm**

Calculating the parameters

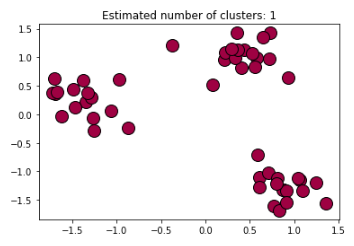
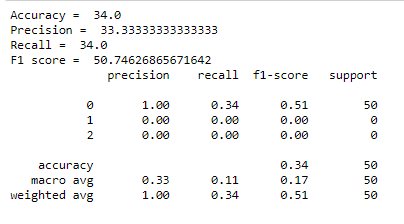
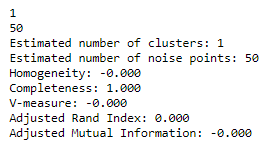
Homogeneity score of CORPUS ONE and TWO = 12.287302840464527

Silhouette score of CORPUS ONE and TWO = -0.03937061427710632

**Conclusion**: Mini-batch K-Means clustering algorithm generates 4 clusters with centroids placed inside the cluster as in CORPUS\_TWO text file. So this algorithm is recommended for combination of two corpuses.

1. Using DBSCAN Clustering Algorithm

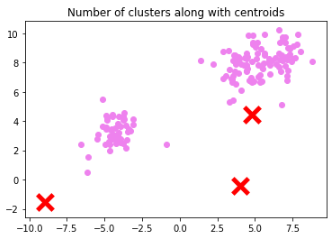
Calculating the parameters



**Fig 14: Plot representing the DBSCAN Clustering Algorithm**

**Conclusion:** DBSCAN algorithm generated one cluster for this merged text file like CORPUS\_ONE and CORPUS\_TWO.

1. Using Mean Shift Clustering Algorithm

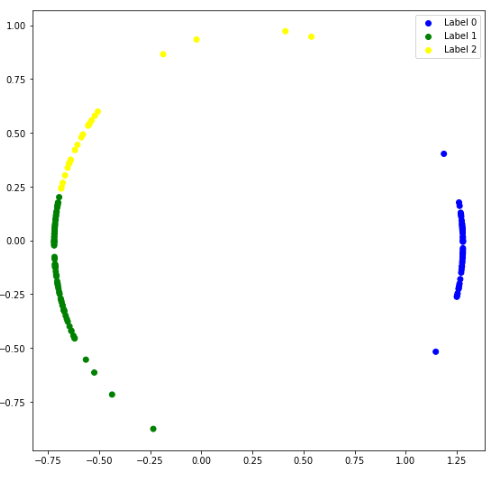


**Fig 15: Plot representing Mean Shift Clustering Algorithm**

**Conclusion**: Mean Shift clustering algorithm generated 2 clusters and 3 centroids are placed away from the clusters. So it is not recommended to run this algorithm for the merged text file.

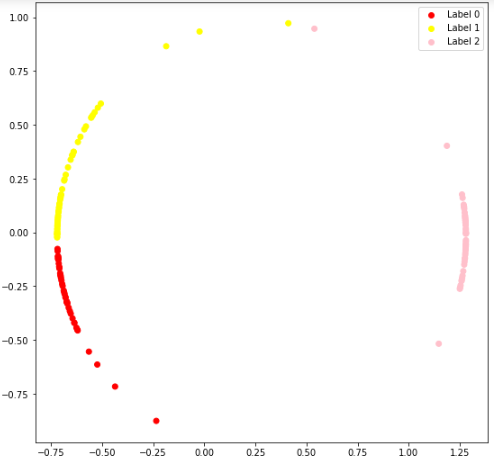
1. Using Spectral Clustering Algorithm

5.1) Using affinity=’rbf

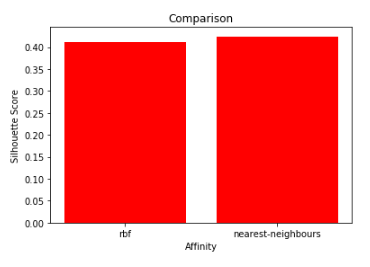


**Fig 16: Plot representing Spectral Clustering Algorithm for rbf**

5.2) Using affinity=’nearest\_neighbors’



**Fig 17: Plot representing Spectral Clustering Algorithm for nearest\_neighbours**



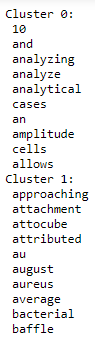
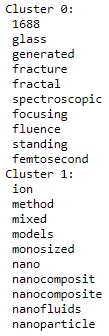
**Fig 18: Bar plot representing for comparing BF and nearest\_neighbours**

**Conclusion**: In spectral clustering algorithm, ‘**nearest\_neighbors**’ affinity has more Silhouette score than the ‘**rbf’** affinity same as in the CORPUS\_ONE text file.

**Results for TEST\_CORPRUS.txt**

1. Using K-Means Clustering Algorithm

Cluster values for TITLE and ABSTRACT



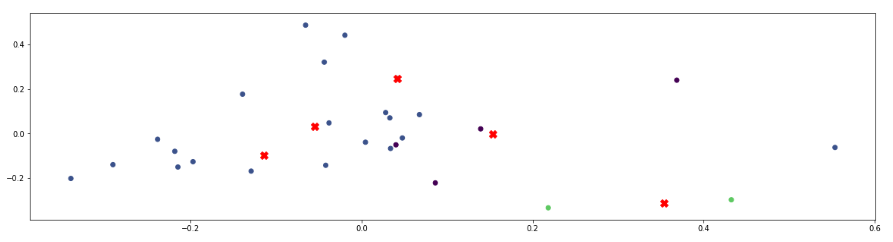
Predicted sentence is present in cluster 0



**Conclusion**: It generates 2 clusters namely cluster 0 and 1 and prediction happens in the cluster 0.

It is same as in CORPUS\_ONE text file.

1. Using Mini-Batch K-Means Clustering Algorithm



**Fig 19: Scatter plot for Mini-Batch K-Means Clustering Algorithm**

Calculating the parameters

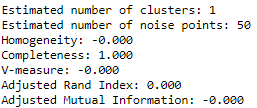
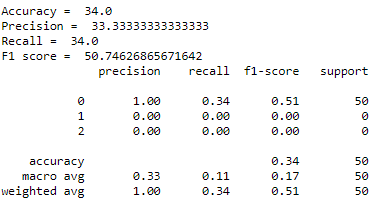
Homogeneity score of CORPUS ONE = 28.354897665496264

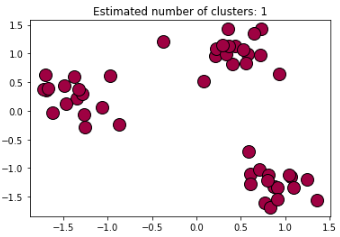
Silhouette score of CORPUS ONE = 0.42613694111319317

**Conclusion**: Using this algorithm, it does not create clusters and centroids are not placed properly. So it is not recommended for TEST\_CORPUS text file.

1. Using DBSCAN Clustering Algorithm

Calculating the parameters

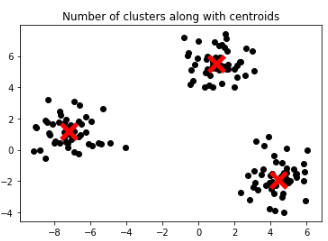
 



**Fig 20: Plot representing the DBSCAN Clustering Algorithm**

**Conclusion**: It generates one cluster for all the text files.

1. Using Mean Shift Clustering Algorithm

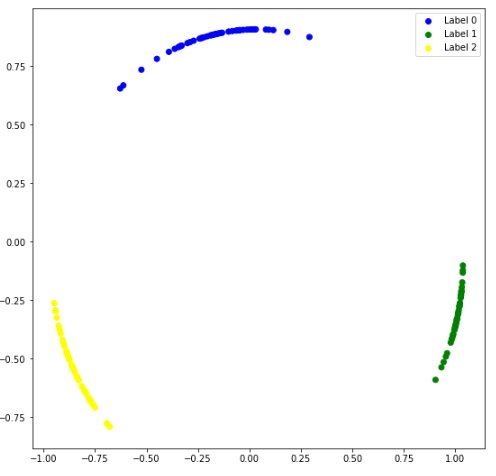


**Fig 21: Plot representing Mean Shift Clustering Algorithm**

**Conclusion**: 3 clusters are being generated using this algorithm with centroids placed in the middle of the cluster. This algorithm is recommended for TEST\_CORPUS text file.

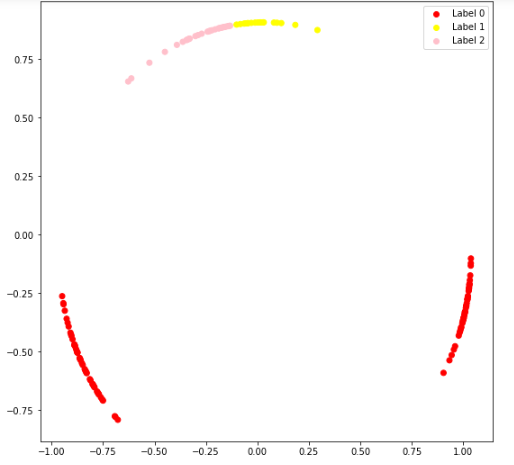
1. Using Spectral Clustering Algorithm

5.1) Using affinity=’rbf

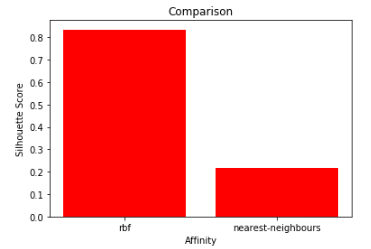


**Fig 22: Plot representing Spectral Clustering Algorithm for rbf**

5.2) Using affinity=’nearest\_neighbors’



**Fig 23: Plot representing Spectral Clustering Algorithm for nearest\_neighbours**



**Fig 24: Bar plot representing for comparing BF and nearest\_neighbours**

**Conclusion**: In this algorithm, affinity ‘rbf’ has very high silhouette score as compared to ‘nearest neighbours’.

**Inferences**:

1. K-Means clustering is only used for finding the clusters present in the text file. It cannot be used for getting differences in the clusters with respect to placement of centroid, calculating silhouette scores etc. It generated 2 clusters for CORPUS\_ONE and TEST\_CORPUS text files. For other files it generates only one cluster.
2. Mini-Batch K-Means clustering algorithm generated better results than K-Means clustering algorithm.

It generates proper clusters for CORPUS\_TWO text file and merged text file of CORPUS\_ONE and CORPUS\_TWO with centroid lying exactly between the clusters. But for the rest files it does not generate any cluster. The data points are scattered everywhere in the given space.

1. DBSCAN clustering generates only one cluster for all the text files. So we cannot figure out any differences among them. It gives the almost the ssame results for all the text files.
2. Mean Shift clustering algorithm generates clusters and marks the centroids in between the clusters. For CORPUS\_ONE and TEST\_CORPUS it gives correct results. But for the rest files it does create a cluster but centroids are placed outside it.
3. In Spectral clustering algorithm, affinity of rbf and nearest neighbours gives same silhouette scores in CORPUS\_TWO text file. But for CORPUS\_ONE and merged text file nearest neighbours has slightly higher value. For TEST\_CORPUS file, rbf has very high silhouette score compared to nearest neighbours.