CSE506: Introduction to Data Mining
Assignment-3: Report

Nimisha Gupta (2019315)
Vani Sikka (2019395)

Q1) Clustering modeling

Importing the dataset values
We were given the datasets, 'covtype_train.csv' in the form of a comma separated file. We imported this using the pandas.read_csv() function and converted it into a dataframe 'data'.

Initial Analysis

1. We used 'pandas.DataFrame.head()', to view rows of the dataframe
2. We used 'pandas.DataFrame.shape' which gives number of rows and columns of the dataframe
3. true_labels variable is initialized as the 'target' column of the dataset

```
true_labels= data['target']
```

Next, a new variable 'label_encoder' was created.

```
label_encoder = preprocessing.LabelEncoder()
```

preprocessing.LabelEncoder() is used to transform non-numerical labels into numerical ones.

This label_encoder is then used to call fit_transform() function for each column of the dataset.

```
data['Elevation']= label_encoder.fit_transform(data['Elevation'])
```

It returns the encoded labels of the column which it takes as input.
The dataset 'data' is again viewed using the .head() function and this time we see the encoded form of it.
(https://www.datacamp.com/community/tutorials/categorical-data)

A new variable 'dt' is created which is made the array form of 'data'.

tsne() function is used to reduce the dimensionality of our data into the number specified, i.e. 2. Next, fit_transform() is used for the encoded labels.

```
data = tsne(n_components=2).fit_transform(dt)
```

1. K-mean Clustering

K-mean Clustering is a method of data cluster analysis and is used to quantize vectors. It partitions n observations into k clusters. Each of the n observations belongs to the cluster with the nearest mean.

For this `KMeans` is imported from `sklearn.cluster`
Here we have kept the value of k as 7.

Kmeans clustering on dataset 'data' is performed as follows:

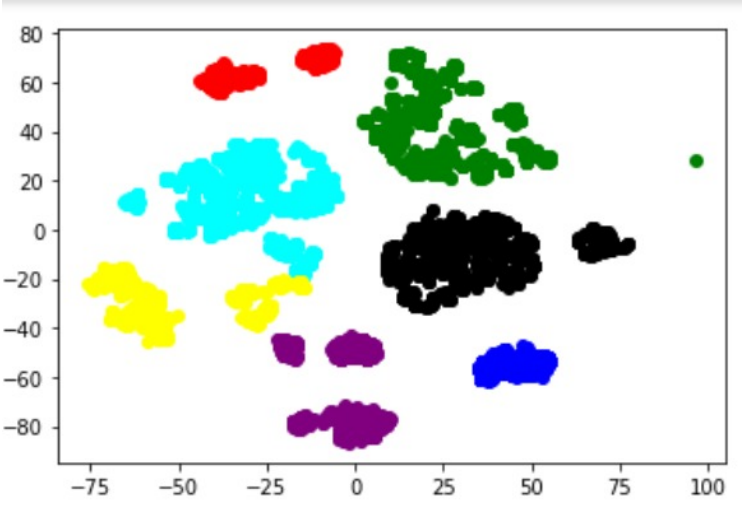
```
kmeans= KMeans(n_clusters=7,max_iter=60,random_state=100).fit(data)
kmean_labels = kmeans.labels_
```

a) Finding Centroid
'kmean_centriods' variable is created and stores the dimensions of the 7 centroid
The number of centroids will be equal to the number of clusters, i.e. 7.

```
kmean_centroids = kmeans.cluster_centers_
print(kmean_centroids)
```

The print statement displays them.

b) Visualization of the clusters:
The clusters along with their centroids are plotted using matplotlib.pyplot
The scatterplot() is used for the same.



c) Comparison with true label count:
'predict' function is used to get the clustering of the data which is stored in a variable 'kmean_labels'. To compare this cluster formation with the true label count, we used the adjusted_mutual_info_score() function.

d) We found out the number of labels in each cluster using the following code

```
label_list = list(np.unique(kmean_labels))
PtsInCluster = []
for val in label_list:
    PtsInCluster.append(list(kmean_labels).count(val))
```

2. DBSCAN Clustering

It stands for Density-based Spatial Clustering of Applications with Noise (DBSCAN) clustering method. It produces more reasonable results than *k*-means across a variety of different distributions.

DBSCAN clustering on dataset 'data' is performed as follows:

```
dbscan = DBSCAN(eps=17, min_samples=4).fit(data)
```

a) Finding Centroid:

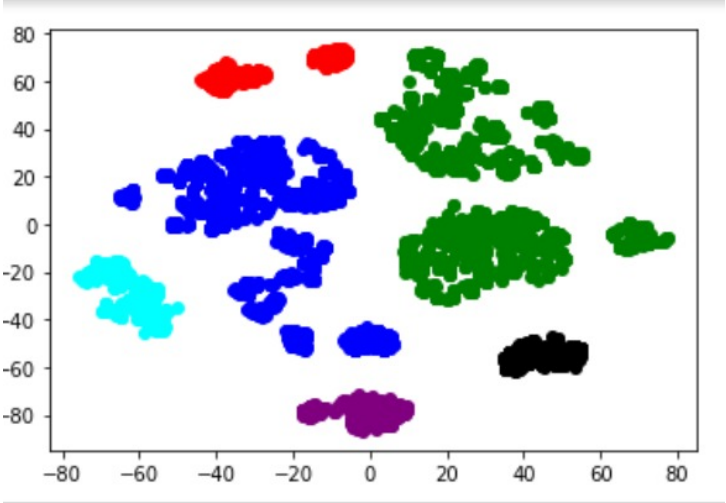
For this Nearest Centroid() is imported

The following code is used to find and print the centroids.

```
clf=NearestCentroid()
X=dbscan.fit_predict(data)
clf.fit(data,X)
print((clf.centroids_))
```

b) Visualization of the clusters:

The clusters along with their centroids are plotted using matplotlib.pyplot. The scatterplot() is used for the same.



c) Comparison with true label count:

'predict' function is used to get the clustering of the data which is stored in a variable 'dbscan_labels'. To compare this cluster formation with the true label count, we used the adjusted_mutual_info_score() function.

d) We found out the number of labels in each cluster using the following code.

```
label_list = list(np.unique(dbscan_labels))
PtsInCluster = []
for val in label_list:
  PtsInCluster.append(list(dbscan_labels).count(val))
```

3. Agglomerative Clustering

Agglomerative clustering is a hierarchical clustering which is used to group objects into clusters based on their similarities.
For this `AgglomerativeClustering` is imported from `sklearn.cluster`

Agglomerative clustering on dataset 'data' is performed as follows:

```
Agmc = AgglomerativeClustering(n_clusters=7).fit(data)
```

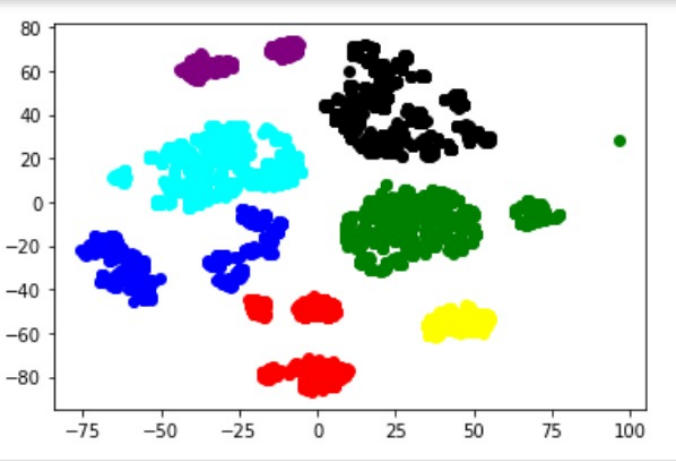
Here we have kept the number of clusters as 7.

a) Finding Centroid:
For this Nearest Centroid() is imported.
The following code is used to find and print the centroids.

```
clf=NearestCentroid()
X=Agmc.fit_predict(data)
clf.fit(data,X)
print(clf.centroids_)
```

b) Visualization of the clusters:
The clusters along with their centroids are plotted using matplotlib.pyplot.
The scatterplot() is used for the same.

c) Comparison with true label count:
'predict' function is used to get the clustering of the data which is stored in a variable 'agmc_labels'. To compare this cluster formation with the true label count, we used the adjusted_mutual_info_score() function.



d) We found out the number of labels in each cluster using the following code

```
label_list = list(np.unique(Agmc_labels))
PtsInCluster = []
for val in label_list:
  PtsInCluster.append(list(Agmc_labels).count(val))
```

4. Gaussian Clustering

It stands for Density-based Spatial Clustering of Applications with Noise (DBSCAN) clustering method. It produces more reasonable results than *k*-means across a variety of different distributions.

For this `GaussianMixture` is imported from `sklearn.cluster`.

Gaussian clustering on dataset 'data' is performed as follows:

```
gmm = GaussianMixture(n_components=7)
gmm.fit(data)
labels = gmm.predict(data)
```

Here we have kept the number of components as 7.

a) <u>Finding Centroid:</u>

For this Nearest Centroid() is imported

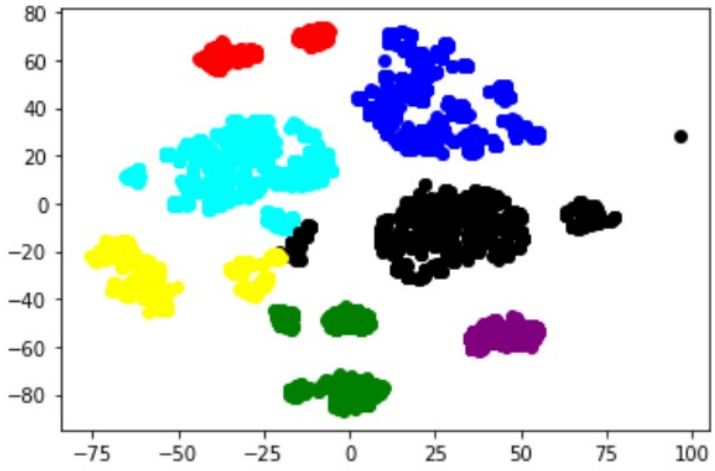
The following code is used to find and print the centroids.

```
clf=NearestCentroid()
X=gmm.fit_predict(data)
clf.fit(data,X)
print((clf.centroids_))
```

b) <u>Visualization of the clusters:</u>

The clusters along with their centroids are plotted using matplotlib.pyplot.

The scatterplot() is used for the same.



c) Comparison with true label count:
'predict' function is used to get the clustering of the data which is stored in a variable 'gsm_labels'. To compare this cluster formation of the gaussian based method with the true label count, we used the adjusted_mutual_info_score() function.

d) We found out the number of labels in each cluster using the following code.

```
label_list = list(np.unique(gsm_labels))
PtsInCluster = []
for val in label_list:
  PtsInCluster.append(list(gsm_labels).count(val))
```

To compare the cluster formation of the gaussian based method with the other clustering methods, we used the adjusted_mutual_info_score() function. For this, 'predict' function is used to get the clustering of the data which is stored in a labels variable [kmean_labels, dbscan_labels, Agmc_labels, gsm_labels]. These variables are then compared.

From the values we got kmean clustering is more similar to gaussian based method. Next comes agglomerative clustering and the least similar to gaussian based method is DBSCAN.

Q2)

The predict function is run.

A new variable ‘label_encoder’ was created.

```
label_encoder = preprocessing.LabelEncoder()
```

preprocessing.LabelEncoder() is used to transform non-numerical labels into numerical ones.

This label_encoder is then used to call fit_transform() function for each column of the dataset.

```
data['Elevation']= label_encoder.fit_transform(data['Elevation'])
```

It returns the encoded labels of the column which it takes as input.

We made variable ‘labels’ which stores the pandas dataframe of the ‘target’ column.
Then we used kmeans clustering on the data. Here we kept the value of k as 7.
Then cluster centers are computed and cluster index is predicted for each sample using function ‘fit_predict’.

‘`cluster`’ variable is defined as the list of the above prediction. Mapping of this cluster and labels is done using the `get_Mapping` function.

The get_Mapping function is run.

Tells the mapping between labels and clusters.

The get_LabelCount function is run.

It computes the number of labels in each cluster and returns this list.

Finally the prediction list is computed using the mapping_list we got and this is then returned.