Network Anomaly Detection

Pattern Recognition

Name	ID
Toka Ashraf Abo Elwafa	19015539
Rowan Nasser Edrees	19015686
Nada Mohamed Ibrahim	19016782

Problem Statement

The exponential growth of network traffic has led to an increase in network anomalies, such as cyber attacks, network failures, and hardware malfunctions. Network anomaly detection is a critical task for maintaining the security and stability of computer networks. The objective of this assignment is to understand how K-Means and Normalized Cut algorithms can be used for network anomaly detection.

Code Flow

- ### Imported Libraries

```
[ ]  import gzip
     import pandas as pd
     import numpy as np
     import random
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split
     from sklearn.neighbors import kneighbors_graph
     from mpl_toolkits.mplot3d import Axes3D
     from sklearn.cluster import KMeans
     from scipy.spatial.distance import pdist, squareform
     from sklearn.impute import SimpleImputer
     from joblib import Parallel, delayed
     from sklearn.neighbors import NearestNeighbors
     from sklearn.metrics import f1_score
     from IPython.core.display import Math
     from sklearn import metrics
     import math
```

- ### Download the dataset and understand the format

```
# Reading The Dadasets
train = pd.read_csv('kddcup.data.gz', header=None)
test = pd.read_csv('corrected.gz', header=None)
print('===========================Train Dataset===================================')
print(train.head())
print(train.shape)
print('============================Test Dataset===================================')
print(test.head())
print(test.shape)
```

```
===========================Train Dataset====================================
   0    1     2   3    4      5  6   7   8   9   ...  32   33   34    35  \
0  0  tcp  http  SF  215  45076  0   0   0   0   ...   0  0.0  0.0  0.00
1  0  tcp  http  SF  162   4528  0   0   0   0   ...   1  1.0  0.0  1.00
2  0  tcp  http  SF  236   1228  0   0   0   0   ...   2  1.0  0.0  0.50
3  0  tcp  http  SF  233   2032  0   0   0   0   ...   3  1.0  0.0  0.33
4  0  tcp  http  SF  239    486  0   0   0   0   ...   4  1.0  0.0  0.25

    36   37   38   39   40       41
0  0.0  0.0  0.0  0.0  0.0  normal.
1  0.0  0.0  0.0  0.0  0.0  normal.
2  0.0  0.0  0.0  0.0  0.0  normal.
3  0.0  0.0  0.0  0.0  0.0  normal.
4  0.0  0.0  0.0  0.0  0.0  normal.

[5 rows x 42 columns]
(4898431, 42)

===========================Test Dataset===================================
   0    1        2   3    4    5  6   7   8   9   ...  32   33    34    35  \
0  0  udp  private  SF  105  146  0   0   0   0   ...  254  1.0  0.01  0.00
1  0  udp  private  SF  105  146  0   0   0   0   ...  254  1.0  0.01  0.00
2  0  udp  private  SF  105  146  0   0   0   0   ...  254  1.0  0.01  0.00
3  0  udp  private  SF  105  146  0   0   0   0   ...  254  1.0  0.01  0.00
4  0  udp  private  SF  105  146  0   0   0   0   ...  254  1.0  0.01  0.01

    36   37   38   39   40              41
0  0.0  0.0  0.0  0.0  0.0         normal.
1  0.0  0.0  0.0  0.0  0.0         normal.
2  0.0  0.0  0.0  0.0  0.0         normal.
3  0.0  0.0  0.0  0.0  0.0  snmpgetattack.
4  0.0  0.0  0.0  0.0  0.0  snmpgetattack.

[5 rows x 42 columns]
(311029, 42)
```

```python
# Changing the Categorical Data into Numerical Data
cat_features = [1, 2, 3, 41]
for column in cat_features:
    train[column] = pd.Categorical(train[column])
    train[column] = train[column].cat.codes
train = train.astype(np.float32)

true_labels=train[41]
train=train.drop(41 , axis=1)
print(train)
```

```
           0    1     2    3      4        5    6    7    8    9  ...   31  \
0        0.0  1.0  24.0  9.0  215.0  45076.0  0.0  0.0  0.0  0.0  ...  0.0
1        0.0  1.0  24.0  9.0  162.0   4528.0  0.0  0.0  0.0  0.0  ...  1.0
2        0.0  1.0  24.0  9.0  236.0   1228.0  0.0  0.0  0.0  0.0  ...  2.0
3        0.0  1.0  24.0  9.0  233.0   2032.0  0.0  0.0  0.0  0.0  ...  3.0
4        0.0  1.0  24.0  9.0  239.0    486.0  0.0  0.0  0.0  0.0  ...  4.0
...      ...  ...   ...  ...    ...      ...  ...  ...  ...  ...  ...  ...
4898426  0.0  1.0  24.0  9.0  212.0   2288.0  0.0  0.0  0.0  0.0  ...  3.0
4898427  0.0  1.0  24.0  9.0  219.0    236.0  0.0  0.0  0.0  0.0  ...  4.0
4898428  0.0  1.0  24.0  9.0  218.0   3610.0  0.0  0.0  0.0  0.0  ...  5.0
4898429  0.0  1.0  24.0  9.0  219.0   1234.0  0.0  0.0  0.0  0.0  ...  6.0
4898430  0.0  1.0  24.0  9.0  219.0   1098.0  0.0  0.0  0.0  0.0  ...  7.0

            32   33   34    35    36   37    38   39   40
0          0.0  0.0  0.0  0.00  0.00  0.0  0.00  0.0  0.0
1          1.0  1.0  0.0  1.00  0.00  0.0  0.00  0.0  0.0
2          2.0  1.0  0.0  0.50  0.00  0.0  0.00  0.0  0.0
3          3.0  1.0  0.0  0.33  0.00  0.0  0.00  0.0  0.0
4          4.0  1.0  0.0  0.25  0.00  0.0  0.00  0.0  0.0
...        ...  ...  ...   ...   ...  ...   ...  ...  ...
4898426  255.0  1.0  0.0  0.33  0.05  0.0  0.01  0.0  0.0
4898427  255.0  1.0  0.0  0.25  0.05  0.0  0.01  0.0  0.0
4898428  255.0  1.0  0.0  0.20  0.05  0.0  0.01  0.0  0.0
4898429  255.0  1.0  0.0  0.17  0.05  0.0  0.01  0.0  0.0
4898430  255.0  1.0  0.0  0.14  0.05  0.0  0.01  0.0  0.0

[4898431 rows x 41 columns]
```

```
[ ]  cat_features = [1, 2, 3, 41]
     for column in cat_features:
         test[column] = pd.Categorical(test[column])
         test[column] = test[column].cat.codes
     test = test.astype(np.float32)

     true_labels_test=test[41]
     test=test.drop(41 , axis=1)
     print(test)
```

```
          0    1     2    3      4      5    6    7    8    9  ...     31  \
0       0.0  2.0  46.0  9.0  105.0  146.0  0.0  0.0  0.0  0.0  ...  255.0
1       0.0  2.0  46.0  9.0  105.0  146.0  0.0  0.0  0.0  0.0  ...  255.0
2       0.0  2.0  46.0  9.0  105.0  146.0  0.0  0.0  0.0  0.0  ...  255.0
3       0.0  2.0  46.0  9.0  105.0  146.0  0.0  0.0  0.0  0.0  ...  255.0
4       0.0  2.0  46.0  9.0  105.0  146.0  0.0  0.0  0.0  0.0  ...  255.0
...     ...  ...   ...  ...    ...    ...  ...  ...  ...  ...  ...    ...
311024  0.0  2.0  46.0  9.0  105.0  147.0  0.0  0.0  0.0  0.0  ...  255.0
311025  0.0  2.0  46.0  9.0  105.0  147.0  0.0  0.0  0.0  0.0  ...  255.0
311026  0.0  2.0  46.0  9.0  105.0  147.0  0.0  0.0  0.0  0.0  ...  255.0
311027  0.0  2.0  46.0  9.0  105.0  147.0  0.0  0.0  0.0  0.0  ...  255.0
311028  0.0  2.0  46.0  9.0  105.0  147.0  0.0  0.0  0.0  0.0  ...  255.0

           32   33    34    35   36   37   38   39   40
0       254.0  1.0  0.01  0.00  0.0  0.0  0.0  0.0  0.0
1       254.0  1.0  0.01  0.00  0.0  0.0  0.0  0.0  0.0
2       254.0  1.0  0.01  0.00  0.0  0.0  0.0  0.0  0.0
3       254.0  1.0  0.01  0.00  0.0  0.0  0.0  0.0  0.0
4       254.0  1.0  0.01  0.01  0.0  0.0  0.0  0.0  0.0
...       ...  ...   ...   ...  ...  ...  ...  ...  ...
311024  255.0  1.0  0.00  0.01  0.0  0.0  0.0  0.0  0.0
311025  255.0  1.0  0.00  0.01  0.0  0.0  0.0  0.0  0.0
311026  255.0  1.0  0.00  0.01  0.0  0.0  0.0  0.0  0.0
311027  255.0  1.0  0.00  0.01  0.0  0.0  0.0  0.0  0.0
311028  255.0  1.0  0.00  0.01  0.0  0.0  0.0  0.0  0.0
```

Clustering Using K-Means

```
[ ]  def kmeans(X, K, max_iterations = 100):
         centroids = X[random.sample(range(X.shape[0]), K)]
         for i in range(max_iterations):
             distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)
             labels = np.argmin(distances, axis=1)
             new_centroids = np.array([X[labels == k].mean(axis=0) for k in range(K)])
             empty_centroids = np.where(np.isnan(new_centroids).any(axis=1))[0]
             if empty_centroids.size > 0:
                 for c in empty_centroids:
                     new_centroids[c] = X[random.randint(0, X.shape[0]-1)]
             if np.allclose(centroids, new_centroids):
                 break
             centroids = new_centroids
         return labels, centroids
```

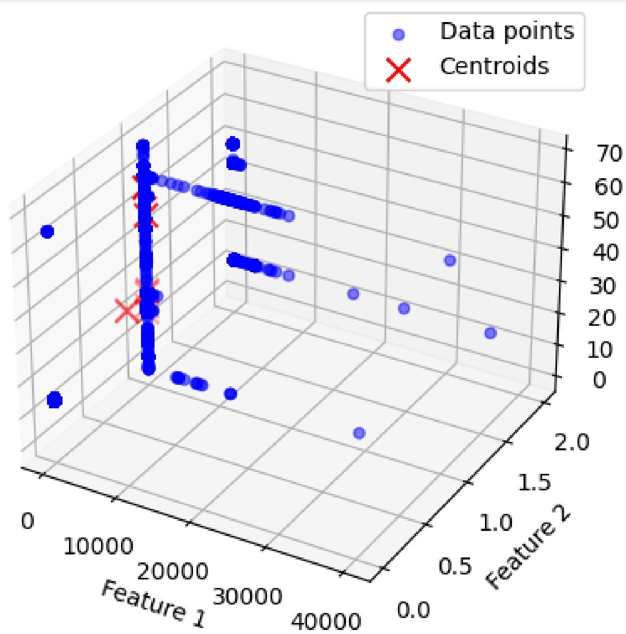
The algorithm initializes K centroids by randomly selecting K data points from the input array X. It then iteratively updates the centroids until convergence. In each iteration, the distances between each data point and the K centroids are computed, and the data points are assigned to their closest centroid. The mean of the data points assigned to each centroid is then computed to update the centroid's position.
If any of the new centroids are empty (i.e., no data points were assigned to them), it is replaced by a random data point from the input array X.
The algorithm stops when either the maximum number of iterations is reached or when the centroids no longer move (i.e., when they converge).

Finally, the function returns two values:
labels: a numpy array of shape (n_samples,) containing the label (i.e., cluster assignment) of each data point.
centroids: a numpy array of shape (K, n_features) containing the final positions of the K centroids.

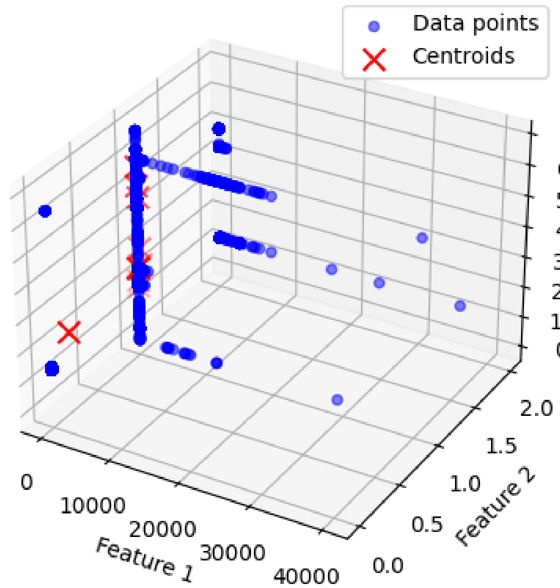
```
[7]  # Kmeans Evaluation
     def kmeans_evaluation(test_data, centroids, true_labels):
       test_k = pd.concat([test_data], ignore_index=True).values
       distances = np.linalg.norm(test_k[:, np.newaxis] - centroids, axis=2)
       labels = np.argmin(distances, axis=1)
       F_score, Fprec, Frecall = calculate_scores(true_labels, labels)
       entropy_score = conditional_entropy(labels, true_labels)
       print("Precision k-means: ",Fprec)
       print("F1 score k-means: ",F_score)
       print("Recall k-means: ",Frecall)
       print("Conditional Entropy k-means: ",entropy_score)
       print('=================================================================================')
```

This function evaluates the performance of a K-means clustering algorithm on test data. It computes the distance of each test data point to each centroid, assigns each point to the closest centroid (labeling), and computes evaluation metrics based on the assigned labels and true labels. Specifically, it computes the F1 score, precision, and recall, as well as the conditional entropy between the assigned labels and true labels.

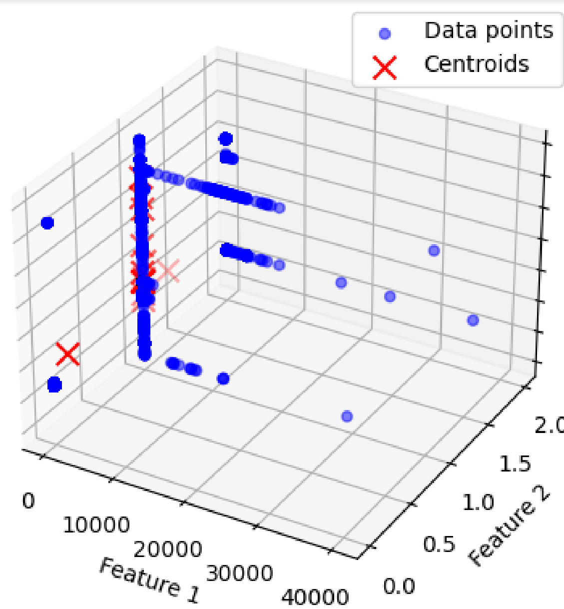
```
k_values = [7, 15, 23, 31, 45]
train_k = train[: int(len(train) * 0.1)]
train_k = pd.concat([train_k], ignore_index=True).values
for k in k_values:
  labels, centroids = kmeans(train_k, k)
  show_result(train_k, centroids, labels, k)
  print(kmeans_evaluation(test, centroids, true_labels_test))
```



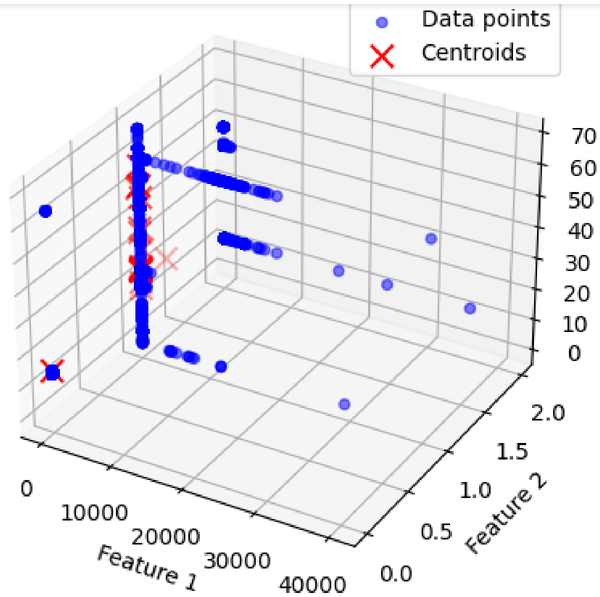

```
Number of clusters: 7
Cluster labels: [0 1 2 3 4 5 6]
Precision k-means:  8.912693988917078e-08
F1 score k-means:  1.7755091784305892e-07
Recall k-means:  2.572107424066566e-05
Conditional Entropy k-means:  1.1617832162376338e-05
==========================================================================================
```



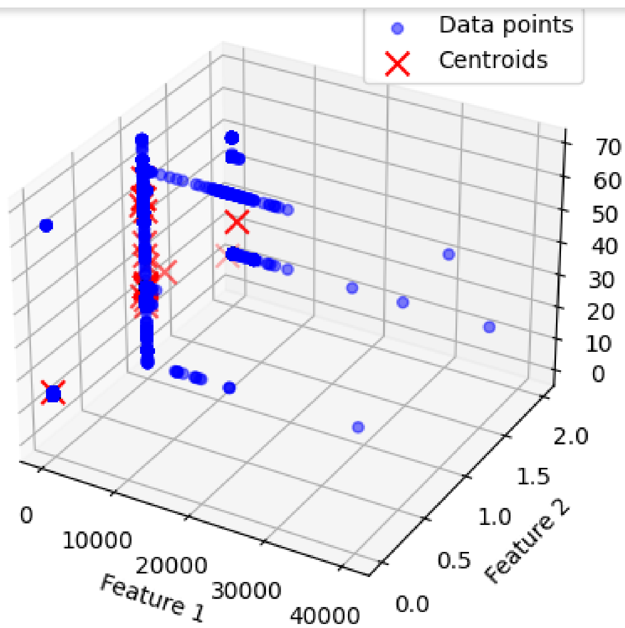

```
Number of clusters: 15
Cluster labels: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
Precision k-means:  0.08927994482376807
F1 score k-means:  0.12074963014213742
Recall k-means:  0.18649064878194638
Conditional Entropy k-means:  2.0603573988349654e-05
====================================================================================
```




```
Number of clusters: 23
Cluster labels: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22]
Precision k-means:  0.19441569691079452
F1 score k-means:  0.020757005658213794
Recall k-means:  0.01099575923788457
Conditional Entropy k-means:  2.5966555803253338e-05
====================================================================================
```




```
Number of clusters: 31
Cluster labels: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30]
Precision k-means:  0.11311638936070181
F1 score k-means:  0.002378534242867889
Recall k-means:  0.003890312478900681
Conditional Entropy k-means:  3.346127131774531e-05
```




```
Number of clusters: 45
Cluster labels: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44]
Precision k-means:  0.17051252138188006
F1 score k-means:  9.357171907593085e-05
Recall k-means:  0.0002990074880477383
Conditional Entropy k-means:  4.544208243140312e-05
------------------------------------------------------------------------------------------
```

- ## Clustering Using Normalized Cut

```
[ ]  def graph_3NN (data):
       D = kneighbors_graph(data, 3, mode='connectivity', include_self=True)
       return D.toarray()

     def Normalize(D):
       rowSums = D.sum(axis=1)
       return D / rowSums[:, np.newaxis];

     def spectralClustering(D,K):
       delta = np.zeros((D.shape[0],D.shape[0]))
       for i in range(D.shape[0]):
         delta[i,i] = np.sum(D[i])
       B = np.identity(D.shape[0]) - np.dot(np.linalg.inv(delta),D)
       U = np.linalg.eigh(B)[1][:,:K]
       Y = Normalize(U);
       imputer = SimpleImputer(strategy='mean')
       Y_imputed = imputer.fit_transform(Y)
       return KMeans(n_clusters=K, random_state=42).fit(Y_imputed).labels_

     def plot(Y):
       plt.figure(figsize = (10, 7))
       ax = plt.axes(projection ="3d")
       ax.scatter3D(Y[:,0],Y[:,1], Y[:,2])
       plt.title("Normalized eign vectors")
       plt.show()

     def ShowResults(result,N):
       print(f'result = {result}')
       for c in range(N):
          print(f'C{c+1} = {train_N[result == c]}')
       return result
```

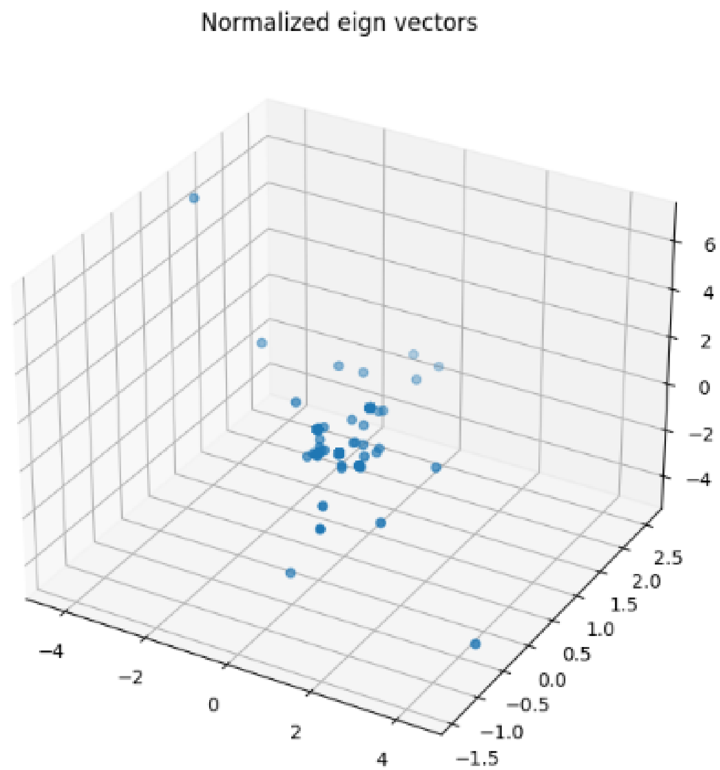
1. In spectral clustering we first compute the similarity graph in the form of adjacency matrix this step can be computed in different ways but we used K-NN (3-NN) here
2. Then we need to project the data onto a lower dimensional space to make the points of the same cluster which are far away from each other closer to be able to cluster them.

   So we need to compute the graph laplacian matrix L then normalize it to reduce the dimensions. The eigenvalues and eigenvectors are computed first then the first k eigenvalues and their eigenvectors are stacked into the matrix such that the eigenvectors are the columns.
3. Finally we cluster the data by using any technique we used K-means here

```
[ ] train_N, test_N = train_test_split(train, test_size=0.9985, train_size=0.0015, random_state=42, shuffle=True, stratify=true_labels)
    labels_normalized=ShowResults(spectralClustering(graph_3NN(train_N),23),23)
```

Splitting the dataset using train_test_split function and taking only 0.0015 of the data. Using a random seed = 42 and number of clusters = 23

```
    return D / rowSums[:, np.newaxis];
```




```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_i
  warnings.warn(
result = [0 0 0 ... 0 0 0]
```

- ## Evaluation

For K-means:

k=7

```
Number of clusters: 7
Cluster labels: [0 1 2 3 4 5 6]
Precision k-means:  8.912693988917078e-08
F1 score k-means:  1.7755091784305892e-07
Recall k-means:  2.572107424066566e-05
Conditional Entropy k-means:  1.1617832162376338e-05
==================================================================
```

k=15

```
Number of clusters: 15
Cluster labels: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
Precision k-means:  0.08927994482376807
F1 score k-means:  0.12074963014213742
Recall k-means:  0.18649064878194638
Conditional Entropy k-means:  2.0603573988349654e-05
==============================================================================
```

k= 23

```
Number of clusters: 23
Cluster labels: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22]
Precision k-means:  0.19441569691079452
F1 score k-means:  0.020757005658213794
Recall k-means:  0.01099575923788457
Conditional Entropy k-means:  2.5966555803253338e-05
```

k=31

```
Number of clusters: 31
Cluster labels: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30]
Precision k-means:  0.11311638936070181
F1 score k-means:  0.002378534242867889
Recall k-means:  0.003890312478900681
Conditional Entropy k-means:  3.346127131774531e-05
==============================================================================
```

k= 45

```
Number of clusters: 45
Cluster labels: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44]
Precision k-means:  0.17051252138188006
F1 score k-means:  9.357171907593085e-05
Recall k-means:  0.0002990074880477383
Conditional Entropy k-means:  4.544208243140312e-05
==============================================================================
```

For spectral (normalized cut):

k= 23

```
Precision spectural:  100.0 %
F1 score spectural:  0.02721829069134458 %
Recall spectural:  0.013610997686130393 %
Conditional Entropy spectural:  0.0 %
```

We can see that spectral clustering is better than k-means:

Spectral clustering helps us overcome two major problems in clustering: the shape of the cluster and determining the cluster centroid. K-means algorithm generally assumes that the clusters are spherical or round,

Many iterations are required to determine the cluster centroid. In spectral, the clusters do not follow a fixed shape or pattern. Points that are far away but connected belong to the same cluster and the points which are less distant from each other could belong to different clusters if they are not connected. This implies that the algorithm could be effective for data of different shapes and sizes.

• New Clustering Algorithm

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm is chosen to be implemented in the new clustering:

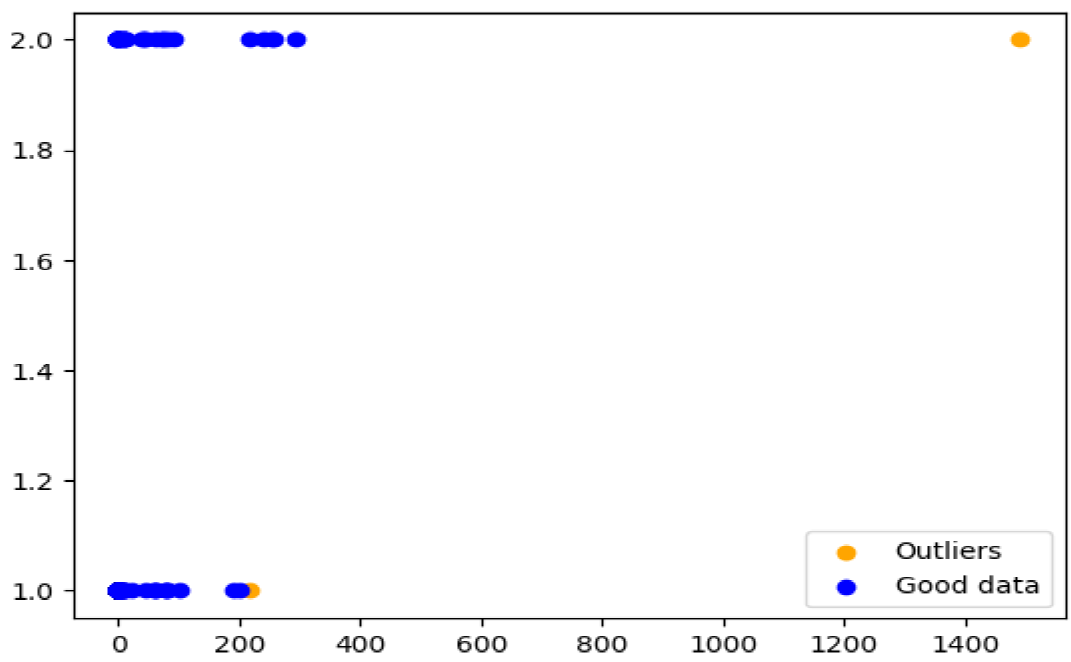
- ☐ It differs from Kmeans and Normalized cut algorithms in some points:
  1. It does not need you to specify the number of clustering unlike k-means and normalized cut.
  2. It can specify the points that were classified as noise or outliers(that do not belong to any cluster).
  3. It is sensitive to the choice of the parameters of epsilon and minPoints as they need to be chosen carefully as it will affect the accuracy of the clustering.

- ☐ It works as followed:
  1. It worked based on two important parameters epsilon(Maximum distance between two points to consider them as neighbors in the same cluster) and minPoints(Minimum number of points to form a cluster)
  2. Loop on the dataset and check if each datapoint is visited or not if not, mark it as visited and go to get its neighbors by calculating the distance between this point and all other points

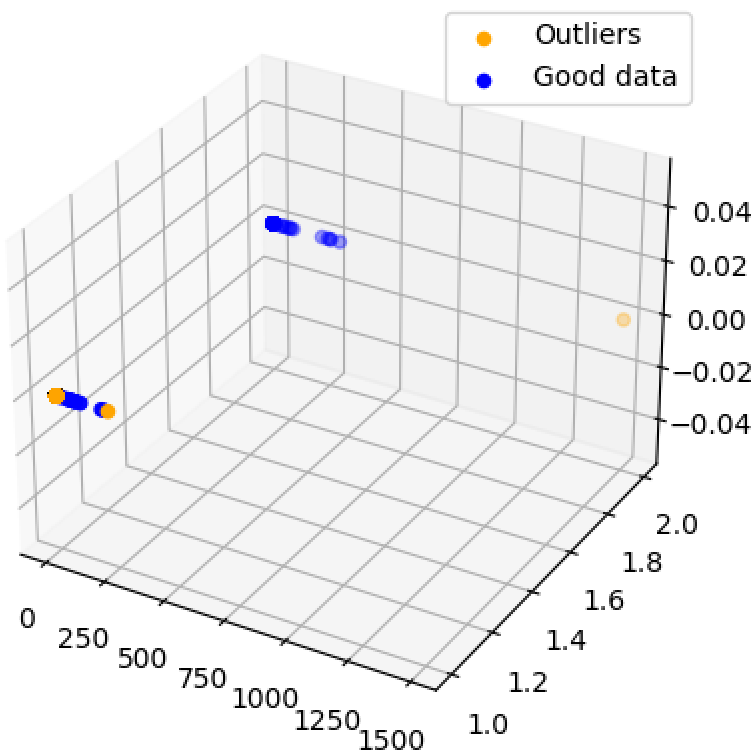
in the dataset and if the distance is less than epsilon, take it as a neighbor to the point. then check if the length of the array of neighbors is larger than or equal to the value of the parameter minPoints or not.

- If yes, put these neighbors in one cluster and give them a number (that initialized to zero and incremented in each iteration) to assign it to the labels in their index. Then expand each neighbor we get as previous steps to get its neighbors.
- If not, mark this point as a noise or outlier by assigning the labels by its index to -1.

☐ After Applying the algorithm of DBSCAN on 0.15% of the data, we get the following clusters:



```
(7347,)
       Cluster
 11.0     1163
 8.0      1098
-1.0       852
 2.0       768
 1.0       434
 5.0       378
 25.0      369
 13.0      348
 7.0       221
 19.0      175
 6.0       157
 10.0      126
 4.0       117
 16.0      116
 26.0       98
 12.0       95
 9.0        90
 3.0        88
 14.0       87
 21.0       83
 15.0       79
 18.0       62
 17.0       42
 20.0       42
 31.0       37
 27.0       36
 28.0       35
 29.0       34
 22.0       28
 32.0       26
 23.0       25
 30.0       15
 24.0       15
 0.0         4
 33.0        3
 34.0        1
```



- The parameters of the DBSCAN algorithm is chosen based on the trial since this algorithm is more sensitive to these parameters by making a range for each of epsilon and min_points then loop on them and evaluate each of both then choose the higher score of them. After that, it is concluded that epsilon=370 and min_points=85.

- After Applying the evaluation to measure its quality, we got the following results:

```
print("F1 Score dbscan = ",F_score*100 , "%")
print("Precision dbscan = ",Fprec*100 , "%" )
print("Recall dbscan = ",Frecall*100 , "%")
print("Conditional Entropy dbscan = ",entropy_score*100 , "%")
```

```
F1 Score dbscan =  27.3325499412456 %
Precision dbscan =  100.0 %
Recall dbscan =  15.82959030896965 %
Conditional Entropy dbscan =  0.0 %
```