Fraud Analytics Assignment 3

Ananthoju Pranav Sai
AI20BTECH11004

Dishank Jain
AI20BTECH11011

Kalvakuntla Umesh
CS20BTECH11024

Nyalapogula Manaswini
CS20BTECH11035

Shivangana Rawat
CS20MTECH12001

1 Problem statement

Identifying outliers in a dataset is an important task in various domains such as finance, healthcare, and marketing. Outliers are data points that deviate significantly from the rest of the data and can have a significant impact on the results of analysis or models built on the data. Traditional methods for outlier detection, such as the Z-score method or the Interquartile range (IQR), are not always effective in detecting outliers that are embedded in complex data structures.

Spectral clustering is a popular clustering technique that uses the spectral properties of the data to group similar data points together. Spectral clustering has been shown to be effective in detecting outliers that are embedded in complex data structures. Spectral clustering works by transforming the data into a high-dimensional space using a similarity matrix, and then clustering the data points in this high-dimensional space.

The problem with spectral clustering is that it requires setting various parameters such as the number of clusters and the similarity threshold. Choosing these parameters correctly is crucial to the success of spectral clustering in identifying outliers. In addition, spectral clustering can be computationally expensive for large datasets.

In this problem, we are given a dataset containing a large number of data points, and the dataset is suspected to have outliers. The objective of this project is to develop an efficient algorithm for identifying outliers using spectral clustering for the given dataset.

	cov1	cov2	cov3	cov4	cov5	cov6	cov7	sal_pur_rat	igst_itc_tot_itc_rat	lib_igst_itc_rat
count	1199.000000	1199.000000	1199.000000	1199.000000	1199.000000	1199.000000	1199.000000	1.199000e+03	1.199000e+03	1.199000e+03
mean	0.956896	0.855770	0.214263	0.147359	0.036329	0.599809	0.527768	-1.251042e-11	-5.004165e-12	1.918265e-11
std	0.135031	0.244927	0.408193	0.388080	0.177615	0.334306	0.385322	1.000000e+00	1.000000e+00	1.000000e+00
min	-0.312219	-0.531958	-0.818128	-0.839158	-0.719622	-0.682734	-0.859529	-3.531330e-02	-1.066436e+00	-5.444774e-02
25%	0.982505	0.840675	-0.095193	-0.143054	0.000000	0.382479	0.245701	-3.284146e-02	-8.884636e-01	-5.424427e-02
50%	0.999235	0.969806	0.175910	0.097584	0.000000	0.691423	0.595623	-3.254101e-02	-3.457085e-01	-5.382146e-02
75%	0.999993	0.996604	0.563061	0.457633	0.000000	0.873218	0.869592	-3.194269e-02	7.059485e-01	-5.191380e-02
max	1.000000	1.000000	1.000000	0.979015	0.999196	0.999999	1.000000	3.436719e+01	2.177948e+00	3.318828e+01

Figure 1: Data Description

2 Description of the dataset

The given data set contains 1199 observations and 10 variables namely cov1, cov2, cov3, cov4, cov5, cov6, cov7, sal_pur_rat, igst_itc_tot_itc_rat, lib_igst_itc_rat.The detailed description of data is present in figure 1.
The variables cov1 to cov7 represent some sort of covariate or input data, while sal_pur_rat, igst_itc_tot_itc_rat, and lib_igst_itc_rat represent the target or response variables.
The summary statistics of the data suggest that the mean of cov1 to cov7 is greater than 0.5, indicating that these variables tend to have higher values. The standard deviation of these variables is relatively small, suggesting that the values tend to cluster around the mean.
The target variables, sal_pur_rat, igst_itc_tot_itc_rat, and lib_igst_itc_rat, have very small mean values, indicating that these variables are low in value. The standard deviation of these variables is 1, suggesting that the values are dispersed around the mean. The minimum and maximum values of these variables also suggest the presence of outliers in the data.

3 Algorithm

The following are the step followed:

1. Calculate Gaussian similarity matrix.
2. Calculate Laplacian Matrix.
3. Calculate eigen vectors and eigen values.
4. Find optimal number of clusters required for the data.
5. Apply k-means spectral clustering to form k clusters.
6. Identifying outliers.

3.1 Calculate Gaussian similarity matrix

A Gaussian similarity matrix is a matrix that encodes the pairwise similarities between data points using a Gaussian kernel function. Given a set of n data points $x_1, x_2, ..., x_n$, the Gaussian similarity matrix W is defined as:

$$W_{ij} = \exp\left(-||x_i - x_j||^2/(2 * \sigma^2)\right) \tag{1}$$

where $||x_i - x_j||$ is the Euclidean distance between data points x_i and x_j, and σ is a parameter that controls the width of the Gaussian kernel. The code is as follows:

```
#calculates gaussian similaruty matrix
def gaussian_similarity_matrix(X, sigma):
    distance_matrix = np.zeros((X.shape[0], X.shape[0]))
    for i in range(X.shape[0]):
        for j in range(X.shape[0]):
            distance_matrix[i,j] = np.linalg.norm(X[i] - X[j])
    similarity_matrix = np.exp(-distance_matrix**2 / (2*sigma**2))
    return similarity_matrix
```

3.2 Calculate Laplacian Matrix.

In spectral clustering, the Laplacian matrix is used to extract the eigenvectors and eigenvalues.The eigenvectors of the Laplacian matrix capture the connectivity structure of the data points and can help reveal underlying patterns in the data.
Laplacian matrix L is defined as $L = D - W$, where W is the similarity matrix and D is the degree matrix, which is a diagonal matrix where D_{ii} is the sum of the weights of the edges incident to vertex i. The code is as follows:

```
#calculates laplacian matrix
def laplacian_matrix(W):
    D = np.diag(np.sum(W, axis=1))
    L = D - W
    return L
```

3.3 Calculate eigen vectors and eigen values.

Eigen values are then used to embed the data points in a lower-dimensional space where they can be clustered using k-means or other clustering algorithms. To compute the eigenvectors and eigenvalues of the Laplacian matrix, we used linalg.eig function from numpy. After that, we sorted eigenvalues of the Laplacian matrix in ascending order. Then, we sorted the eigen vectors according to the order of eigenvalues. The code is as follows:

```
def calculate_eigen_val_and_vec(L):
    #calculates eigen values and eigen vectors of given laplacian matrix
```

```
    eigvals, eigvecs = np.linalg.eig(L)
    #sort eigenvalues in ascending order
    sorted_indices = np.argsort(eigvals)
    sorted_eigenval=eigvals[sorted_indices]
    #sort eigen vectors according to sorted eigen values
    sorted_eigenvec = eigvecs[:,sorted_indices]
    return sorted_eigenval,sorted_eigenvec,sorted_indices
```

3.4 Find optimal number of clusters required for the data.

We can use any of the below method to determine optimal number of clusters.

3.4.1 Elbow method

The elbow method is a heuristic approach used to determine the optimal number of clusters in a dataset when performing clustering analysis. The idea behind the elbow method is to plot the relationship between the number of clusters and the within-cluster sum of squares (WCSS), also known as the distortion or inertia.
The WCSS measures the distance between each data point and its cluster centroid, and the goal of clustering is to minimize this distance. Therefore, the WCSS is a measure of the quality of the clustering, and a lower value indicates better clustering performance. The code is as follows:

```
def clusters_elbow_method(eigen_vectors):
    squared_distance_sum = []
    K = range(2,11)
    for clusters in K :
        # fit KMeans model to the data
        kmeans = KMeans(n_clusters=clusters)
        evec=eigen_vectors[:, 1:clusters]
        kmeans.fit(evec)
        # calculate the sum of squared distance for the current clustering
        squared_distance_sum.append(kmeans.inertia_)
        print('#clusters=',clusters,' sum of squared distance=',squared_distance_sum[-1])
    plt.plot(K,squared_distance_sum)
    plt.xlabel('Values of K')
    plt.ylabel('Sum of squared distances')
    plt.title('Elbow Method For Optimal K')
    plt.show()
```

3.4.2 Silhouette method

The silhouette method is a heuristic approach used to determine the optimal number of clusters in a dataset when performing clustering analysis. The idea behind the silhouette method is to measure the quality of the clustering by computing a silhouette score for each data point.

The silhouette score measures how similar a data point is to its own cluster compared to other clusters. A high silhouette score indicates that a data point is well-matched to its own cluster, while a low silhouette score indicates that the data point may be better suited to a different cluster. The code is as follows:

```
from sklearn.metrics import silhouette_score

def clusters_Silhouette_method(eigen_vectors):
   K = range(2, 11)
   silhouette_scores = []
   for clusters in K:
    # fit KMeans model to the data
    kmeans = KMeans(n_clusters=clusters)
    evec=eigen_vectors[:, 1:clusters]
    kmeans.fit(evec)
    cluster_labels = kmeans.labels_
    # calculate the silhouette score for the current clustering
    silhouette_avg = silhouette_score(evec, cluster_labels)
    # append the silhouette score to the list
    silhouette_scores.append(silhouette_avg)
    print('#clusters=',clusters,' Silhouette score=',silhouette_scores[-1])


   # plot the silhouette scores against the number of clusters
   plt.plot(K, silhouette_scores)
   plt.xlabel('Number of clusters')
   plt.ylabel('Silhouette score')
   plt.title('Silhouette Method For Optimal K')
   plt.show()
```

3.5 Apply k-means spectral clustering to form k clusters.

We applied k-means spectral clustering using inbuilt cluster.KMeans function in sklearn. We then calculated a number of points in each cluster. The code is as follows:

```
def clustering(eigen_vectors,clusters):
        #kmeans clustering
        evec=eigen_vectors[:, 1:clusters]
        kmeans = KMeans(n_clusters=clusters).fit(evec)
        #labels
        colours=kmeans.labels_
        cluster_classes =  [ 0 for eigen_vectors in range(clusters)]
        #count number of points in each cluster
        for i in range(len(colours)):
             cluster_classes[colours[i]] +=1
        #print cluster number and number of points in it
```

```
        for i in range(clusters):
            print(f"Cluster {i}: {cluster_classes[i]}")

        col=['blue','green','red','cyan','magenta','orange','brown',
        'olive','gray','steelblue','black','orchid','indigo','chocolate']

        for i in range(clusters):
             st='cluster'+ str(i)
             plt.scatter(evec[colours == i, 0], evec[colours == i, 1],
             s = 100, c = col[i], label = 'Cluster'+str(i))
        #plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
        #s = 80, c = 'yellow', label = 'Centroid')
        plt.title('Clusters')
        plt.legend()
        plt.show()
        return colours
```

3.6 Identifying outliers

After calculating number of points in each cluster, we identified clusters with relatively less number of points as outliers. We found the rows corresponding to the outliers in the given data.

4 Results

4.1 Optimal number of clusters

Elbow method:

The graph in figure 2 refers to the plot of sum of squared distances vs number of clusters calculated in elbow method to find the optimal number of clusters for given data set.The table in Table 1 contains output obtained (number of clusters and its corresponding to sum of squared distance).
From the table 1 and graph 2, we can clearly observe that when the Number of clusters=3, we get least value of sum of squared distance. This implies that optimal Number of clusters for the given data set =3 according to Elbow method.

Silhouette method:

The graph in figure 3 refers to the plot of Silhouette score vs number of clusters calculated in Silhouette method to find the optimal number of clusters for given data set.The table in Table 2 contains output obtained (number of clusters and its corresponding to Silhouette score).
From the table 2 and graph 3, we can clearly observe that when the Number of clusters=3, we get high value of Silhouette score. This implies that the optimal Number of clusters for the given data set can be 3 according to Silhouette method.

No.of Clusters	sum of squared distance
2	5.632968929140402e-33
3	4.558896230907061e-33
4	0.0008360763286446491
5	4.188200234309614e-07
6	8.796337437185944e-08
7	0.0004015952753948029
8	0.045160521925665664
9	0.13492985047229306
10	0.12399251907848273

Table 1: Table corresponding to elbow method results

No.of Clusters	sum of squared distance
2	0.9991659716430359
3	0.9983319432860718
4	0.9973989934143735
5	0.9966123182972415
6	0.9957964322666064
7	0.9752738063681311
8	0.7713267443029114
9	0.7095613943220516
10	0.7067924745613722

Table 2: Table corresponding to silhouette method results

4.2 Outliers

Rows corresponding to outliers in sorted data(sorted according to eigen values):

Rows corresponding to outliers of cluster 1 : [591]
Rows corresponding to outliers of cluster 2 : [202]

Rows corresponding to outliers in original data:

Rows corresponding to outliers of cluster 1 : [360]
Rows corresponding to outliers of cluster 2 : [1086]

Data corresponding to outliers in original data:

Data corresponding to outliers of cluster 1 :
(0.99592505 0.99735545 -0.42788943 -0.43771254 0. 0.82972292
-0.24801104 -0.03261568 -1.06543615 0.39544647)
Data corresponding to outliers of cluster 2 :
(1. 0.98925865 0.40673191 0.41437495 0. 0.39828522
0.99991626 -0.03313532 2.16902174 -0.05438652)

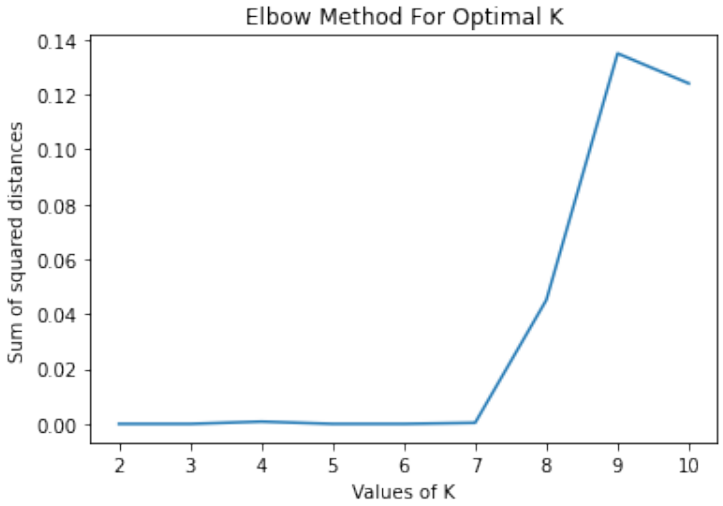

Figure 2: Elbow method graph

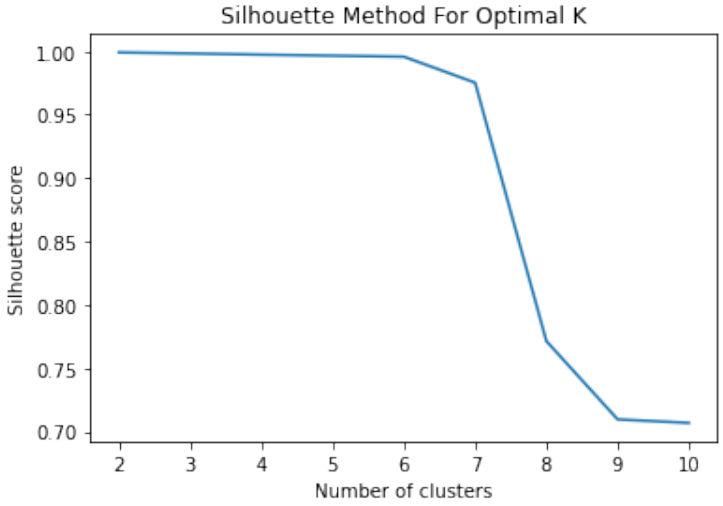

Figure 3: Silhouette method graph