Pattern Recognition Assignment 2

Team Members

Name	ID
Ramy Ahmed ElSayed	19015649
Zyad Samy Ramadan	19015720
Abdelmoniem Hany	19017359

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 help students understand how K-Means and Normalized Cut algorithms can be used for network anomaly detection.

Data Set

The dataset used in the assignment is from the ”KDD Cup 1999” dataset and is partitioned into the following:

- 10% Training Data and its labels.
- Correction Testing Data and its labels.
- Full Data for Spectral Clustering and its labels.

```
def loadData():
 trainingData =  pd.read_csv('drive/MyDrive/Anomaly Detection
Data/kddcup.data_10_percent', sep=",", header = None)
 trainingDataSpectral =  pd.read_csv('drive/MyDrive/Anomaly Detection
Data/kddcup.data/kddcup.data', sep=",", header = None)
 testingData = pd.read_csv('drive/MyDrive/Anomaly Detection Data/corrected/corrected',
sep=",", header = None)
 stringCols = [1, 2, 3, 41]
 for i in stringCols:
```

```
    trainingData[i] = pd.factorize(trainingData[i])[0]
    testingData[i] = pd.factorize(testingData[i])[0]
    trainingDataSpectral[i] = pd.factorize(trainingDataSpectral[i])[0]
  return trainingData.loc[:,0:40], trainingData.loc[:, 41:], testingData.loc[:,0:40],
testingData.loc[:, 41:], trainingDataSpectral
```

K-Means

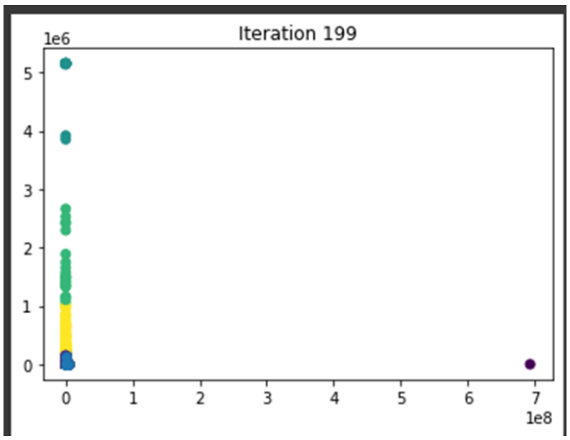
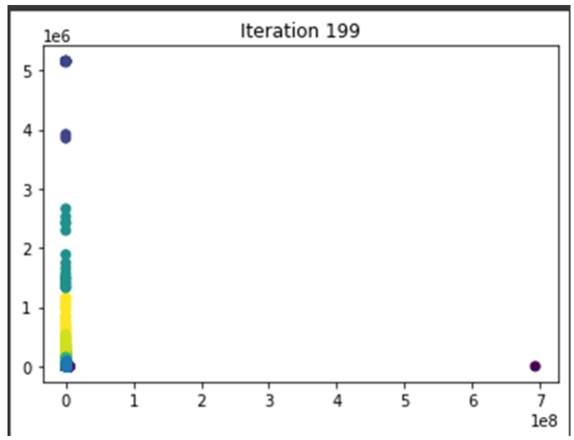
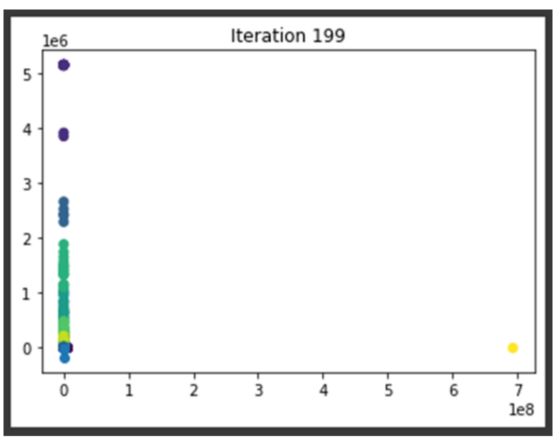
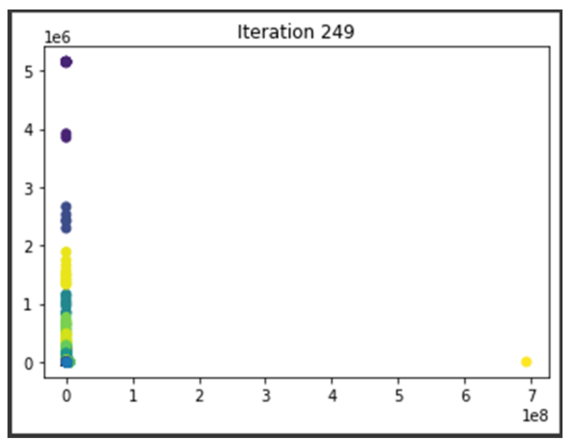
The implementation of K-means is as follows:

1. Select a random subset of the points to be the initial centroids
   by generating a permutation of the numbers from 1:n_rows and then selecting those points from the dataset as the centroids.
2. Iterate until
   a. Convergence: when the delta (change in centroids) becomes less than some given threshold epsilon.
   b. Max iterations are exceeded.
3. During each iteration:
   a. Calculate the proximity matrix (n,k) which tells how far each point is from each centroid.
   b. Select the cluster for each point based on the closest centroid.
   c. Update each cluster centroid.
   d. Calculate the change in centroids to detect convergence.

```
def kMeans(data, k, epsilon, iterations):
 n_rows = data.shape[0]
 random_idx = np.random.RandomState(42).permutation(n_rows)
 centroids = data[random_idx[:k]]
 clusters = np.zeros(n_rows)
 proximityMatrix = np.zeros((n_rows, k))
 delta = np.inf
 iteration = 0

 while delta > epsilon and iteration < iterations:
   for i in range(k):
     proximityMatrix[:,i] = np.linalg.norm(data - centroids[i], axis=1)

   clusters = np.argmin(proximityMatrix, axis = 1)

   iteration += 1
   old_centroids = deepcopy(centroids)

   for i in range(k):
```

```
      centroids[i,:] = np.mean(data[clusters == i,:], axis = 0)
      if np.isnan(centroids).any():
        centroids[i] = old_centroids[i]

    delta = np.linalg.norm(centroids - old_centroids)
   return clusters, centroids
```

Clusters Visualized

K = 7	K = 15
	
K = 23	**K = 31**
	

K-Means Evaluation

The model was trained using the 10% dataset "*kddcup.data_10_percent*" on different values of K = [7, 15, 23, 21, 31, 45] and was then tested on the "*corrected.gz*" dataset and it produced the following measures:

```
actual_labels = testingLabels.to_numpy().flatten()
for k in kArray:
 clusters, centroids = kMeans(trainingData.to_numpy(), k, 0.001, 200)
 predicted_labels = predict(centroids, testingData)
 print(f'K-Means with k={k}')
 checkClustering(predicted_labels, actual_labels, k)
 print()
```

```
K-Means with k=7
Precision Score:  0.35724224700025636
Recall Score:  0.9994639263064989
F Score:  0.5263493980559961
Conditional Entropy:  0.7512333969826389

K-Means with k=15
Precision Score:  0.6311021489567514
Recall Score:  0.6146898689806143
F Score:  0.6227878998581184
Conditional Entropy:  0.3720591818015853

K-Means with k=23
Precision Score:  0.6227486330572504
Recall Score:  0.4462170403306566
F Score:  0.5199064082798301
Conditional Entropy:  0.3981877037228226

K-Means with k=31
Precision Score:  0.7286580436816733
Recall Score:  0.42186554122561254
F Score:  0.5343579636239033
Conditional Entropy:  0.42213098103853636

K-Means with k=45
Precision Score:  0.9218192366460805
Recall Score:  0.36134602791237447
F Score:  0.5191782053575126
Conditional Entropy:  0.24592909396855236
```

Spectral Clustering using K-ways Normalized Cut

```
def SpectralClustering(data, clustersNumber, gammaValue = 1):
   similarityMatrix = rbf(data,gamma=gammaValue)
   degreeMatrix = np.diag(np.sum(similarityMatrix, axis=1))
   laplacianMatrix = degreeMatrix - similarityMatrix
   normAsymLaplacianMatrix = np.dot(sc.linalg.inv(degreeMatrix), laplacianMatrix)
   eigenValues,eigenVectors = sc.linalg.eig(normAsymLaplacianMatrix)
   eigenValues =  np.abs(np.real(eigenValues))
   eigenVectors = np.real(eigenVectors)
   sort_perm = eigenValues.argsort()
   eigenValues = eigenValues[sort_perm]
   eigenVectors = eigenVectors[:, sort_perm]
   eigenValues = eigenValues[:clustersNumber]
   U = eigenVectors[:, :clustersNumber]
   Y = normalize(U, axis=1)
   labels = KMeans(n_clusters=clustersNumber,  n_init="auto",
random_state=42).fit_predict(Y)
   return eigenValues, Y, labels
```

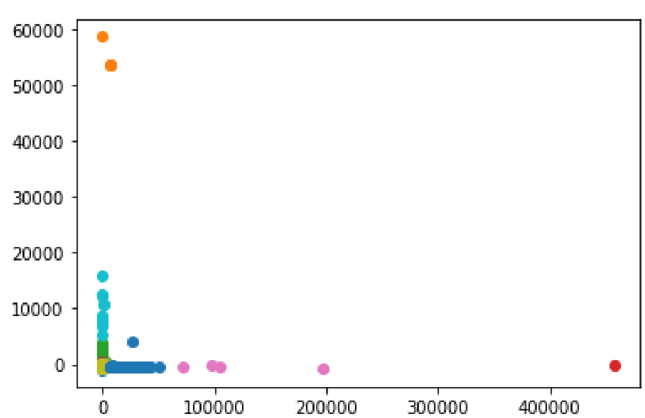
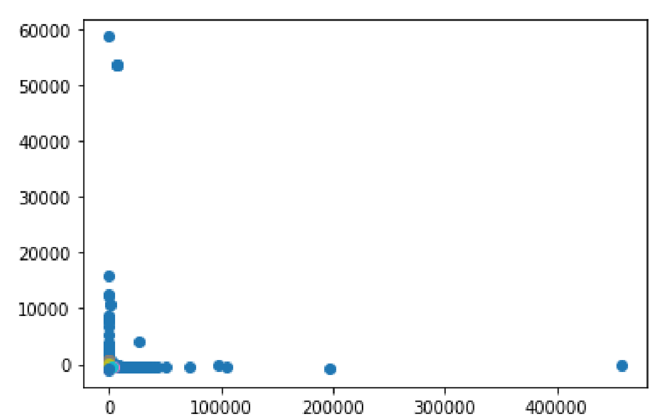
In order to compare spectral clustering with k-means we will run both on k=23, and on a smaller subset of the dataset (0.15%) so that it can run on the available resources.

```
X_train_spectral, _ = train_test_split( totalDataSpectral, train_size=0.0015,
random_state=42)
X_train_meanShift, _ = train_test_split( totalDataSpectral, train_size=0.00015,
random_state=42)


spectral_train = X_train_spectral.loc[:,0:40]
spectral_labels = X_train_spectral.loc[:, 41:]
```

And the results was as follows:

	K-Means	**Spectral Clustering**
Precision	0.8034	0.4210
Recall	0.7236	0.9987
F Score	0.7614	0.5923
Conditional Entropy	0.2934	0.9945

And then PCA was used on the data to apply dimensionality reduction to visualize the results

<table>
<tr><th>K-Means</th><th>Spectral Clustering</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

New Clustering Algorithm (Mean Shift)

```
def mean_shift(data, bandwidth = None, bandwidthStep = 100):

 if bandwidth is None:
   all_data_centroid = np.average(data,axis=0)
   all_data_norm = np.linalg.norm(all_data_centroid)
   bandwidth = all_data_norm/bandwidthStep

 centroids = {}

 for i in range(len(data)):
   centroids[i] = data[i]

 weights = [i for i in range(bandwidthStep)][::-1]
 while True:
   new_centroids = []
   for i in centroids:
     in_bandwidth = []
     centroid = centroids[i]

     for featureset in data:

       distance = np.linalg.norm(featureset-centroid)
       if distance == 0:
         distance = 0.00000000001
       weight_index = int(distance/bandwidth)
       if weight_index > bandwidthStep - 1:
         weight_index = bandwidthStep - 1

       to_add = (weights[weight_index]**2)*[featureset]
       in_bandwidth +=to_add

     new_centroid = np.average(in_bandwidth,axis=0)
     new_centroids.append(tuple(new_centroid))

   uniques = sorted(list(set(new_centroids)))

   to_pop = []

   for i in uniques:
     for j in [i for i in uniques]:
       if i == j:
           pass
       elif np.linalg.norm(np.array(i)-np.array(j)) <= bandwidth:
         to_pop.append(j)
         break
```

```
  for i in to_pop:
    try:
      uniques.remove(i)
    except:
      pass

  prev_centroids = dict(centroids)
  centroids = {}
  for i in range(len(uniques)):
    centroids[i] = np.array(uniques[i])

  optimized = True

  for i in centroids:
    if not np.array_equal(centroids[i], prev_centroids[i]):
      optimized = False

  if optimized:
    break

proximityMatrix = np.zeros((data.shape[0], len(centroids)))
for i in range(len(centroids)):
    proximityMatrix[:,i] = np.linalg.norm(data - centroids[i], axis=1)

clusters = np.argmin(proximityMatrix, axis = 1)

return clusters
```

Mean Shift Evaluation

```
Precision Score:  0.47576078664067134
Recall Score:  0.9584108877922166
F Score:  0.6358713200514358
Conditional Entropy:  0.8557752609213186
```

Evaluation

The following is the code used for the evaluation of the models

```
from sklearn.metrics.cluster import pair_confusion_matrix

def checkClustering(resultingLabels, trueLabels, K):
 confusionMat = pair_confusion_matrix(trueLabels, resultingLabels)
 precision = confusionMat[1][1] / (confusionMat[1][1] + confusionMat[0][1])
 recall = confusionMat[1][1] / (confusionMat[1][1] + confusionMat[1][0])
 f1score = 2 * precision * recall / (precision + recall)
 print("Precision Score: " , precision)
 print("Recall Score: " , recall)
 print("F Score: " , f1score)
 print("Conditional Entropy: " , conditionalEntropy(resultingLabels, trueLabels, K))
```

```
def conditionalEntropy(predictedLabels, trueLabels, K):
 predictedLabeledClusters = [[] for _ in range(K)]
 for i in range(len(predictedLabels)):
     predictedLabeledClusters[predictedLabels[i]].append(trueLabels[i])

 entropy = np.zeros(K)
 N = 0

 for i in range(len(predictedLabeledClusters)):
   N += len(predictedLabeledClusters[i])
   count = np.zeros(K)
   for j in range(len(predictedLabeledClusters)):
     for k in range(len(predictedLabeledClusters[i])):
       if j == predictedLabeledClusters[i][k]:
         count[j] += 1
   for j in range(len(predictedLabeledClusters)):
     if count[j] != 0:
       entropy[i] += (- count[j]/len(predictedLabeledClusters[i])) *
math.log(count[j]/len(predictedLabeledClusters[i]))

 totalEntropy = 0
 for i in range(len(predictedLabeledClusters)):
   totalEntropy += (len(predictedLabeledClusters[i]) / N) * entropy[i]

 return totalEntropy
```