Data Reading and Cleaning

importing data

In [11]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import cv2 as cv
import glob
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder,MinMaxScaler,RobustScaler
from sklearn.metrics.pairwise import rbf_kernel
from sklearn.preprocessing import normalize
from sklearn.model_selection import train_test_split
from sklearn.metrics.cluster import contingency_matrix
from sklearn.cluster import KMeans
import math
from sklearn.metrics.pairwise import euclidean_distances
from scipy.spatial import distance_matrix
from scipy.optimize import linear_sum_assignment
```

In [12]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

In [13]:
```
columns=[
    "duration"
    , "protocol_type",
    "service", "flag",
    "src_bytes",
    "dst_bytes", "land", "wrong_fragment", "urgent",
    "hot", "num_failed_logins", "logged_in",
    "num_compromised", "root_shell",
    "su_attempted", "num_root",
    "num_file_creations", "num_shells",
    "num_access_files", "num_outbound_cmds",
    "is_host_login", "is_guest_login", "count",
    "srv_count", "serror_rate",
    "srv_serror_rate", "rerror_rate",
    "srv_rerror_rate", "same_srv_rate",
    "diff_srv_rate", "srv_diff_host_rate",
    "dst_host_count", "dst_host_srv_count",
    "dst_host_same_srv_rate", "dst_host_diff_srv_rate",
    "dst_host_same_src_port_rate",
    "dst_host_srv_diff_host_rate",
    "dst_host_serror_rate",
    "dst_host_srv_serror_rate", "dst_host_rerror_rate", "dst_host_srv_rerror_rate", "target"]
```

In [14]:
```
def import_data(path, columns, categorical_cols):
    data = pd.read_csv(path,compression = "gzip", names = columns)
    data = data.drop_duplicates(keep = 'first')
    data = data[~data['service'].isin(['red_i', 'urh_i', 'icmp'])]
    dummy_data = pd.get_dummies(data,columns=categorical_cols)
    labels = dummy_data['target']
    dummy_data.drop('target',axis = 1, inplace=True)
    dummy_data.drop('num_outbound_cmds',axis = 1, inplace=True)
    dummy_data.drop('is_host_login',axis = 1, inplace=True)
    return [dummy_data,labels]
```

data cleaning and encoding

In [15]:
```
# One-Hot Encoding for categorical data.
#data, labels = import_data("kddcup.data_10_percent.gz", columns, ['service', 'flag', 'protocol_type'])
path = "/content/drive/MyDrive/pattern/NetworkAnomaly/dataset/kddcup.data_10_percent.gz/kddcup.data_10_percen
path_of_whole_data = "/content/drive/MyDrive/pattern/NetworkAnomaly/dataset/kddcup.data.gz/kddcup.data.gz"
path_test="/content/drive/MyDrive/pattern/NetworkAnomaly/dataset/corrected.gz/corrected.gz"

# importing training set of kmeans
data, labels = import_data(path, columns, ['service', 'flag', 'protocol_type'])

# importing training set of spectral clustering
whole_data, whole_labels = import_data(path_of_whole_data, columns, ['service', 'flag', 'protocol_type'])

#importing test set of kmeans
test_set,ground_truth=import_data(path_test, columns, ['service', 'flag', 'protocol_type'])

# removing rows from test set taht have targets not in the training set
test = pd.concat([test_set, ground_truth],axis = 1)
test = test[test['target'].isin(np.unique(labels).tolist())]
ground_truth = test['target']
test.drop('target', axis = 1, inplace=True)

# label encoding for labels of training and testing set
le = LabelEncoder()
le.fit(labels)
labels = le.transform(np.array(labels))
ground_truth = le.transform(np.array(ground_truth))
```

In [ ]:
```
```

Evaluation Functions

Conditional Entropy

In [2]:
```
def conditional_entropy(contingency_matrix):
    n=np.sum(contingency_matrix)
    n_classes=np.sum(contingency_matrix,axis=1)
    entropy=0
    for i in range(contingency_matrix.shape[0]):
        for j in range(contingency_matrix.shape[1]):
            if contingency_matrix[i][j] != 0:
                h = contingency_matrix[i][j]/np.sum(contingency_matrix[i])
                entropy += (n_classes[i]/n)*(-h*math.log2(h))
    return entropy
```

In [3]:
```
# calculating TP,TN,FP,FN for precison, recall and F1-Score
def confusion_mat(contingency_matrix):
    x=contingency_matrix.shape[0]
    y=contingency_matrix.shape[1]
    TP=0
    for i in range (x):
         for j in range(y):
            TP += math.comb(contingency_matrix[i][j],2)
    FP=0
    for i in range (x):
        s=np.sum(contingency_matrix[i])
        for j in range(y):
            l=contingency_matrix[i][j]*(s-contingency_matrix[i][j])
            FP+=l
    FP=FP//2
    FN=0
    for i in range(x):
        for j in range(y):
            FN+=contingency_matrix[i][j]*np.sum(contingency_matrix[i+1:,j])
    total=math.comb(np.sum(contingency_matrix),2)
    TN=total-(FN+TP+FP)
    conf=[TP,FP,FN,TN]
    return conf
```

Precision

In [4]:
```
def precision(contingency_matrix):
    precision=[]
    for i in range(contingency_matrix.shape[0]):
        class_max=np.max(contingency_matrix[i])
        precision.append(class_max/np.sum(contingency_matrix[i]))
    return precision
```

Recall

In [5]:
```
def recall(contingency_matrix):
    recall=[]
    for i in range(contingency_matrix.shape[0]):
        class_max=np.max(contingency_matrix[i])
        index=np.where(contingency_matrix==class_max)[1]
        recall.append(class_max/np.sum(contingency_matrix[:,index]))
    return recall
```

F1-Score

In [6]:
```
def f1_score(precision,recall,clusters):
  f1=0
  for i in range(clusters):
    f1+=(2*precision[i]*recall[i])/(precision[i]+recall[i])
  return (1/clusters)*(f1)
```

K-means

In [ ]:
```
k_list = [7, 15, 23, 31, 45]
```

In [7]:
```
def euclidean(point, data):
    return np.sqrt(np.sum(np.subtract(point,data)**2, axis=1))
```

Implementation

In [18]:
```
def k_means(data, k):
    new = data.sample(n=k).to_numpy()
    epoch = 0
    while True:
        points = [[] for i in range(k)]
        previous=[]
        cluster=[]
        epoch += 1
        #print(epoch)
        for point in data.values.tolist():
            distances=np.array(euclidean(np.array([np.array(point)]*k),np.array(new)))
            #distances = np.array([np.linalg.norm(point-centroid) for centroid in new])
            centroid_idx = np.argmin(distances)
            cluster.append(centroid_idx)
            points[centroid_idx].append(point)
        previous=new
        new=[np.mean(cluster, axis=0) for cluster in points]
        if(np.array_equal(new, previous)):
            break
    return cluster,new
```

In [9]:
```
def predict(data, centroids):
    cluster=[]
    for point in test.values.tolist():
        distances=np.array(euclidean(np.array(point),centroids))
        centroid_idx = np.argmin(distances)
        cluster.append(centroid_idx)
    return cluster
```

Data Normalization

In [10]:
```
#normalizing data to improve k means performance
def standardization_robust(df, scaler):
    arr = np.array(df)
    arr=scaler.transform(arr)
    return arr

def standardization_minmax(arr, col, minmax_scaler):
    df=pd.DataFrame(minmax_scaler.transform(arr),columns=col)
    return df
```

In [16]:
```
scaler = RobustScaler()
minmax_scaler= MinMaxScaler()
arr = np.array(data)
scaler.fit(arr)
arr = standardization_robust(data,scaler)
minmax_scaler.fit(arr)
data = standardization_minmax(arr, data.columns, minmax_scaler)

arr_test = np.array(test)
arr_test = standardization_robust(test,scaler)
test = standardization_minmax(arr_test, test.columns, minmax_scaler)
```

Clustering Using K-Means

In [ ]:
```
cluster_7,centroids_7 = k_means(data, 7)
```

In [ ]:
```
clusters_7=predict(test, centroids_7)
```

In [ ]:
```
data
```

Out[54]:
```
(145571, 114)
```

In [ ]:
```
##Evaluation when k = 7
cont_7 = metrics.cluster.contingency_matrix(ground_truth,clusters_7)
e_7=conditional_entropy(cont_7.T)
p_7=precision(cont_7.T)
r_7=recall(cont_7.T)
f1_7=f1_score(p_7,r_7,cont_7.shape[1])
print(f"conditional entropy when k is 7 = {e_7}")
print(f"precision when k is 7 = {p_7}")
print(f"recall when k is 7 = {r_7}")
print(f"F1 score when k is 7 = {f1_7}")
```

```
conditional entropy when k is 7 = 0.35894129224731053
precision when k is 7 = [0.9744268077601411, 0.9493218378719378, 0.768194611982308, 1.0, 0.7306805960977109, 0.9490234104446599, 0.6322701688555347]
recall when k is 7 = [0.32608695652173914, 0.3769071820667488, 0.07975204024128071, 0.4380413683705203, 0.09926739162196573, 0.6739130434782609, 0.3918604651162791]
F1 score when k is 7 = 0.4612489954277501
```

In [ ]:
```
cluster_15,centroids_15 = k_means(data, 15)
```

In [ ]:
```
clusters_15=predict(test, centroids_15)
```

In [ ]:
```
##Evaluation when k = 15
cont_15 = metrics.cluster.contingency_matrix(ground_truth,clusters_15)
e_15=conditional_entropy(cont_15.T)
p_15=precision(cont_15.T)
r_15=recall(cont_15.T)
f1_15=f1_score(p_15,r_15,cont_15.shape[1])
print(f"conditional entropy when k is 15= {e_15}")
print(f"precision when k is 15 = {p_15}")
print(f"recall when k is 15 = {r_15}")
print(f"F1 score when k is 15 = {f1_15}")
```

```
conditional entropy when k is 15= 0.2853081783108416
precision when k is 15 = [1.0, 1.0, 0.949210088691796, 0.9995949777237748, 0.9426778687652877, 0.99968622529
02416, 0.8085106382978723, 1.0, 0.49001248439450684, 0.6827133479212254, 0.9953488372093023, 0.9231448763250
883, 0.9293119698397738, 0.7116883116883117, 0.9963518443453587]
recall when k is 15 = [0.04257894846694914, 0.3936674250172194, 0.6737654928192013, 0.12138500885303954, 0.3
700194109912129, 0.06649829892926468, 0.95, 0.15310840055085578, 0.03276909269270105, 1.0, 0.013399845546951
639, 0.05139681290576431, 0.02057982509235875, 0.3186046511627907, 0.05130345849596126]
F1 score when k is 15 = 0.3347449968336772
```

In [ ]:
```
cluster_23,centroids_23 = k_means(data, 23)
```

In [ ]:
```
clusters_23 = predict(test, centroids_23)
```

In [ ]:
```
cont_23 = metrics.cluster.contingency_matrix(ground_truth,clusters_23)
e_23=conditional_entropy(cont_23.T)
p_23=precision(cont_23.T)
r_23=recall(cont_23.T)
f1_23=f1_score(p_23,r_23,cont_23.shape[1])
print(f"conditional entropy when k is 15= {e_23}")
print(f"precision when k is 15 = {p_23}")
print(f"recall when k is 15 = {r_23}")
print(f"F1 score when k is 15 = {f1_23}")
```

```
conditional entropy when k is 15= 0.2535268982825815
precision when k is 15 = [0.9829706717123936, 0.4521072796934866, 0.49588607594936707, 0.9505833905284832, 
1.0, 1.0, 0.9953488372093023, 1.0, 0.8, 0.8631578947368421, 1.0, 0.9147003745318352, 0.964856455946825, 0.68
27133479212254, 0.9824035192961408, 0.9676025917926566, 1.0, 0.9828473413379074, 0.9047619047619048, 0.99935
58776167472, 0.9293119698397738, 0.7116883116883117, 1.0]
recall when k is 15 = [0.05110171158764509, 0.6781609195402298, 0.03270647659201436, 0.028907766483688507, 
0.043497317943687254, 0.26144309240049257, 0.013399845546951639, 0.12064725555774149, 0.0007513932082402788, 
0.09534883720930233, 0.06649829892926468, 0.20389889586942456, 0.6711095809561283, 1.0, 0.10254430089123584, 
0.009350671035879025, 0.15305921699783592, 0.035879025693473315, 0.95, 0.1295318402871992, 0.020579825092358
75, 0.3186046511627907, 0.04257894846694914]
F1 score when k is 15 = 0.26036820368872193
```

In [ ]:
```
cluster_31,centroids_31 = k_means(data, 31)
```

In [ ]:
```
clusters_31 = predict(test, centroids_31)
```

In [ ]:
```
cont_31 = metrics.cluster.contingency_matrix(ground_truth,clusters_31)
e_31=conditional_entropy(cont_31.T)
p_31=precision(cont_31.T)
r_31=recall(cont_31.T)
f1_31=f1_score(p_31,r_31,cont_31.shape[1])
print(f"conditional entropy when k is 45= {e_31}")
print(f"precision when k is 45 = {p_31}")
print(f"recall when k is 45 = {r_31}")
print(f"F1 score when k is 45 = {f1_31}")
```

```
conditional entropy when k is 45= 0.24736919864636803
precision when k is 45 = [0.7063711911357341, 0.9985974754558204, 1.0, 0.9987129987129987, 0.984591679506933
7, 1.0, 0.9504132231404959, 0.5778894472361809, 0.999618320610687, 0.8500196309383589, 0.9402618657937807,
0.9588268471517203, 1.0, 0.8954545454545455, 0.9182608695652174, 1.0, 1.0, 0.6428571428571429, 0.99019745133
73477, 0.3414154652686763, 1.0, 1.0, 1.0, 1.0, 0.8947368421052632, 0.6815245478036176, 0.9991652754590985,
0.9378125400692396, 0.7415730337078652, 1.0, 0.6827133479212254]
recall when k is 45 = [0.29651162790697677, 0.014860887896307738, 0.047167027346055476, 0.01619669804429045
6, 0.2199980326578792, 0.09285854810151485, 0.03360397403519025, 0.6609195402298851, 0.054663855899480286,
0.10648239228801888, 0.023981966563002232, 0.03548245705579094, 0.01903403501868975, 0.02906747983474326, 0.
022040867441714847, 0.06956648786291249, 0.21959466510822148, 0.010356731875719217, 0.34777690340350187, 0.5
199600798403193, 0.023397549623259795, 1.0, 0.07525083612040134, 0.11725908455260796, 0.0003485677964364069,
0.022019995408152616, 0.05887271296478458, 0.30531610694829997, 0.0032461144993114303, 0.022270459810899375,
1.0]
F1 score when k is 45 = 0.21402228442257912
```

In [ ]:
```
cluster_45,centroids_45 = k_means(data, 45)
```

In [20]:
```
clusters_45 = predict(test, centroids_45)
```

In [21]:
```
cont_45 = metrics.cluster.contingency_matrix(ground_truth,clusters_45)
e_45=conditional_entropy(cont_45.T)
p_45=precision(cont_45.T)
r_45=recall(cont_45.T)
f1_45=f1_score(p_45,r_45,cont_45.shape[1])
print(f"conditional entropy when k is 45= {e_45}")
print(f"precision when k is 45 = {p_45}")
print(f"recall when k is 45 = {r_45}")
print(f"F1 score when k is 45 = {f1_45}")
```

```
conditional entropy when k is 45= 0.20687376379107345
precision when k is 45 = [1.0, 0.9796167957602935, 1.0, 0.6798444588464031, 1.0, 1.0, 1.0, 1.0, 0.9215909090
909091, 1.0, 0.6153846153846154, 0.9370932754880694, 0.9388609715242882, 1.0, 0.9926470588235294, 0.98365561
69429097, 0.9715846994535519, 1.0, 0.86, 1.0, 0.6827133479212254, 1.0, 1.0, 1.0, 0.9992896718283847, 0.34571
997345719974, 0.6370510396975425, 1.0, 0.9171240985900334, 0.995835646862854, 0.9382716049382716, 1.0, 0.870
9677419354839, 0.9831908831908832, 0.9984496124031008, 0.9708454810495627, 0.9713541666666666, 0.72093023255
81395, 0.9769784172661871, 1.0, 1.0, 0.9976958525345622, 0.9, 1.0]
recall when k is 45 = [0.09236671257131615, 0.11818807790674798, 0.043664294212185095, 0.021894763206779237,
0.011115482982490656, 0.0905428815929536, 0.01335810147982718, 0.017699484460771013, 0.016927219218968503,
0.023272317421886415, 0.14507772020725387, 0.018033436997766692, 0.023397549623259795, 0.01849262173613575,
0.47093023255813954, 0.21016132205390517, 0.01855523783682244, 0.07589022230965965, 0.00634467833956325, 0.0
3359236671257131, 1.0, 0.05689716349063889, 0.05593704994677631, 0.03531548078729311, 0.34595711194176665,
0.5199600798403193, 0.3918604651162791, 0.017052451420341883, 0.17785059798376157, 0.0748679843877189, 0.95,
0.0474621286641747, 0.0941860465116279, 0.07202938782325562, 0.0134415896140761, 0.034751935881112896, 0.007
785268518711778, 0.7126436781609196, 0.04251633236626245, 0.05813495966948652, 0.008703637995449896, 0.00903
7590532445575, 0.00036559358180156395, 0.06597649809020893]
F1 score when k is 45 = 0.18001787411451572
```

Spectral Clustering

reading and encoding data

In [ ]:
```
whole_data, whole_labels = import_data(path_of_whole_data, columns, ['service', 'flag', 'protocol_type'])
whole_data['labels'] = whole_labels
train_spectral, test_spectral = train_test_split(whole_data, train_size = 0.0025, random_state = 42, stratify
train_spectral_labels = train_spectral['labels']
le = LabelEncoder()
train_spectral_labels_encoded = le.fit_transform(train_spectral_labels)
train_spectral.drop(['labels'], axis = 1, inplace=True)
k = len(np.unique(train_spectral_labels))
train_spectral_labels_unique_alphabetically = sorted(train_spectral_labels.unique())
```

Implementation

In [ ]:
```
def spectral_clustering(sim_matrix, k):
    degree_matrix = np.diag(np.sum(sim_matrix, axis=1))
    L = degree_matrix - sim_matrix
    La = np.dot(np.linalg.inv(degree_matrix), L)
    eigen_vals, eigen_vectors = np.linalg.eigh(La)
    idx = np.argsort(eigen_vals) #Index of arranged eigen values in ascending order
    eigen_vals = eigen_vals[idx]
    eigen_vectors = eigen_vectors[:,idx]
    U = eigen_vectors[:, :k]
    Y = normalize(U)
    #Y_df = pd.DataFrame(Y)
    #kms, centroids = k_means(Y_df, k)
    kms = KMeans(n_clusters=k)
    C = kms.fit_predict(Y)
    return C, Y
```

Clustering using spectral clustering

In [ ]:
```
sim_matrix = rbf_kernel(np.array(train_spectral), gamma=0.1)
C, Y = spectral_clustering(sim_matrix,k)
```

Evaluation

In [ ]:
```
contg_matrix_spectral = contingency_matrix(labels_true = train_spectral_labels, labels_pred = C).T
pd.DataFrame(contg_matrix_spectral)
```

Out[112]:

	0	1	2	3	4	5	6	7	8	9	10
0	2	9	20	4	2009	1	5	10	8	2	2
1	0	0	143	0	0	0	1	0	0	0	0
2	0	0	135	0	0	0	0	0	0	0	0
3	0	0	63	0	1	0	1	1	0	0	0
4	0	0	70	0	2	0	1	0	0	0	0
5	0	0	36	0	6	0	0	0	0	0	0
6	0	0	47	0	3	0	0	0	0	0	0
7	0	0	28	0	8	0	1	1	0	0	0
8	0	0	36	0	3	0	0	1	0	0	0
9	0	0	14	0	0	0	0	0	0	0	0
10	0	0	13	0	0	0	0	0	0	0	0

In [ ]:
```
x=precision(contg_matrix_spectral)
y=recall(contg_matrix_spectral)
f1=f1_score(x,y,contg_matrix_spectral.shape[0])
print("Precision for each Cluster: " + str(x))
print("Recall for each Cluster: " +str(y))
print("Average F1 score across all clusters:" + str(f1))
```

```
Precision for each Cluster: [0.9695945945945946, 0.9930555555555556, 1.0, 0.9545454545454546, 0.958904109589
041, 0.8571428571428571, 0.94, 0.7368421052631579, 0.9, 1.0, 1.0]
Recall for each Cluster: [0.9886811023622047, 0.23636363636363636, 0.2231404958677686, 0.10413223140495868,
0.11570247933884298, 0.02975206611570248, 0.07768595041322314, 0.04628099173553719, 0.02975206611570248, 0.0
23140495867768594, 0.021487603305785124]
Average F1 score across all clusters:0.2320943551061556
```

In [ ]:
```
#num_detected_spectral = sum(contg_matrix_spectral[0])
num_detected_spectral =  sum(sum(contg_matrix_spectral[1:][:]))
```

In [ ]:
```
num_detected_spectral
```

Out[99]:
```
615
```

In [ ]:
```
scaler_2 = RobustScaler()
minmax_scaler_2= MinMaxScaler()
arr = np.array(train_spectral)
scaler_2.fit(arr)
arr = standardization_robust(train_spectral,scaler_2)
minmax_scaler_2.fit(arr)
train_spectral_means = standardization_minmax(arr, train_spectral.columns, minmax_scaler_2)
```

Kmeans on same K of spectral clustering

In [ ]:
```
C_means, centroids_means = k_means(train_spectral_means, 11)
```

In [ ]:
```
contg_matrix_means = contingency_matrix(labels_true = train_spectral_labels, labels_pred = C_means).T
pd.DataFrame(contg_matrix_means)
```

Out[113]:

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	91	0	0	0	0	0	0
1	0	0	0	0	494	0	0	0	0	0	0
2	0	0	529	2	0	0	3	6	0	0	0
3	0	0	0	0	98	0	0	0	0	0	0
4	0	1	76	0	85	0	6	4	0	0	0
5	0	1	0	0	211	1	0	1	1	0	2
6	0	0	0	0	60	0	0	0	0	0	0
7	1	0	0	0	467	0	0	0	0	0	0
8	0	7	0	2	126	0	0	2	7	2	0
9	1	0	0	0	268	0	0	0	0	0	0
10	0	0	0	0	132	0	0	0	0	0	0

In [ ]:
```
num_detected_means = sum(contg_matrix_means[2]) + sum(contg_matrix_means[4])
num_detected_means
```

Out[114]:
```
712
```

In [ ]:
```
entropy_spectral = conditional_entropy(contg_matrix_spectral)
p_s = precision(contg_matrix_spectral)
r_s = recall(contg_matrix_spectral)
f1_s = f1_score(p_s, r_s, 11)
```

In [ ]:
```
print("Conditional entropy = " + str(entropy_spectral))
print("Precision = " + str(p_s))
print("Recall = " + str(r_s))
print("F1 score = " + str(f1_s))
```

```
Conditional entropy = 0.2759862350568628
Precision = [0.9695945945945946, 0.9930555555555556, 1.0, 0.9545454545454546, 0.958904109589041, 0.857142857
1428571, 0.94, 0.7368421052631579, 0.9, 1.0, 1.0]
Recall = [0.9886811023622047, 0.23636363636363636, 0.2231404958677686, 0.10413223140495868, 0.11570247933884
298, 0.02975206611570248, 0.07768595041322314, 0.04628099173553719, 0.02975206611570248, 0.02314049586776859
4, 0.021487603305785124]
F1 score = 0.2320943551061556
```

Agglomerative Clustering

Implementation

In [ ]:
```
def agglomerative(data, k):
    clusters = [[i] for i in range(data.shape[0])] # each point in a separate cluster
    means = data.to_numpy()
    distances = distance_matrix(means, means) # compute distance matrix
    distances[distances == 0] = np.Inf
    while(len(means) > k):
        print(len(means))
        print(len(clusters))

        # getting the minimum distance
        (index1,index2) = np.unravel_index(distances.argmin(), distances.shape)

        # compute the mean of the closest clusters
        mean = [means[index1], means[index2]]
        mean = np.mean(mean, axis = 0)

        # concatenating the clusters
        clusters[index1] = clusters[index1] + clusters[index2]

        #updating the mean vector
        means[index1] = mean

        # deleting the old clusters and their means
        clusters.pop(index2)
        means = np.delete(means, index2, axis = 0)

        #updating the distance matrix
        distances = distance_matrix(means, means)
        distances[distances == 0] = np.Inf
        #print('--------------------')
    return clusters, means
```

Clustering using Agglomerative Clustering

In [ ]:
```
clusters, means = agglomerative(train_spectral, 11)
```

In [ ]:
```
def mapping(data,n):
    map = np.zeros(n)
    for i in range(len(data)):
        for j in range(len(data[i])):
            map[data[i][j]] = i
    return map
```

In [ ]:
```
agg_cluster = mapping(clusters, len(train_spectral))
np.unique(agg_cluster, return_counts = True)
```

Out[121]:
```
(array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.]),
 array([2575,    9,   78,    3,    6,    8,    3,    2,    1,    1,    1]))
```

Evaluation

In [ ]:
```
contg_matrix_agg = contingency_matrix(labels_true = train_spectral_labels, labels_pred = agg_cluster).T
pd.DataFrame(contg_matrix_agg)
```

Out[127]:

	0	1	2	3	4	5	6	7	8	9	10
0	0	9	605	4	1922	1	9	13	8	2	2
1	0	0	0	0	9	0	0	0	0	0	0
2	0	0	0	0	78	0	0	0	0	0	0
3	2	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	6	0	0	0	0	0	0
5	0	0	0	0	8	0	0	0	0	0	0
6	0	0	0	0	3	0	0	0	0	0	0
7	0	0	0	0	2	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0	0	0	0
9	0	0	0	0	1	0	0	0	0	0	0
10	0	0	0	0	1	0	0	0	0	0	0

In [ ]:
```
entropy_agg = conditional_entropy(contg_matrix_agg)
TP_agg, FP_agg, FN_agg, TN_agg = confusion_mat(contg_matrix_agg)
p_agg = precision(contg_matrix_agg)
r_agg = recall(contg_matrix_agg)
f1_agg = f1_score(p_agg, r_agg, 11)
```

In [ ]:
```
print("Conditional entropy = " + str(entropy_agg))
print("Precision = " + str(p_agg))
print("Recall = " + str(r_agg))
print("F1 score = " + str(f1_agg))
```

```
Conditional entropy = 0.9232042032831927
Precision = [0.7464077669902912, 1.0, 1.0, 0.6666666666666666, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Recall = [0.9458661417322834, 0.004390243902439025, 0.038385826771653545, 0.0009813542688910696, 0.002952755
905511811, 0.00392156862745098, 0.0014763779527559055, 0.0009813542688910696, 0.00012301636117603641, 0.0001
2301636117603641, 0.00012301636117603641]
F1 score = 0.08530598749559327
```

Describing agglomerative clustering

Agglomerative clustering is a hierarchical clustering algorithm that begins with each data point as its own cluster and iteratively merges the two closest clusters until a stopping criterion is met. In this algorithm, the distance between clusters is defined as the distance between their closest points (i.e., single linkage), their furthest points (i.e., complete linkage), or the average distance between their points (i.e., average linkage).

At each iteration, the two closest clusters are merged into a single cluster, and the distance matrix between clusters is updated accordingly. This process continues until all points are in a single cluster or until a predefined number of clusters is reached.

One advantage of agglomerative clustering is that it does not require a predefined number of clusters, as the algorithm determines the number of clusters based on the merging process. Additionally, the hierarchical structure of the resulting clusters can provide insight into the relationships between data points.

Disadvantages of agglomeartive clustering

One disadvantage of agglomerative clustering is that it can be computationally expensive for large datasets. Since the algorithm requires the computation of pairwise distances between all data points at each iteration, the time complexity can be O(n^3) or higher for large datasets, where n is the number of data points.

Another potential disadvantage of agglomerative clustering is that it is sensitive to the choice of linkage criterion. Different linkage criteria can result in different cluster structures, and the choice of linkage criterion may depend on the nature of the data and the goals of the analysis. Moreover, agglomerative clustering is a greedy algorithm and once a decision is made to merge two clusters, it cannot be undone, which can lead to suboptimal clustering in some cases.

Finally, agglomerative clustering can also suffer from the "chaining" effect, where small clusters may be merged into larger clusters too quickly, resulting in suboptimal clusters. This effect can be mitigated by using a stopping criterion that takes into account the size and density of the clusters, This is what happened with Neptune and Normal Classes

In [ ]:

```
```