Names	IDs
Hassan Ali Hassan Ibrahim	19015603
Ahmed Hesham Abdel-Razzak	19015378
Abdelrahman Ibrahim Abdelhalim Saad	19015880

Assignment 2
Clustering

Problem Statement

It’s required to implement different cluster algorithms and test them using Kdd_cub data, the required algorithms were:

1. K-means algorithm: with only 10% which contains about 400,000 record with different K values [7, 15, 23, 31, 45] and validate the model using another test data
2. Spectral clustering algorithm (normalized cut): with only 0.15% and with k = 11
3. New algorithm that wasn’t introduced: with 10% of the data and then show how it works and how it differs from the first 2 algorithms

After finishing training and testing the algorithms all the models should be validated using :

1. Precision
2. Recall
3. F1-score
4. Conditional Entropy

Used Cluster algorithm:

1. KNN algorithm

ALGORITHM 13.1. K-means Algorithm

K-MEANS $(\mathbf{D}, k, \epsilon)$:

$t = 0$
Randomly initialize k centroids: $\boldsymbol{\mu}_1^t, \boldsymbol{\mu}_2^t, \ldots, \boldsymbol{\mu}_k^t \in \mathbb{R}^d$
repeat
    $t \leftarrow t + 1$
    $C_j \leftarrow \emptyset$ for all $j = 1, \cdots, k$
    `// Cluster Assignment Step`
    foreach $\mathbf{x}_j \in \mathbf{D}$ **do**
        $j^* \leftarrow \arg\min_i \left\{ \left\| \mathbf{x}_j - \boldsymbol{\mu}_i^{t-1} \right\|^2 \right\}$ `// Assign` $\mathbf{x}_j$ `to closest centroid`
        $C_{j^*} \leftarrow C_{j^*} \cup \{\mathbf{x}_j\}$
    `// Centroid Update Step`
    foreach $i = 1$ *to* k **do**
        $\boldsymbol{\mu}_i^t \leftarrow \frac{1}{|C_i|} \sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j$
until $\sum_{i=1}^k \left\| \boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1} \right\|^2 \leq \epsilon$

2. Spectral cluster

Normalized Cut Algorithm

ALGORITHM 16.1. Spectral Clustering Algorithm

SPECTRAL CLUSTERING $(\mathbf{D}, k)$:

Compute the similarity matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$
if *ratio cut* **then** $\mathbf{B} \leftarrow \mathbf{L}$
else if *normalized cut* **then** $\mathbf{B} \leftarrow \mathbf{L}^s$ or $\mathbf{L}^a$
Solve $\mathbf{B}\mathbf{u}_i = \lambda_i \mathbf{u}_i$ for $i = n, \ldots, n-k+1$, where $\lambda_n \leq \lambda_{n-1} \leq \cdots \leq \lambda_{n-k+1}$
$\mathbf{U} \leftarrow \begin{pmatrix} \mathbf{u}_n & \mathbf{u}_{n-1} & \cdots & \mathbf{u}_{n-k+1} \end{pmatrix}$
$\mathbf{Y} \leftarrow$ normalize rows of $\mathbf{U}$ using Eq. (16.19)
$\mathcal{C} \leftarrow \{C_1, \ldots, C_k\}$ via K-means on $\mathbf{Y}$

3. New algorithm(GMM)

P %*Point Cloud*
K %*Number of Probability Distribution*
π %*Weight of Probability Distribution*
μ %*Mean of Probability Distribution*
Σ %*Covariance of Probability Distribution*
Input : $P = \{p_1, \ldots, p_N\}$, K
Parameter Initialization π, μ, Σ
for $t = 1:T$ %E-step
  for $n = 1:N$
    for $k = 1:K$

$$\gamma(z_{nk}) = \frac{\pi_k N(p_n | \mu_k, \Sigma_k)}{\sum_{i=1}^{K} \pi_i N(p_n | \mu_i, \Sigma_i)};$$

    end
  end
  for $k = 1:K$ %M-step

$$\mu_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) p_n}{\sum_{n=1}^{N} \gamma(z_{nk})};$$

$$\Sigma_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})(p_n - \mu_k)(p_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma(z_{nk})};$$

$$\pi_k = \frac{1}{N} \sum_{n=1}^{N} \gamma(z_{nk});$$

  end
end
Output : $\pi = \{\pi_1, \ldots, \pi_K\}$, $\mu = \{\mu_1, \ldots, \mu_K\}$, $\Sigma = \{\Sigma_1, \ldots, \Sigma_K\}$

Data Used:

1. Knn algorithm:

   The data Used: KDD_cub_10_percent : the data consistent of 494021 record and 42 features(including the target)
   The data contains 4 features (including the target) are Strings which will be decoded before building the model

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 494021 entries, 0 to 494020
Data columns (total 42 columns):
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   duration                     494021 non-null  int64  
 1   protocol_type                494021 non-null  object 
 2   service                      494021 non-null  object 
 3   flag                         494021 non-null  object 
 4   src_bytes                    494021 non-null  int64  
 5   dst_bytes                    494021 non-null  int64  
 6   land                         494021 non-null  int64  
 7   wrong_fragment               494021 non-null  int64  
 8   urgent                       494021 non-null  int64  
 9   hot                          494021 non-null  int64  
 10  num_failed_logins            494021 non-null  int64  
 11  logged_in                    494021 non-null  int64  
 12  num_compromised              494021 non-null  int64  
 13  root_shell                   494021 non-null  int64  
 14  su_attempted                 494021 non-null  int64  
 15  num_root                     494021 non-null  int64  
 16  num_file_creations           494021 non-null  int64  
 17  num_shells                   494021 non-null  int64  
 18  num_access_files             494021 non-null  int64  
 19  num_outbound_cmds            494021 non-null  int64  
 20  is_host_login                494021 non-null  int64  
 21  is_guest_login               494021 non-null  int64  
 22  count                        494021 non-null  int64  
 23  srv_count                    494021 non-null  int64  
 24  serror_rate                  494021 non-null  float64
 25  srv_serror_rate              494021 non-null  float64
 26  rerror_rate                  494021 non-null  float64
 27  srv_rerror_rate              494021 non-null  float64
 28  same_srv_rate                494021 non-null  float64
 29  diff_srv_rate                494021 non-null  float64
 30  srv_diff_host_rate           494021 non-null  float64
 31  dst_host_count               494021 non-null  int64  
 32  dst_host_srv_count           494021 non-null  int64  
 33  dst_host_same_srv_rate       494021 non-null  float64
 34  dst_host_diff_srv_rate       494021 non-null  float64
 35  dst_host_same_src_port_rate  494021 non-null  float64
 36  dst_host_srv_diff_host_rate  494021 non-null  float64
 37  dst_host_serror_rate         494021 non-null  float64
 38  dst_host_srv_serror_rate     494021 non-null  float64
 39  dst_host_rerror_rate         494021 non-null  float64
 40  dst_host_srv_rerror_rate     494021 non-null  float64
 41  target                       494021 non-null  object 
dtypes: float64(15), int64(23), object(4)
memory usage: 158.3+ MB
```

2. Spectral & GMM Algorithm

Specteral clustering

```
[65] np.random.seed(42)
```

```
[66] def prepareData(df):
       train_data_spectral, test_data_spectral = train_test_split(df, train_size = 0.0015, random_state = 42, stratify=df['target'])
       train_data_spectral = encoder(train_data_spectral)
       train_spectral_labels = train_data_spectral['target']
       train_data_spectral = train_data_spectral.drop(['target'], axis=1)

       return train_data_spectral, train_spectral_labels
```

data:

duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_count	dst_host_srv_count	dst_host_same_srv_rate
0	1	19	0	0	0	0	0	0	0	...	1	28	1.00
0	1	36	3	0	0	0	0	0	0	...	255	5	0.02
0	1	19	6	279	296	0	0	0	0	...	255	255	1.00
0	1	19	6	305	502	0	0	0	0	...	15	255	1.00
0	0	12	6	1032	0	0	0	0	0	...	255	255	1.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...
0	0	12	6	520	0	0	0	0	0	...	255	255	1.00
0	0	12	6	1032	0	0	0	0	0	...	255	255	1.00
0	1	19	6	207	2592	0	0	0	0	...	55	255	1.00
0	0	12	6	1032	0	0	0	0	0	...	255	255	1.00
0	1	19	0	0	0	0	0	0	0	...	45	149	1.00

× 41 columns

Labels:

```
labels = pd.DataFrame(train_spectral_labels)
display(labels)
```

	target
1399261	4
649857	2
4822974	4
861609	4
1760842	7
...	...
4321480	7
2541949	7
200003	4
2934061	7
3407579	4

7347 rows × 1 columns

Model:

1. Knn algorithm

```
def KNN_model_train(df, clusters, sigma):

  rows = df.shape[0] #numper of samples ot cluser it
  centroids = []

  #get random values to represents the initial centroids
  for i in range(clusters):
    centroids.append(df.iloc[random.randint(0, rows)])


  error = 100          # the error between old and new centroids
  c = []               # save the cluster after each loop
  l = 0                # number of iteration needd to finish the cluster


  #iterate until the error between the old and new centroids < sigma
  while(error > sigma):
    c = [] #save the cluster of each point

    #calculate the error between the points and each centriod
    for i in range(rows):
      err = []
      for j in range(clusters):
        err.append(euclidean(df.iloc[i], centroids[j]))

      c.append(np.argmin(err)) #add the right cluster to the list

    df['cluster'] = c   # save the new cluster at the data

    new_centroids = df.groupby('cluster').mean() # calculate the new centroids

    #get the error bewteen the old and enw centroids
    for i in range(new_centroids.shape[0]):
      temp = euclidean(centroids[i], new_centroids.iloc[i])
      if(i != 0):
        error = max(temp, error)
      else:
        error = temp

      centroids[i] = new_centroids.iloc[i] #update centroids

    #remove the old clusters to starn next iteration
    df.drop(columns = ['cluster'], inplace = True)

    print(f"l = {l}, Error = {error}")
    l+=1

  #save the last version of clusters
  df['cluster'] = c
  #return the data clustered, and centroids to use them in test
  return df, centroids
```

2. Spectral

```
def NormalizedCut(train_data_spectral):

  # Construct similarity matrix
  W = pairwise_distances(np.array(train_data_spectral), metric="euclidean")
  W = np.exp(-0.5 * W**2)

  # Construct degree  matrix
  D = np.diag(np.sum(np.array(W), axis=1))

  # laplacian matrix
  L = D -  W

  #normalized asymmetric laplacian matrix
  L = np.linalg.inv(D) @ (L)

  #get eigenvalues and vectoes
  e, v = np.linalg.eigh(L)

  #sort eigenvectors corresponding to eigen values
  idx = e.argsort()[::1]
  w = e[idx]
  v = v[:,idx]

  #select eigen vectors required
  U =  getVectors(11, v)
  U = U.T

  # normalize eigenvectors
  Y = np.zeros(np.shape(U))
  for i in range (0, np.shape(U)[0]):
    nor = np.linalg.norm(U[i])
    if (nor == 0):
      Y[i] = U[i]
      continue
    Y[i] = (1 / nor) * U[i]
```

3. GMM

```python
class GaussianMixture:

    def __init__(self, num_clusters, max_iterations=20):

        """Initialize num_clusters(K) and max_iterations for the model"""

        self.num_clusters = num_clusters
        self.max_iterations = max_iterations

    def fit(self, X):

        """Initialize parameters and run E and M step storing log-likelihood value after every iteration"""

        self.pi = np.ones(self.num_clusters)/self.num_clusters
        self.mu = np.random.randint(min(X[:, 0]), max(X[:, 0]), size=(self.num_clusters, len(X[0])))
        self.cov = np.zeros((self.num_clusters, len(X[0]), len(X[0])))

        for n in range(len(self.cov)):
            np.fill_diagonal(self.cov[n], 5)

        # reg_cov is used for numerical stability i.e. to check singularity issues in covariance matrix
        self.reg_cov = 1e-6*np.identity(len(X[0]))

        x,y = np.meshgrid(np.sort(X[:,0]), np.sort(X[:,1]))
        self.XY = np.array([x.flatten(), y.flatten()]).T

        for m, c in zip(self.mu, self.cov):
            c += self.reg_cov
            multi_normal = multivariate_normal(mean=m, cov=c)
        self.log_likelihoods = []
```

```python
for iters in range(self.max_iterations):
    # E-Step

    self.ric = np.zeros((len(X), len(self.mu)))

    for pic, muc, covc, r in zip(self.pi, self.mu, self.cov, range(len(self.ric[0]))):
        covc += self.reg_cov
        mn = multivariate_normal(mean=muc, cov=covc)
        self.ric[:, r] = pic*mn.pdf(X)

    for r in range(len(self.ric)):
        self.ric[r, :] = self.ric[r, :] / np.sum(self.ric[r, :])

    # M-step

    self.mc = np.sum(self.ric, axis=0)
    self.pi = self.mc/np.sum(self.mc)
    self.mu = np.dot(self.ric.T, X) / self.mc.reshape(self.num_clusters,1)

    self.cov = []

    for r in range(len(self.pi)):
        covc = 1/self.mc[r] * (np.dot( (self.ric[:, r].reshape(len(X), 1)*(X-self.mu[r]) ).T, X - self.mu[r]) + self.reg_cov)
        self.cov.append(covc)

    self.cov = np.asarray(self.cov)

    likelihood_sum = np.sum([self.pi[r]*multivariate_normal(self.mu[r], self.cov[r] + self.reg_cov).pdf(X) for r in range(len(self.pi))])
    self.log_likelihoods.append(np.sum(np.log(likelihood_sum)))
```

```
        for m, c in zip(self.mu, self.cov):
            c += self.reg_cov
            multi_normal = multivariate_normal(mean=m, cov=c)


def predict(self, Y):

    """Predicting cluster for new samples in array Y"""

    predictions = []

    for pic, m, c in zip(self.pi, self.mu, self.cov):
        prob = pic*multivariate_normal(mean=m, cov=c).pdf(Y)
        predictions.append([prob])

    predictions = np.asarray(predictions).reshape(len(Y), self.num_clusters)
    predictions = np.argmax(predictions, axis=1)


    colors = ['r', 'b', 'g']

    for m, c, col, i in zip(self.mu, self.cov, colors, range(len(colors))):
        multi_normal = multivariate_normal(mean=m, cov=c)


    return predictions
```

Validation:

1. Knn algorithm

```
[235] def percision(df, c):
        perc = 0
        c_perc = []
        n_t = df.shape[0]

        for i in range(c):
          cluster = df[df['cluster'] == i]
          n = cluster.shape[0]
          max_k = max(cluster.value_counts())
          prc = ((max_k / n)* (n/n_t))
          perc += prc
          c_perc.append(prc)

        return c_perc, perc
```

```
def recall(df, c):
  rec = 0
  c_rec = []
  n_t = df.shape[0]

  for i in range(c):
    cluster = df[df['cluster'] == i]
    cluster_size = cluster.shape[0]
    max_k = max(cluster['target'].value_counts())
    max_label = cluster['target'].value_counts().idxmax()
    total = df[df['target'] == max_label].shape[0]

    recall = max_k / total
    rec += recall * (cluster_size / n_t)
    c_rec.append(recall)

  return c_rec, rec
```

```
[237] def F1_score(percision, recall, c):
        f1 = 0
        for i in range(c):
          perc_i = percision[i]
          rec_i = recall[i]
          f = (2 * perc_i * rec_i) / (perc_i + rec_i)
          f1 += f

        return f1/c
```

```
def Conditional_Entropy(df, c):
  H_t_c = 0
  n_t = df.shape[0]

  for i in range(c):
    cluster = df[df['cluster'] == i]
    n_i = cluster.shape[0]
    labels = cluster['target'].value_counts()
    H_c_i = 0
    for j in labels:
      H_c_i -= (j/n_i)*log(j/n_i)

    H_t_c += (n_i/n_t) * H_c_i

  return H_t_c
```

2. Spectral / New algorithm

```
print("number of detected anomalies = ", (1 - score) * np.size(train_spectral_labels))
```

```
number of detected anomalies =  3298.7130732207265
```

Precision

```
def Precision(labels_pred, train_spectral_labels, K):
  labels_pred = labels_pred.tolist()
  labels_true = np.array(train_spectral_labels)
  co = 0;
  purity = 0
  out_arr = np.argsort(labels_pred)
  cb = 0;

  for j in range(0, K):
    cj = labels_pred.count(j) + cb            #count of datarows in cluster j
    true_labels_in_clusterj = []
    for i in range(cb, cj):                   #store the labels of data in cluster j
      true_labels_in_clusterj.append(labels_true[out_arr[i]])
    cb = cj
    set_true_labels_in_clusterj = [*set(true_labels_in_clusterj)]
    cmr , element = count_max_repeated(set_true_labels_in_clusterj, true_labels_in_clusterj)
    purity += cmr / np.size(labels_true)
    true_labels_in_clusterj = []

  return purity
  #adjusted_rand_score(labels_true, labels_pred)
```

Recall

```
def Recall(labels_pred, train_spectral_labels, K):
  labels_pred = labels_pred.tolist()
  labels_true = np.array(train_spectral_labels)
  co = 0;
  recall = 0
  out_arr = np.argsort(labels_pred)
  cb = 0;
  for j in range(0, K):
    cj = labels_pred.count(j) + cb            #count of datarows in cluster j
    true_labels_in_clusterj = []              #store the true labels of datarows in cluster j
    for i in range(cb, cj):
      true_labels_in_clusterj.append(labels_true[out_arr[i]])
    cb = cj
    set_true_labels_in_clusterj = [*set(true_labels_in_clusterj)]
    cmr, element = count_max_repeated(set_true_labels_in_clusterj, true_labels_in_clusterj)
    recallj = cmr / labels_true.tolist().count(element)
    recall += recallj * (np.size(true_labels_in_clusterj) / np.size(labels_true))
    true_labels_in_clusterj = []

  return recall
  #print("recall = ", recall)
```

F1 score

```
[105] def F1_score(labels_pred, train_spectral_labels):
        labels_pred = labels_pred.tolist()
        labels_true = np.array(train_spectral_labels)
        co = 0;
        f = 0;
        out_arr = np.argsort(labels_pred)
        cb = 0;

        for j in range(0, 11):
          cj = labels_pred.count(j) + cb            #count of datarows in cluster j
          true_labels_in_clusterj = []          #store the true labels of datarows in cluster j
          for i in range(cb, cj):
            true_labels_in_clusterj.append(labels_true[out_arr[i]])
          cb = cj
          set_true_labels_in_clusterj = [*set(true_labels_in_clusterj)]
          cmr, element = count_max_repeated(set_true_labels_in_clusterj, true_labels_in_clusterj)
          pj = cmr / np.size(true_labels_in_clusterj)
          recallj = cmr / labels_true.tolist().count(element)
          fj = (2 * pj * recallj) / (pj + recallj)
          f += fj
          true_labels_in_clusterj = []

        f = f / 11
        return f
```

Conditional Entropy

```
[106] def Conditional_Entropy(labels_pred, train_spectral_labels):
        labels_pred = labels_pred.tolist()
        labels_true = np.array(train_spectral_labels)
        co = 0;
        H = 0
        h = 0
        out_arr = np.argsort(labels_pred)
        cb = 0;
        for j in range(0, 11):
          cj = labels_pred.count(j) + cb        #count of datarows in cluster j
          true_labels_in_clusterj = []          #store the true labels of datarows in cluster j
          for i in range(cb, cj):
            true_labels_in_clusterj.append(labels_true[out_arr[i]])
          cb = cj
          set_true_labels_in_clusterj = [*set(true_labels_in_clusterj)]

          for element in set_true_labels_in_clusterj:
            x1 = true_labels_in_clusterj.count(element)
            y1 = np.size(true_labels_in_clusterj)
            h -= (x1 / y1) * math.log((x1 / y1))
          H += ((np.size(true_labels_in_clusterj) / np.size(labels_true)) * h)
          h = 0
          true_labels_in_clusterj = []

        return H
```

Validation Output:

1) K-means outputs

algorithm	k	Precision	Recall	F1-score	Conditional entropy	Train Iterations
KNN	7	0.71	0.99	0.14	0.75	full(33)
	15	0.76	0.96	0.072	0.66	70
	23	0.81	0.56	0.05	0.58	70
	15	0.74	0.73	0.07	0.67	30
	23	0.86	0.55	0.055	0.44	30
	31	0.90	0.41	0.04	0.3	30
	45	0.92	0.28	0.03	0.22	30
Spectral Clustering	11	0.89	0.55	0.32	0.34	
GMM	11	0.9	0.92	0.39	0.24	

```
[40] F1_score(y_pred, train_spectral_labels, 11)
     #f1_score(test_spectral_labels, y_pred, average='weighted')
```

```
Cluster  0  F1 score =  0.8288659793814432
Cluster  1  F1 score =  0.6666666666666666
Cluster  2  F1 score =  0.0013698630136986301
Cluster  3  F1 score =  0.0013698630136986301
Cluster  4  F1 score =  0.6338951310861423
Cluster  5  F1 score =  0.11620400258231117
Cluster  6  F1 score =  0.7499999999999999
Cluster  7  F1 score =  0.0027378507871321013
Cluster  8  F1 score =  0.9982187388671179
Cluster  9  F1 score =  0.21052631578947367
Cluster  10  F1 score =  0.11010362694300517
F1 score =  0.39272345801188085
```