# Network Anomaly Detection

## Pattern Recognition
## Assignment (2)


Sohayla Khaled Abouzeid

ID: 6851

Lara Hossam Eldine Mostafa

ID: 6853

Mohamed Alaa ElZeftawy

ID: 6886

## Table of Contents

## Problem statement:

With the enormous growth of computer networks usage and the huge increase in the number of applications running on top of it, network security is becoming increasingly more important. Therefore, the role of Intrusion Detec-tion Systems (IDSs), as special-purpose devices to detect anomalies and attacks in the network, is becoming more important.

## 1. Download the Dataset and Understand the Format

1] Preprocess the "kddcup.data_10_percent.gz" dataset used for K-means clustering which is 10% of the original data:

- Use the pandas library to read the data while adding the feature names.
- Visualize the data :

In [77]:
```
# Visualize the data to understand the format
df = pd.read_csv('kddcup.data_10_percent.gz', names = features)
df.head()
```

Out[77]:

| | duration | protocol_type | service | flag | src_bytes | dst_bytes | land | wrong_fragment | urgent | hot | ... | dst_host_srv_count | dst_host_same_srv_rate | dst_ho |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | tcp | http | SF | 181 | 5450 | 0 | 0 | 0 | 0 | ... | 9 | 1.0 | |
| **1** | 0 | tcp | http | SF | 239 | 486 | 0 | 0 | 0 | 0 | ... | 19 | 1.0 | |
| **2** | 0 | tcp | http | SF | 235 | 1337 | 0 | 0 | 0 | 0 | ... | 29 | 1.0 | |
| **3** | 0 | tcp | http | SF | 219 | 1337 | 0 | 0 | 0 | 0 | ... | 39 | 1.0 | |
| **4** | 0 | tcp | http | SF | 217 | 2032 | 0 | 0 | 0 | 0 | ... | 49 | 1.0 | |

5 rows × 42 columns

- Understand the dataset and the labels using the describe() Method .

.

In [78]:
```
df.describe()
```

Out[78]:

| | duration | src_bytes | dst_bytes | land | wrong_fragment | urgent | hot | num_failed_logins | logged_in | num_ |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 494021.000000 | 4.940210e+05 | 4.940210e+05 | 494021.000000 | 494021.000000 | 494021.000000 | 494021.000000 | 494021.000000 | 494021.000000 | 4 |
| **mean** | 47.979302 | 3.025610e+03 | 8.685324e+02 | 0.000045 | 0.006433 | 0.000014 | 0.034519 | 0.000152 | 0.148247 | |
| **std** | 707.746472 | 9.882181e+05 | 3.304000e+04 | 0.006673 | 0.134805 | 0.005510 | 0.782103 | 0.015520 | 0.355345 | |
| **min** | 0.000000 | 0.000000e+00 | 0.000000e+00 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **25%** | 0.000000 | 4.500000e+01 | 0.000000e+00 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **50%** | 0.000000 | 5.200000e+02 | 0.000000e+00 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **75%** | 0.000000 | 1.032000e+03 | 0.000000e+00 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **max** | 58329.000000 | 6.933756e+08 | 5.155468e+06 | 1.000000 | 3.000000 | 3.000000 | 30.000000 | 5.000000 | 1.000000 | |

8 rows × 38 columns

In [79]:
```
print('Number of datapoints: ',df.shape[0])
print('Number of features:',df.shape[1])
print("Features are:",features)
```

```
Number of datapoints:  494021
Number of features: 42
Features are: ['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_rat
e', 'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_c
ount', 'dst_host_srv_count', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate', 'dst_host_sr
v_diff_host_rate', 'dst_host_serror_rate', 'dst_host_srv_serror_rate', 'dst_host_rerror_rate', 'dst_host_srv_rerror_rate',
'label']
```

In [80]:
```
# True class labels
description = df['label'].describe()
description
```

Out[80]:
```
count     494021
unique        23
top       smurf.
freq      280790
Name: label, dtype: object
```

- Check for Null values.
- Check for duplicate values and remove them.
- Check for redundant attributes and remove them
- Convert categorical features to numerical features using one hot encoding.
- Standardize continuous features.
- Encode the categorical labels to classes' numbers(0-22).
- Split data into features and labels.

2] Preprocess the "kddcup.data.gz" dataset used for the Normalized cut algorithm:

- Use the pandas library to read the data while adding the feature names.
- Check for Null values.
- Check for duplicate values and remove them.
- Check for redundant attributes and remove them
- Convert categorical features to numerical features using one hot encoding.
- Standardize continuous features.
- Encode the categorical labels to classes' numbers(0-10).
- Split data into features and labels

## 2. Clustering using K-means (your implementation):

1. Initialize K random centroids.
2. Loop on on points:
    a. Calculate the distance between each point and each one of the K centroids .
    b. Find the minimum distance.
    c. Append the point to this centroid's cluster.
3. Update the K centroids by calculating the mean of each cluster.
4. Repeat the steps 2-3 until there is no update in the assignment of centroids.

## 3. K-means Clustering Evaluation

Evaluation measures are an essential aspect of measuring the effectiveness and accuracy of various machine learning and data mining algorithms. These measures are used to evaluate the performance of these algorithms in various applications such as text classification, image recognition, and natural language processing. One of the most commonly used evaluation measures is the F-1 measure, which is a combination of precision and recall. Precision measures the accuracy of the positive predictions made by a model, while recall measures the ability of the model to correctly identify positive cases.

Another evaluation measure that is commonly used is recall. Recall is a measure of the ability of a model to correctly identify all relevant instances of a class, regardless of whether they are classified as positive or negative. Purity is another evaluation measure that is used

in clustering analysis to evaluate the quality of clustering results. It measures the degree to which the clusters produced by an algorithm contain only members of a single class.

Finally, conditional entropy is an evaluation measure that is used to evaluate the effectiveness of a model in predicting the value of a variable based on the value of another variable. It measures the amount of uncertainty in the predicted variable that remains after taking into account the value of the other variable.

- K = 7

```
#labels = kMeans_implemented(7,np.array(data_k_means))
contingency_matrix = get_contingency(labels1,np.array(labels_kmeans))
evaluation(np.array(data_k_means),contingency_matrix)
```

```
---------Conditional Entropy--------------------
Conditional Entropy: 0.35235793694748563
----------------Purity---------------------------
Per cluster purity: [0.63, 0.93, 0.99, 1.0, 0.98, 0.72, 0.83]
Purity: 0.9272251452749578
--------------Recalls---------------------------
Per cluster Recall: [0.189, 0.0892, 0.3665, 0.8109, 0.362, 0.0454, 0.0839]
--------------F-measure---------------------------
F: 0.37865756371357984
```

- K = 15

```
#labels = kMeans_implemented(15,np.array(data_k_means))
contingency_matrix = get_contingency(labels2,np.array(labels_kmeans))
evaluation(np.array(data_k_means),contingency_matrix)
```

```
---------Conditional Entropy--------------------
Conditional Entropy: 0.19353415547078842
----------------Purity---------------------------
Per cluster purity: [0.97, 0.95, 0.97, 0.98, 0.7, 0.99, 0.98, 0.98, 0.91, 0.97, 1.0, 0.53, 1.0, 0.57, 1.0]
Purity: 0.9672770733449645
--------------Recalls---------------------------
Per cluster Recall: [0.0583, 0.0446, 0.0531, 0.0524, 0.7551, 0.0723, 0.2908, 0.294, 0.189, 0.1071, 0.3826, 0.0045, 0.9913,
0.0203, 0.4283]
--------------F-measure---------------------------
F: 0.324018694422339
```

- K = 23

```
#labels = kMeans_implemented(23,np.array(data_k_means))
contingency_matrix = get_contingency(labels3,np.array(labels_kmeans))
evaluation(data_k_means,contingency_matrix)
```

```
---------Conditional Entropy--------------------
Conditional Entropy: 0.2619780402966085
----------------Purity---------------------------
Per cluster purity: [0.98, 1.0, 1.0, 0.67, 0.93, 0.96, 1.0, 0.98, 0.94, 0.97, 1.0, 1.0, 0.67, 1.0, 1.0, 0.54, 0.95, 0.74, 0.
98, 0.7, 0.98, 0.69, 1.0]
Purity: 0.9368002417814901
--------------Recalls---------------------------
Per cluster Recall: [0.252, 0.0557, 0.2481, 0.189, 0.7859, 0.0411, 0.1737, 0.1007, 0.0203, 0.1515, 0.0676, 0.1846, 0.8356,
0.1782, 0.0567, 0.0042, 0.0052, 0.0348, 0.0222, 0.7582, 0.0259, 0.0252, 0.106]
--------------F-measure---------------------------
F: 0.245611790960462
```

- K = 31

```
#labels = kMeans_implemented(31,np.array(data_k_means))
contingency_matrix = get_contingency(labels4,np.array(labels_kmeans))
evaluation(np.array(data_k_means),contingency_matrix)
```

```
---------Conditional Entropy--------------------
Conditional Entropy: 0.14174278422676723
----------------Purity---------------------------
Per cluster purity: [1.0, 1.0, 1.0, 0.95, 0.99, 0.65, 1.0, 1.0, 1.0, 0.98, 1.0, 0.98, 1.0, 0.9, 0.95, 0.99, 0.96, 0.87, 0.9
5, 1.0, 1.0, 0.99, 1.0, 1.0, 0.97, 0.95, 1.0, 0.69, 0.54, 0.93, 0.71]
Purity: 0.9756638687785916
--------------Recalls---------------------------
Per cluster Recall: [0.0024, 0.1489, 0.1019, 0.0565, 0.0789, 0.75, 0.0295, 0.0458, 0.9913, 0.005, 0.1252, 0.2148, 0.1567, 0.
0042, 0.0174, 0.0289, 0.037, 0.0137, 0.624, 0.0777, 0.0585, 0.0528, 0.0001, 0.1569, 0.189, 0.7826, 0.3062, 0.8356, 0.0042,
0.0624, 0.0253]
--------------F-measure---------------------------
F: 0.24337134966674756
```

- K = 45

```
#labels = kMeans_implemented(45,np.array(data_k_means))
contingency_matrix = get_contingency(labels5,np.array(labels_kmeans))
evaluation(np.array(data_k_means),contingency_matrix)
```

```
---------Conditional Entropy--------------------
Conditional Entropy: 0.1250806575035399
----------------Purity---------------------------
Per cluster purity: [0.96, 0.68, 1.0, 0.95, 1.0, 0.79, 1.0, 0.99, 0.95, 0.9, 1.0, 0.99, 1.0, 0.85, 0.87, 0.98, 0.96, 1.0, 1.
0, 1.0, 0.74, 0.54, 0.96, 1.0, 0.47, 1.0, 1.0, 0.98, 1.0, 1.0, 0.99, 1.0, 1.0, 1.0, 1.0, 1.0, 0.92, 1.0, 0.72, 1.0, 1.0, 0.6
9, 1.0, 0.97, 1.0]
Purity: 0.9765155990273789
--------------Recalls---------------------------
Per cluster Recall: [0.0368, 0.0041, 0.0288, 0.624, 0.0533, 0.8326, 0.7362, 0.226, 0.0348, 0.0042, 0.0859, 0.0234, 0.0422,
0.0049, 0.0136, 0.0151, 0.0497, 0.0987, 0.1554, 0.1554, 0.0151, 0.0042, 0.0969, 0.0921, 0.0004, 0.0562, 0.0244, 0.0127, 0.05
85, 0.0401, 0.0668, 0.0295, 0.0946, 0.005, 0.0135, 0.0316, 0.0047, 0.125, 0.0213, 0.0271, 0.9913, 0.2885, 0.0307, 0.016, 0.0
997]
--------------F-measure---------------------------
F: 0.1657574004027622
```

✏️: notice that the clustering gets better as the number of clusters(k) approaches the number of classes(23).

## 4. Clustering using Normalized Cut (your implementation):

📍:Since the train-test-split only returns 11 classes in the train set, run the algorithm to cluster into 11 clusters.

```
# Explore the classes in training dataset after splitting
for i in y_train.unique():
    print(category_map(i))
```

```
neptune.
normal.
teardrop.
ipsweep.
back.
satan.
smurf.
portsweep.
pod.
nmap.
warezclient.
```

1. Compute the Similarity Matrix using the rbf kernel similarity measure.
2. Compute the Degree Matrix.
3. Compute the Laplacian Matrix(L) .
4. Compute the inverse Laplacian Matrix(La).
5. Compute the eigenValues and eigenVectors of the inverse Laplacian Matrix.
6. Sort the eigenVectors according to the ascending order of their corresponding eigenValues.
7. Slice the eigenVectors to the desired number of clusters(23).
8. Normalize the eigenVectors.
9. Apply K-means on the normalized eigenVectors to cluster.

## 5. Normalized Cut Evaluation:

```
contingency_matrix = get_contingency(labels_spectral,new_y_train)
evaluation(X_train,contingency_matrix)
```

```
---------Conditional Entropy--------------------
Conditional Entropy: 0.5290454824191261
----------------Purity---------------------------
Per cluster purity: [0.75, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.66, 0.6]
Purity: 0.8262002232973576
--------------Recalls---------------------------
Per cluster Recall: [0.0347, 0.2, 0.2, 0.2, 0.2, 0.2, 1.0, 0.0015, 0.7128, 0.0861, 0.8596]
--------------F-measure---------------------------
F: 0.4024869886301618
```

## 6. K-means Vs Spectral Clustering [k=11]:

**K-means evaluation**

```
contingency_matrix = get_contingency(labels_spectral,new_y_train)
evaluation(X_train,contingency_matrix)
```

```
---------Conditional Entropy--------------------
Conditional Entropy: 0.5290454824191261
----------------Purity---------------------------
Per cluster purity: [0.75, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.66, 0.6]
Purity: 0.8262002232973576
--------------Recalls---------------------------
Per cluster Recall: [0.0347, 0.2, 0.2, 0.2, 0.2, 0.2, 1.0, 0.0015, 0.7128, 0.0861, 0.8596]
--------------F-measure---------------------------
F: 0.4024869886301618
```

**Spectral Clustering evaluation**

```
labels6 = kMeans_implemented(11,np.array(X_train))
contingency_matrix = get_contingency(labels6,new_y_train)
evaluation(X_train,contingency_matrix)
```

```
---------Conditional Entropy--------------------
Conditional Entropy: 0.17327577432161273
----------------Purity---------------------------
Per cluster purity: [1.0, 0.52, 1.0, 0.72, 1.0, 1.0, 0.99, 0.98, 1.0, 0.9, 0.55]
Purity: 0.9518049869743208
--------------Recalls---------------------------
Per cluster Recall: [0.351, 0.0482, 0.0081, 0.0135, 0.0827, 0.0047, 0.3281, 0.1329, 0.8629, 0.0281, 0.9474]
--------------F-measure---------------------------
F: 0.29239279534715756
```

✏️: Compare the results of K-Means and Normalized Cut clustering in terms of the number of detected anomalies and their characteristics.

- Number of normal samples using spectral clustering= 3656
- Number of abnormal samples using spectral clustering= 1718
- Number of normal samples using k means clustering= 4149
- Number of abnormal samples using k means clustering= 1225

| Spectral clustering labels | K-means labels |
|---|---|
| `new_labels_spectral`<br>`]: {0: 'normal.',`<br>`1: 'teardrop.',`<br>`2: 'teardrop.',`<br>`3: 'teardrop.',`<br>`4: 'teardrop.',`<br>`5: 'teardrop.',`<br>`6: 'pod.',`<br>`7: 'normal.',`<br>`8: 'normal.',`<br>`9: 'normal.',`<br>`10: 'neptune.'}` | `new_labels_kmeans`<br>`]: {0: 'normal.',`<br>`1: 'normal.',`<br>`2: 'normal.',`<br>`3: 'normal.',`<br>`4: 'normal.',`<br>`5: 'neptune.',`<br>`6: 'normal.',`<br>`7: 'normal.',`<br>`8: 'neptune.',`<br>`9: 'normal.',`<br>`10: 'normal.'}` |

## 7. New Clustering Algorithm: Gaussian Mixture Model GMM

A Gaussian Mixture Model (GMM) is a probabilistic model that is widely used for clustering and density estimation tasks. The model assumes that the data is generated from a mixture of several Gaussian distributions with different means and variances. GMMs are widely used in various applications such as speech recognition, image segmentation, and anomaly detection.

The main idea behind the GMM is to represent the data as a combination of several Gaussian distributions. Each Gaussian distribution in the model represents a cluster in the data. The goal of the GMM is to find the parameters of these Gaussian distributions

that best fit the data. These parameters include the mean, variance, and the weight of each Gaussian distribution in the mixture.

The GMM algorithm works by iteratively estimating the parameters of the Gaussian distributions. The algorithm starts by randomly initializing the parameters of the Gaussian distributions. Then, it calculates the probability of each data point belonging to each of the Gaussian distributions using Bayes' theorem. Based on these probabilities, the algorithm updates the parameters of the Gaussian distributions to maximize the likelihood of the data.

The algorithm iteratively repeats the above steps until convergence. At convergence, the GMM produces the parameters of the Gaussian distributions that best fit the data. These parameters can be used to cluster the data by assigning each data point to the Gaussian distribution with the highest probability.

One of the advantages of the GMM is that it can capture the complex structure of the data by modeling it as a mixture of several Gaussian distributions. The GMM is also a flexible model that can handle data with different shapes and sizes. Additionally, the GMM provides a probabilistic framework for clustering that allows for uncertainty in the clustering results.

In conclusion, the GMM is a powerful probabilistic model that is widely used for clustering and density estimation tasks. It is a flexible model that can handle various types of data and can capture the complex structure of the data by modeling it as a mixture of several Gaussian distributions.

In a one dimensional space, the probability density function of a Gaussian distribution is given by:

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where μ is the mean and σ2 is the variance.

But this would only be true for a single variable. In the case of two variables, instead of a 2D bell-shaped curve, we will have a 3D bell curve as shown below:

The probability density function would be given by:

$$f(x \mid \mu, \Sigma) = \frac{1}{\sqrt{2\pi|\Sigma|}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^t \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right]$$

Expectation-Maximization (EM) is a statistical algorithm for finding the right model parameters. We typically use EM when the data has missing values, or in other words, when the data is incomplete. Expectation-Maximization tries to use the existing data to determine the optimum values for these variables and then finds the model parameters. Based on these model parameters, we go back and update the values for the latent variable, and so on.
Broadly, the Expectation-Maximization algorithm has two steps:

- E-step: In this step, the available data is used to estimate (guess) the values of the missing variables

For each point xi, calculate the probability that it belongs to cluster/distribution c1, c2, ... ck. This is done using the below formula:

$$r_{ic} = \frac{\text{Probability Xi belongs to c}}{\text{Sum of probability Xi belongs to } c_1, c_2, .. c_k} = \frac{\pi_c \mathcal{N}(x_i \; ; \; \mu_c, \Sigma_c)}{\sum_{c'} \pi_{c'} \mathcal{N}(x_i \; ; \; \mu_{c'}, \Sigma_{c'})}$$

This value will be high when the point is assigned to the right cluster and lower otherwise.

- M-step: Based on the estimated values generated in the E-step, the complete data is used to update the parameters

Post the E-step, we go back and update the Π, μ and Σ values. These are updated in the following manner:

- The new density is defined by the ratio of the number of points in the cluster and the total number of points:

$$\Pi = \frac{\text{Number of points assigned to cluster}}{\text{Total number of points}}$$

- The mean and the covariance matrix are updated based on the values assigned to the distribution, in proportion with the probability values for the data point. Hence, a data point that has a higher probability of being a part of that distribution will contribute a larger portion:

$$\mu = \frac{1}{\text{Number of points assigned to cluster}} \sum_i r_{ic} x_i$$

$$\Sigma_c = \frac{1}{\text{Number of points assigned to cluster}} \sum_i r_{ic} (x_i - \mu_c)^T (x_i - \mu_c)$$

✏️:k-means only considers the mean to update the centroid while GMM takes into account the mean as well as the variance of the data!

## 8. GMM Clustering Evaluation

```
contingency_matrix = get_contingency(labels,labels_kmeans)
evaluation(np.array(data_k_means),contingency_matrix)
```

```
---------Conditional Entropy--------------------
Conditional Entropy: 0.17864866395510653
----------------Purity---------------------------
Per cluster purity: [0.6, 1.0, 1.0, 1.0, 0.54, 0.76, 0.95, 0.76, 1.0, 1.0, 0.71, 0.87, 0.89, 1.0, 0.5, 1.0, 0.91, 0.91, 0.8
5, 0.89, 0.45, 0.97, 0.4]
Purity: 0.956383168711277
--------------Recalls---------------------------
Per cluster Recall: [0.0158, 0.7774, 0.0024, 0.1838, 0.0042, 0.0118, 1.0, 0.0346, 0.0, 0.6992, 0.75, 0.0539, 0.066, 0.9913,
0.125, 0.0001, 0.0001, 0.0005, 0.9811, 0.0622, 0.0233, 0.0445, 0.6]
--------------F-measure---------------------------
F: 0.30014664845251915
```

## 9. K-means evaluation using Test Set! [Bonus Test]

Trying to evaluate the clustering K-means by mapping the output labels to the true classes by getting the max occurrence of corresponding labels in ground truth. Then predict via KNN and check accuracies.

```
# map labels resulting in k-means to true labels in able to do predictions
def map_and_change(y_train, labels):
    mapping = {}
    labels = np.array(list(labels))
    for i in np.unique(labels):
        binary = [int(x) for x in labels == i]
        mapping[i] = np.bincount([value for value, flag in zip(y_train, binary) if flag == 1]).argmax()

    # Map the cluster labels to the true class labels
    mapped_labels = np.array([mapping[label] for label in labels])

    # Print the mapped labels
    print(mapping)
    return mapping, mapped_labels
```

```
def map_and_change_test(mapping, labels):
    mapped_labels = np.array([mapping[label] for label in labels])
    return mapped_labels
```

```
X_train, X_test, y_train, y_test = train_test_split(data_spectral, labels_spectral, test_size=0.995, train_size=0.005,strati
```

```
kmeans = KMeans(n_clusters=23, random_state=42)
```

```
kmeans.fit(X_train)
```

```
KMeans(n_clusters=23, random_state=42)
```

```
train_labels = kmeans.labels_
```

```
mapping, train_labels = map_and_change(y_train, train_labels)
```

```
{0: 11, 1: 9, 2: 9, 3: 11, 4: 11, 5: 20, 6: 11, 7: 11, 8: 11, 9: 11, 10: 11, 11: 11, 12: 11, 13: 11, 14: 11, 15: 11, 16: 11, 1
7: 17, 18: 5, 19: 11, 20: 11, 21: 11, 22: 11}
```

```
test_labels = kmeans.predict(X_test)
```

```
test_labels = map_and_change_test(mapping,test_labels)
```

```
accuracy = accuracy_score(test_labels, y_test)
```

```
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.9860987754506749
```