

Network Anomaly Detection

Assignment 2

Hazem Mohammed Abdallah 6723

Ahmed Ashraf Abdelkarim 6940

Amr Abdelsamee Yousef 7126

**Introduction:**

In this lab we are testing different unsupervised clustering algorithms where we don’t have labels for each data entry and then we will test our clustering algorithms using different performance measures.

The dataset used for this assignment is the KDD CUP 1999 dataset which is a widely

used benchmark dataset for network anomaly detection.  This dataset contains

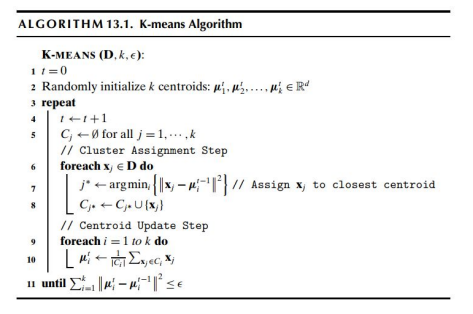
network traffic data collected from a simulated environment, including features

such as protocol type, service, source and destination IP addresses, source and

destination ports, and attack types.and we are only using 25% of this data.

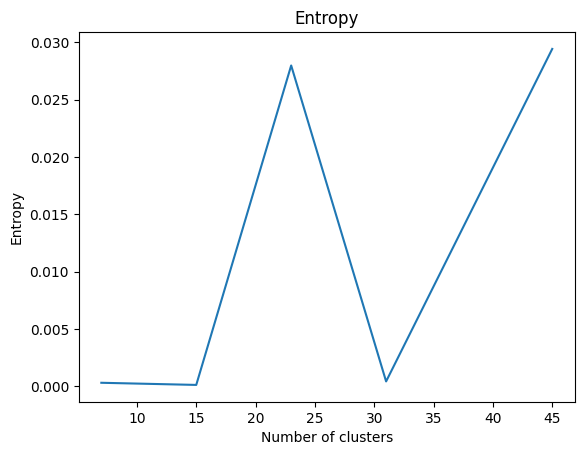
**K-means**

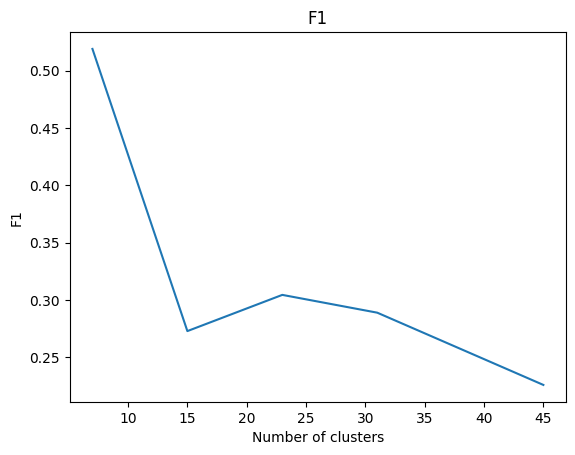
k-means clustering is a method of [vector quantization](https://en.wikipedia.org/wiki/Vector_quantization), originally from [signal processing](https://en.wikipedia.org/wiki/Signal_processing), that aims to [partition](https://en.wikipedia.org/wiki/Partition_of_a_set) n observations into k clusters in which each observation belongs to the [cluster](https://en.wikipedia.org/wiki/Cluster_(statistics)) with the nearest [mean](https://en.wikipedia.org/wiki/Mean) (cluster centers or cluster [centroid](https://en.wikipedia.org/wiki/Centroid)), serving as a prototype of the cluster. This results in a partitioning of the data space into [Voronoi cells](https://en.wikipedia.org/wiki/Voronoi_cell). k-means clustering minimizes within-cluster variances ([squared Euclidean distances](https://en.wikipedia.org/wiki/Squared_Euclidean_distance)), but not regular Euclidean distances, which would be the more difficult [Weber problem](https://en.wikipedia.org/wiki/Weber_problem): the mean optimizes squared errors, whereas only the [geometric median](https://en.wikipedia.org/wiki/Geometric_median) minimizes Euclidean distances. For instance, better Euclidean solutions can be found using [k-medians](https://en.wikipedia.org/wiki/K-medians_clustering) and [k-medoids](https://en.wikipedia.org/wiki/K-medoids).

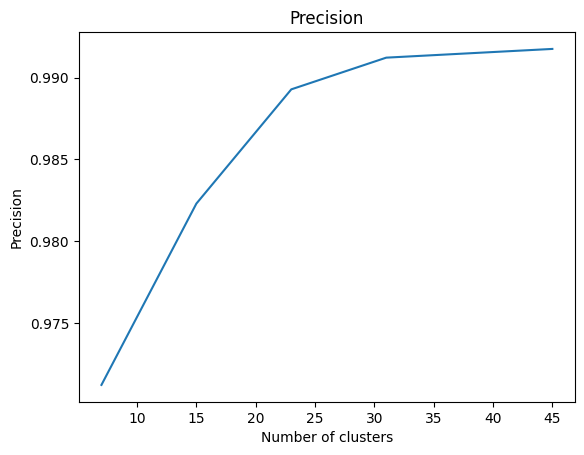
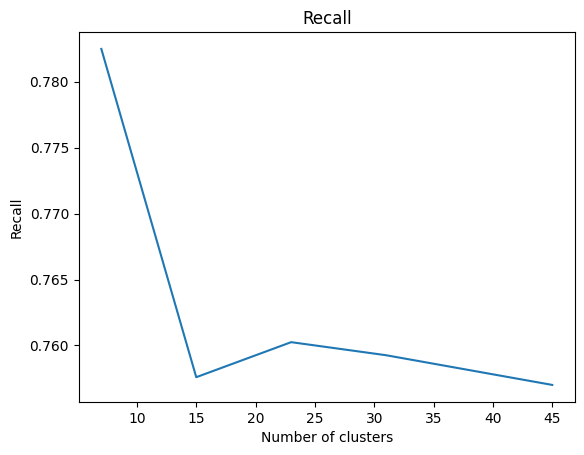


Kmeans is tested in 7,15,23,31,45 clusters and the performance measures were measured for all those clusters

The result for different values of k is:







**Spectral Clustering Algorithm:**

***implementation***

K-means clustering is the most well-known and representative clustering algorithm. Probably, you may also know the algorithm, but it has a limitation that comes from its assumption. K-means clustering assumes that the data lie in convex sets. In other words, the algorithm does not work properly for non-convex shaped data. In this case, spectral clustering can be an alternative for such data. The figure below shows the different behaviors of K-means and spectral clustering algorithms for concentric circles.

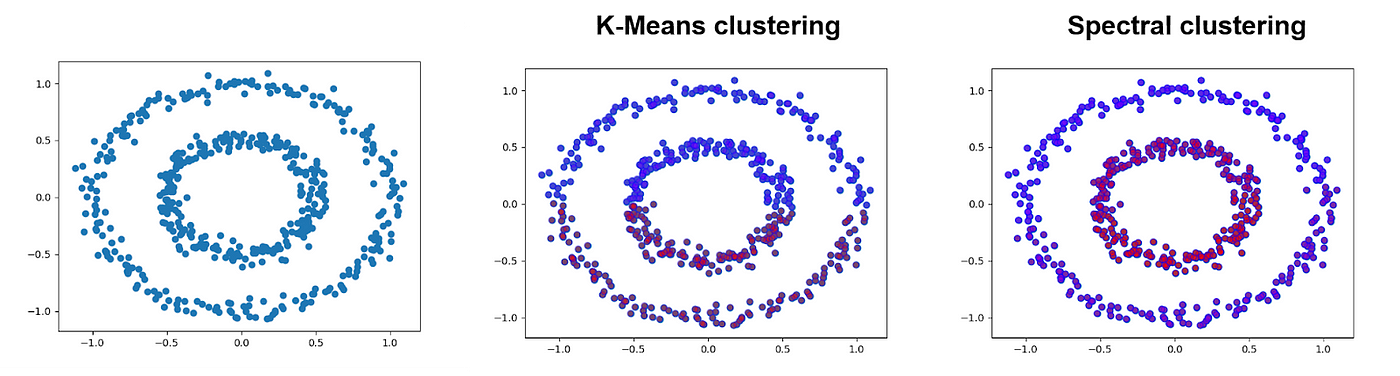
Due to the excellence of the algorithm for general problems the scope of its applications is very broad. Segmentation of 3D point clouds can be one of the examples, and Figure 2 represents segmentation results of point clouds to represent trees.

Steps used to implement the algorithms:

1- Construct a similarity graph (KNN, 𝜺-neighborhood, or the fully connected graph). Let A be its adjacency matrix.

2- Compute the degree matrix D using the similarity matrix.

3- Compute the normalized graph Laplacian matrix L.

4- Compute eigenvectors u₁, u₂, …, uₖ , whose corresponding eigenvalues are the k smallest ones of L respecting multiplicity.

5- Let U be an n by k matrix [u₁, u₂, …, uₖ] where its columns are the eigenvectors.

6- Cluster the points y₁, y₂, …, yₙ into clusters C₁, C₂, …, Cₖ by K-means.

***results***

**My implementations of a clustering algorithm:**

***A)DB-scan:***

What Exactly is DBSCAN Clustering?

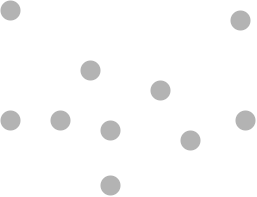
DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise.

It groups ‘densely grouped’ data points into a single cluster. It can identify clusters in large spatial datasets by looking at the local density of the data points. The most exciting feature of DBSCAN clustering is that it is robust to outliers. It also does not require the number of clusters to be told beforehand, unlike K-Means, where we have to specify the number of centroids.

DBSCAN requires only two parameters: *epsilon* and *minPoints*. *Epsilon* is the radius of the circle to be created around each data point to check the density and *minPoints* is the minimum number of data points required inside that circle for that data point to be classified as a Core point.

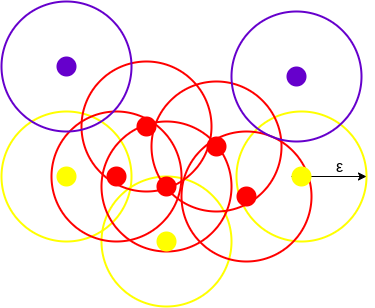
In higher dimensions the circle becomes hypersphere, *epsilon* becomes the radius of that hypersphere, and *minPoints* is the minimum number of data points required inside that hypersphere.

Sounds confusing? Let’s understand it with the help of an example.



Here, we have some data points represented by grey color. Let’s see how DBSCAN clusters these data points.

DBSCAN creates a circle of *epsilon* radius around every data point and classifies them into Core point, Border point, and Noise. A data point is a Core point if the circle around it contains at least ‘*minPoints’* number of points. If the number of points is less than *minPoints*, then it is classified as Border Point, and if there are no other data points around any data point within *epsilon* radius, then it treated as Noise.



The above figure shows us a cluster created by DBCAN with *minPoints = 3*. Here, we draw a circle of equal radius *epsilon* around every data point. These two parameters help in creating spatial clusters.

All the data points with at least 3 points in the circle including itself are considered as Core points represented by red color. All the data points with less than 3 but greater than 1 point in the circle including itself are considered as Border points. They are represented by yellow color. Finally, data points with no point other than itself present inside the circle are considered as Noise represented by the purple color.

For locating data points in space, DBSCAN uses [Euclidean distance](https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/?utm_source=blog&utm_medium=DBSCAN), although other methods can also be used (like great circle distance for geographical data). It also needs to scan through the entire dataset once, whereas in other algorithms we have to do it multiple times.

Reachability and Connectivity

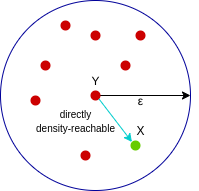
These are the two concepts that you need to understand before moving further. Reachability states if a data point can be accessed from another data point directly or indirectly, whereas Connectivity states whether two data points belong to the same cluster or not. In terms of reachability and connectivity, two points in DBSCAN can be referred to as:

* Directly Density-Reachable
* Density-Reachable
* Density-Connected

Let’s understand what they are.

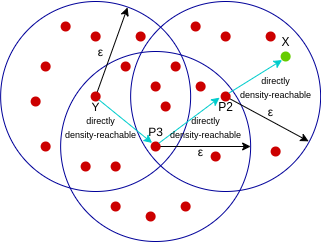
A point X is directly density-reachable from point Y w.r.t *epsilon, minPoints* if,

* X belongs to the neighborhood of Y, i.e, *dist(X, Y) <= epsilon*
* Y is a core point



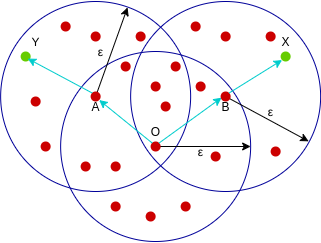
Here, X is directly density-reachable from Y, but vice versa is not valid.

A point X is density-reachable from point Y w.r.t *epsilon, minPoints* if there is a chain of points p1, p2, p3, …, pn and p1=X and pn=Y such that pi+1 is directly density-reachable from pi.



Here, X is density-reachable from Y with X being directly density-reachable from P2, P2 from P3, and P3 from Y. But, the inverse of this is not valid.

A point X is density-connected from point Y w.r.t *epsilon and minPoints* if there exists a point O such that both X and Y are density-reachable from O w.r.t to *epsilon and minPoints.*

**

Here, both X and Y are density-reachable from O, therefore, we can say that X is density-connected from Y.

Parameter Selection in DBSCAN Clustering

DBSCAN is very sensitive to the values of *epsilon* and *minPoints*. Therefore, it is very important to understand how to select the values of *epsilon* and *minPoints*. A slight variation in these values can significantly change the results produced by the DBSCAN algorithm.

The value of *minPoints* should be at least one greater than the number of dimensions of the dataset, i.e.,

*minPoints>=Dimensions+1*.

It does not make sense to take *minPoints* as 1 because it will result in each point being a separate cluster. Therefore, it must be at least 3. Generally, it is twice the dimensions. But domain knowledge also decides its value.

The value of *epsilon* can be decided from the K-distance graph. The point of maximum curvature (elbow) in this graph tells us about the value of *epsilon*. If the value of *epsilon* chosen is too small then a higher number of clusters will be created, and more data points will be taken as noise. Whereas, if chosen too big then various small clusters will merge into a big cluster, and we will lose details.

And to get the value of epsilon the distance was plotted between all points and we take the value at the foot of the curve before going up.

**The result of the performance measures**

**Precision = 43.245**

**recall = 53.45818653810495**

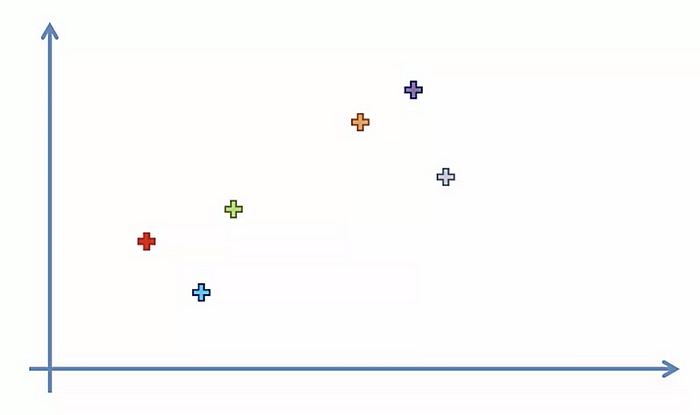
**f1 = 49.906133392299397**

**entropy = 0.5492348132445**

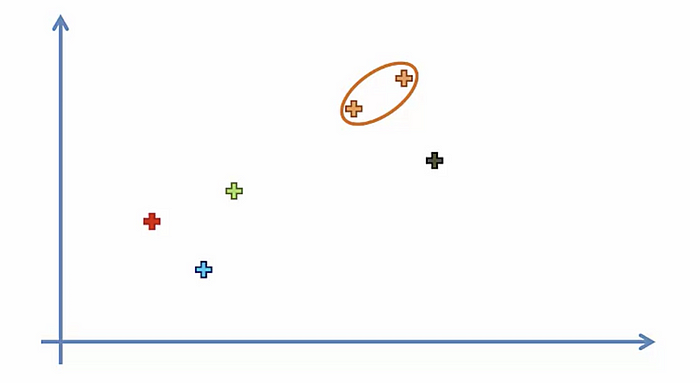
**B) *Agglomerative* *clustering***

**How it works**

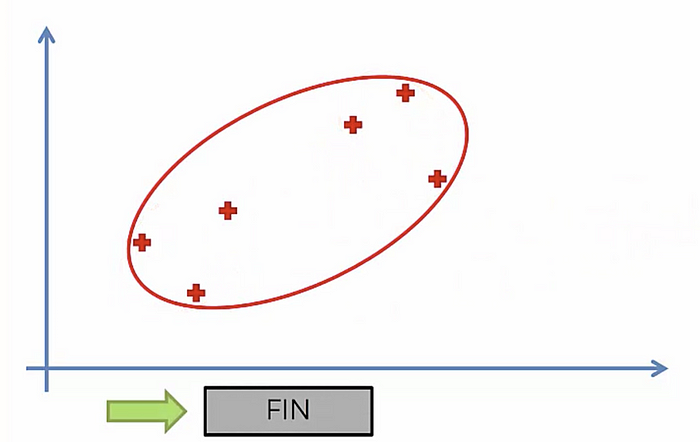
1. Make each data point a cluster



2. Take the two closest clusters and make them one cluster



3. Repeat step 2 until there is only one cluster



The biggest issue with this clustering algorithm is that it takes O(𝑛2 log(𝑛)) which took too much time to cluster