Computer network Intrusion Detection:

Name	ID
Hossam el-din ahmed	6721
Adham el Kazzaz	6713
Saeed abokhatwa	6829

Intro:

The KDD-CUP-1999 dataset is a famous benchmark dataset used in the field of network intrusion detection. It was created for a competition organized by the University of California, Irvine (UCI), and the International Conference on Knowledge Discovery and Data Mining (KDD). The goal of the competition was to encourage the development of effective and efficient methods for detecting network intrusions in real-time by analyzing network traffic data. The dataset consists of a set of network traffic data captured over nine weeks from an environment that simulated a typical US Air Force local area network (LAN), subjected to various types of attacks. The dataset has become a standard benchmark for evaluating the performance of intrusion detection systems (IDSs), and it is often used to test and compare the effectiveness of various machine learning algorithms and techniques for detecting different types of network attacks.

(Attack vs normal)

[23]:
```
print(f'precentage of unique data is {(len(training_df)-len(training_df.drop_duplicates(
```

```
precentage of unique data is 70.53040255373759 %
```

[24]:
```
def remove_duplicates(df):
    # count the duplicate rows and add this count as a new column
    duplicates_count = df.groupby(list(df.columns)).size().reset_index().rename(columns=
    df_with_counts = pd.merge(df, duplicates_count, on=list(df.columns), how='left')
    return df_with_counts.drop_duplicates()
```

Since 30% of the data are duplicated , removing the duplicates will improve the effectiveness of clustering but also just removing is not the optimal solution , in this case I introduced a new feature to each instance named “duplicate count” which state the number of times the instance was repeated throughout the dataset.

This will keep the information of the rows that was removed.

```
training_df['Duplicate_count'].value_counts()
```

```
1      126209
2        8251
3        3625
4        2165
5        1375
        ...
148         1
127         1
153         1
111         1
59          1
Name: Duplicate_count, Length: 147, dtype: int64
```

[30]:

```
training_df.corr()['target_encoded'].sort_values()
```

[30…

```
dst_host_serror_rate       -0.326148
srv_serror_rate            -0.325529
dst_host_srv_serror_rate   -0.325256
flag_S0                    -0.325138
serror_rate                -0.323583
                              ...
wrong_fragment              0.469979
target_encoded              1.000000
num_outbound_cmds                NaN
is_host_login                    NaN
service_icmp                     NaN
Name: target_encoded, Length: 121, dtype: float64
```

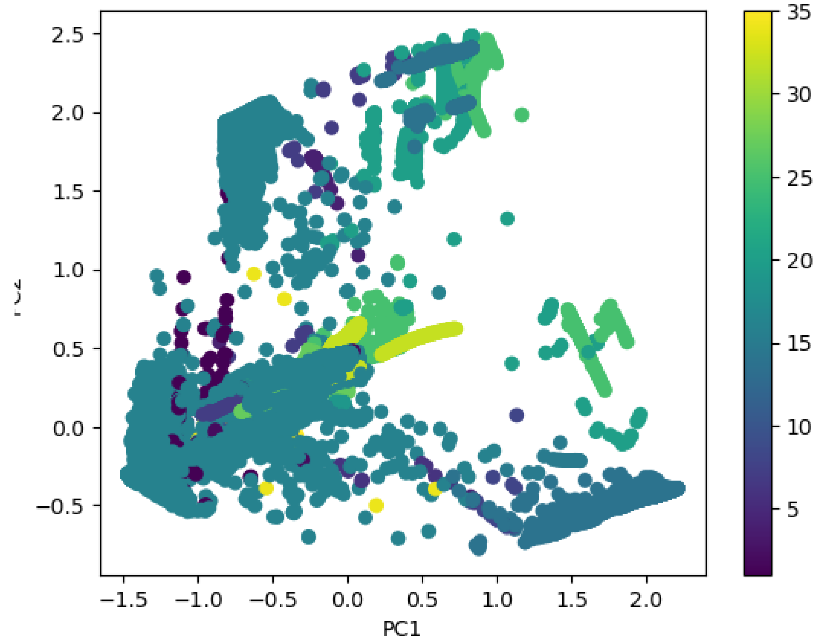
num_outbound_cmds NaN is_host_login NaN so dropping them will affect nothing

[31]:

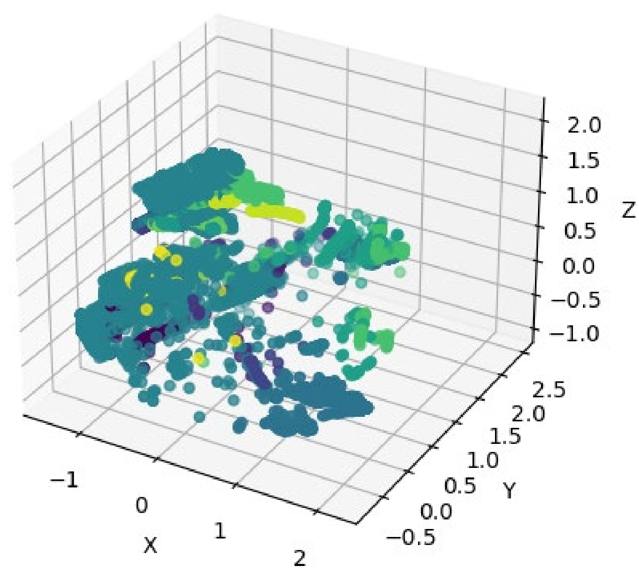
```
training_df.drop(inplace=True,columns=['num_outbound_cmds','is_host_login','service_icmp
testing_df.drop(inplace=True,columns=['num_outbound_cmds','is_host_login','service_icmp'
```

dropped the feature who has no correlation with the target label, this will save us some time and will provide better results.

Visualization of the data using PCA

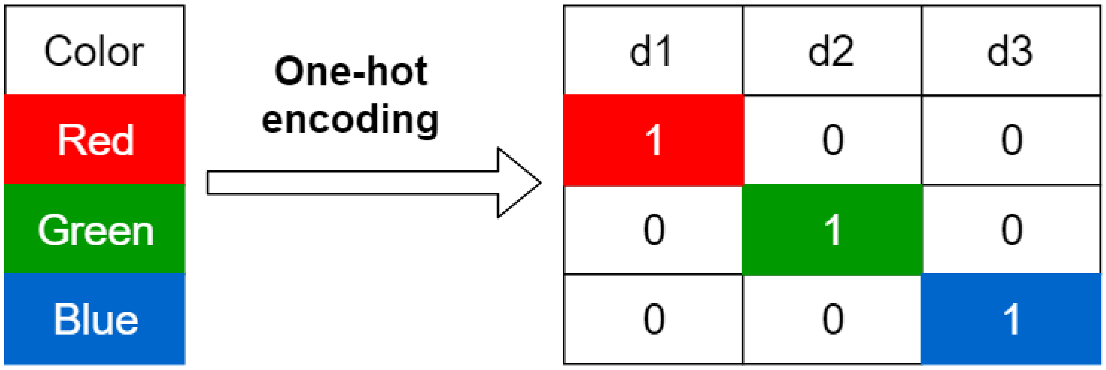


It’s obvious here the result of any clustering algorithm will be bad since there is overlapping points of different classes and more likely to cluster different classes together.



3d-plot

Encoding chosen -> OneHotEncoding:



One hot encoding is a popular technique used in machine learning to represent categorical data as numerical values. It works by converting each category value into a binary vector that has a dimension equal to the total number of categories. In this vector, only one bit is set to 1, representing the category value, while all the other bits are set to 0. One hot encoding is used because many machine learning algorithms require numerical input data, and categorical data cannot be used directly. By converting categorical data into numerical values using one hot encoding, machine learning models can effectively process and analyze categorical data.

Kmeans clustering:

On training data:

K	Precision	Recall	F-measure	Entropy
7	[0.9972191323692993, 0.6296201067043774, 0.9830784022375666, 0.960945323452834, 0.9364118092354277, 0.9990163594947195, 0.8205407614247769]	[0.3460054033191818, 0.1890196835198765, 0.6402791693232535, 0.09258973369355461, 0.16900446306585298, 0.3723851794673871, 0.13717096274706259]	0.4018305444382901	0.35811831459179383
15	[0.9994793293762366, 0.9981499356499357, 0.3540229885057471 5, 0.9879336349924586, 1.0, 0.8678068410462777, 0.7734056987788331, 0.9832057017211312, 0.8933764135702746, 0.8803962333374098, 0.5408163265306123, 0.9864864864864865, 0.9662261380323054, 0.9613303947104789, 0.9061054579093432]	[0.1092768011658621, 0.7183905827865689, 0.9609984399375975, 0.03728709354221696, 0.28860779670279624, 0.049105109754986796, 0.019468986246470536, 0.29527962473813646, 0.012592221513799072, 0.08196329356043355, 0.0042239730394389285, 0.05319245832953821, 0.04494944894799162, 0.09258973369355461, 0.1890196835198765]	0.23376920248258506	0.19716632018485683
23	[0.9861949956859362, 0.8588957055214724, 0.9988998604701084, 0.9994347088750707, 0.9872448979591837, 0.4, 1.0, 0.8804547121378804, 0.6781115879828327, 0.8679775280898876, 0.975557917109458, 0.9843665318714303, 0.9610829103214 89, 0.9381255686988171, 1.0, 0.9732489732489733, 0.6763990267639	[0.22122916476910465, 0.006375808361417251, 0.7183905827865689, 0.060388013480280535, 0.004406138992622279, 0.0007514345568813189, 0.03709354221695965, 0.0820088350487294, 0.9859594383775351, 0.7427884615384616, 0.03135531469168412, 0.2688314054103288, 0.00646689133	0.23539282598618855	0.18424488025534586

	902, 0.9662261380323054, 0.9696784696784697, 0.7160804020100503, 0.933835446658404, 0.8561151079136691, 0.9975594874923734]	8008926, 0.011738318608252118, 0.0486269241278805, 0.09982694234447581, 0.8540706605222734, 0.04494944894799162, 0.09195291393284447, 0.009734493123235268, 0.1890196835198765, 0.04335549685763731, 0.018615083340923582]		
31	[0.9878938857567414, 0.9402899107228568, 0.7191011235955056, 0.9506048387096774, 0.9952963311382879, 0.5107913669064749, 0.9692902176123652, 1.0, 1.0, 0.9834399698908544, 0.5648050579557429, 0.9859390518007423, 0.9976359338061466, 0.999767576990122, 0.925249169435216, 0.9204892966360856, 0.9988479262672811, 0.9998167155425219, 0.9966814159292036, 0.9142011834319527, 0.34557109557109555, 0.9991620404315492, 0.5771495877502945, 0.8022284122562674, 0.9862901014532492, 0.97469723183391, 0.988221081244337, 0.9831932773109243, 0.9706717123935666, 0.8606382978723405, 0.9994353472614342]	[0.25735495035977773, 0.1890196835198765, 0.7692307692307693, 0.01819760710150521, 0.01204572365424902, 0.0008083614172511158, 0.09197221150135083, 0.021700519172966573, 0.026903634210766008, 0.02974997722925585, 0.6002239641657335, 0.21475088805902176, 0.033632389106476, 0.3320339637205712, 0.006341652245195373, 0.0034269969942617726, 0.00987111758812278, 0.0621072046634484, 0.010258220238637398, 0.017590399854267236, 0.9251170046801872, 0.36815901196449247, 0.016736496948720283, 0.322508398656215, 0.040953183350031876, 0.10262774387466983, 0.03725293742599508, 0.005328354130612989, 0.011681391747882321, 0.009210766007833	0.18042175838862057	0.16280182471318053

		136, 0.060456325712724295]		
45	[0.999815973500184, 1.0, 0.4667093469910371, 1.0, 0.9706213417379559, 0.9494845360824742, 0.8333333333333334, 1.0, 0.9976359338061466, 1.0, 1.0, 0.9892413961733603, 1.0, 1.0, 0.7717171717171717, 0.9611166500498505, 1.0, 0.9992589455854329, 0.9654978962131837, 0.9758874748827864, 0.9766666666666667, 1.0, 0.538235294117647, 0.9836244541484717, 0.5887445887445888, 0.9995698924731182, 0.9822784810126582, 0.9165919282511211, 0.9996424740793708, 0.6587677725118484, 0.9780534351145038, 1.0, 0.6876404494382022, 0.999759239195859, 0.9934904996481351, 0.9414141414141414, 0.9997076023391813, 0.9041666666666667, 1.0, 1.0, 0.9932432432432432, 0.8594386600271616, 0.8950437317784257, 0.9993510707332901, 0.9962292609351433]	[0.0618567264778213, 0.0433441114855633 5, 0.008299936241916385, 0.014948993533108662, 0.16640293323041297, 0.01777306059436511, 5.692686036979689e-05, 0.11344384734493124, 0.033632389106476, 0.024410237726568904, 0.018854176154476727, 0.1842494762728846, 0.6346578366445916, 0.035715912196010564, 0.01739684852900993, 0.01097549867929684, 0.05244102377265689, 0.18214974913160942, 0.039188450678568174, 0.016588487111758813, 0.022616750289463528, 0.05279814743342339, 0.004167046179069132, 0.010258220238637398, 0.0015484106020584752, 0.02645960469988159, 0.0044175243646962 38, 0.011635850259586484, 0.03183350031879042, 0.8540706605222734, 0.011670006375808362, 0.06335959559158394, 1.0, 0.3205326128907758, 0.0642931961016486, 0.0106111667729301 4, 0.1979351601698186, 0.008375144731763798, 0.013639675744603334, 0.0016981860285604014, 0.00669459	0.14809281862049387	0.1269048253199128

		8779488113, 0.043230257764823754, 0.7379807692307693, 0.029718255499807025, 0.015040076509700337]		

On testing data:

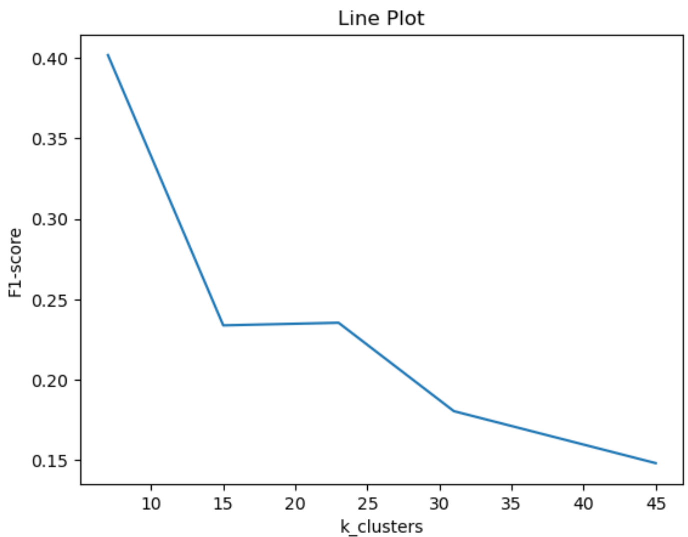
K	Precision	Recall	F-measure	Entropy
7	[0.9721153846153846, 0.8801156440732413, 0.9697504035763069, 0.5802332037756802, 0.609375, 0.998046875, 0.6488982501620221]	[0.14917371630926618, 0.6737654928192013, 0.81495627491495, 0.05139681290576431, 0.09930499029490952, 0.12566397796576825, 0.08358900507169244]	0.3634380722931391	0.6763361541569572
15	[0.908520946138501, 0.9837713882518963, 0.6797047970479705, 0.9957912457912458, 1.0, 0.6419408812046848, 0.8952583156404812, 0.912260793315108, 0.3393017009847807, 0.8678925719816114, 0.6900662251655629, 0.5025773195876289, 0.6007130124777184, 0.5851063829787234, 0.89556238154369]	[0.06653726546031348, 0.27429667519181583, 0.9839743589743589, 0.04938117003735938, 0.3704213887671404, 0.02402270782459875, 0.026402020328512094, 0.3691273767036086, 0.007910170517396113, 0.07486485922401019, 0.5199600798403193, 0.2455919395465995, 0.3918604651162791, 0.05139681290576431, 0.6739622270312807]	0.3352736762226111	0.5412498185554266
23	[1.0, 0.7966101694915254, 0.9973171167948489, 0.7805325987144169, 0.8510301109350238, 0.32242424242424245, 0.001, 0.8678925719816114, 0.8559556786703602, 0.4101796407185629, 0.9990034877927254, 0.928	[0.28136413916891034, 0.0019618892576127563, 0.27424749163879597, 0.03548097593554985, 0.6763224181360201, 0.3575268817204301, 0.001, 0.07486485922401019, 0.9903846153846154, 0.1306005	0.3230314098727655	0.4437368964474166

	8025889967637, 0.31752178121974833, 0.7623762376237624, 0.9873501997336884, 1.0, 0.551094890510949, 0.6017857142857143, 0.898989898989899, 0.522567703109328, 0.9192909420533136, 0.6719846841097639, 1.0]	719733079, 0.041846680441633796, 0.33544132072715127, 0.44086021505376344, 0.14680648236415633, 0.03095193371318849, 0.1316135495585749, 0.9741935483870968, 0.3918604651162791, 0.048150698406452884, 0.5199600798403193, 0.6733720243950423, 0.02197733391772588, 0.02469058501867969]		
31	[0.9178419252235491, 0.9245627068278517, 0.5398550724637681, 0.8, 0.6666666666666666, 0.35046113306982873, 0.8924339106654512, 0.9874651810584958, 1.0, 0.001, 0.9205479452054794, 1.0, 0.6923076923076923, 1.0, 0.37792397660818716, 0.6918990703851262, 0.8666127728375101, 1.0, 1.0, 0.9483050847457627, 0.6788648244958925, 0.998375964271214, 0.6324503311258278, 0.001, 0.5947521865889213, 1.0, 1.0, 0.9095477386934674, 0.7522727272727273, 0.8010204081632653, 0.8424182358771061]	[0.31920355644605847, 0.6733228408420224, 0.2840800762631077, 0.95, 0.11764705882352941, 0.3575268817204301, 0.048150698406452884, 0.01479765408135579, 0.016196021956462754, 0.001, 0.33532934131736525, 0.27261912215891304, 0.0009392023041763195, 0.15301003344481606, 0.3970814132104455, 0.5199600798403193, 0.022373886001711436, 0.05468244526537683, 0.014651555945150585, 0.023354830630517815, 0.9711538461538461, 0.12094235687586072, 0.019931960010853005, 0.001, 0.25692695214105793, 0.13497380669129463, 0.04973597979671488, 0.0037776803790203076, 0.38488372093023254, 0.00327677248345	0.25575791551987453	0.42899907916190977

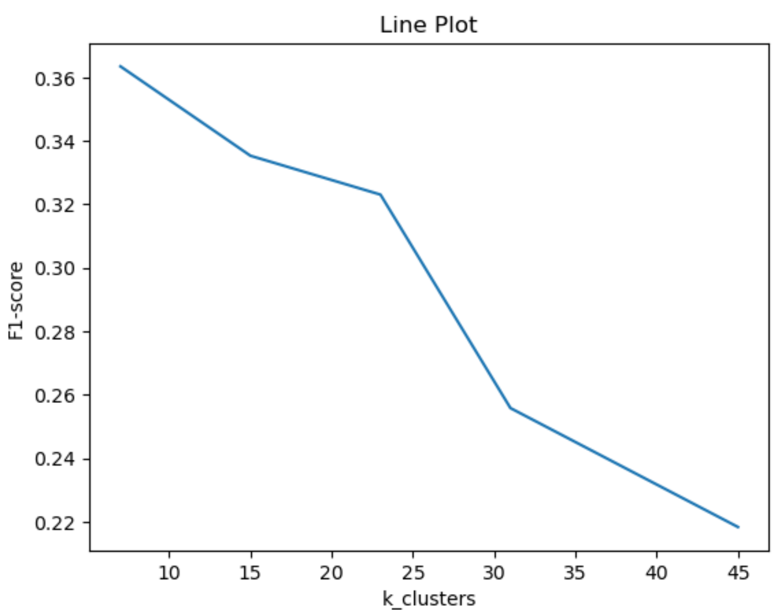
		9604, 0.03548097593554985]		
45	[1.0, 1.0, 0.7959183673469388, 1.0, 0.9833884153765873, 0.8172043010752689, 0.6, 1.0, 0.6923076923076923, 1.0, 0.9873417721518988, 0.9351612038939001, 0.690671031096563, 0.5652173913043478, 0.9255928045789044, 0.3443223443223443, 1.0, 0.9983416252072969, 0.98413344182262, 0.6216798277099784, 0.7536186280679673, 1.0, 0.6951871657754011, 0.9929873772791024, 0.8579088471849866, 0.6514767932489451, 0.5404040404040404, 0.9640479360852197, 0.9607623318385651, 0.5067114093959731, 0.8026666666666666, 1.0, 0.7090163934426229, 1.0, 0.971926647045506, 0.6434108527131783, 1.0, 0.34822601839684625, 0.9976580796252927, 1.0, 1.0, 0.6741790083708951, 0.6706586826347305, 0.9967105263157895, 1.0]	[0.05464070294074677, 0.05649823638678438, 1.0, 0.008515434224531965, 0.5561184339956718, 0.95, 6.261348694508798e-05, 0.1436979525389769, 0.0009392023041763195, 0.03122325882328387, 0.013023605284578299, 0.21854194060067206, 0.4906976744186046, 0.0163727959697733, 0.023626155740613195, 0.6317204301075269, 0.07284035647945233, 0.059216997835923664, 0.05048734164005594, 0.0180744265648153930, 0.11779460948258902, 0.027591973244147156, 0.5189620758483033, 0.01477678291904076, 0.8060453400503779, 0.016112537307202638, 0.0022332143677081377, 0.01511072151608123, 0.01788658610398013, 0.9741935483870968, 0.35, 0.07707720242940329, 0.003610711080500073, 0.1353039543576628, 0.08959989981842088, 0.07912297426120114, 0.07952980523313004, 0.35618279569892475, 0.008891115146202493, 0.0009836710603974032, 2.0871162315029324e-05, 0.021852106943835704, 0.64	0.21827929427567405	0.34659061464387547

		36781609195402, 0.014902616565020657, 0.021142487425124704]		

Training graph:



Testing graph:



Testing K_means on iris dataset:

presicion :
[0.8333333333333334, 1.0, 0.8076923076923077]

recall :
[0.8, 1.0, 0.84]

F_1 score :
0.8799519807923168

the conditional entropy of T given clustering C
[0.6500224216483541, 0.0, 0.7062740891876007]

entropy :
0.4528488591791749

Using sktlearn k-means:

presicion :
[1.0, 0.9230769230769231, 0.7704918032786885]

recall :
[1.0, 0.72, 0.94]

F_1 score :
0.8852785369639302

the conditional entropy of T given clustering C
[0.0, 0.39124356362925566, 0.7771529943226336]

entropy :
0.4177655442348108

Normalized Cut (Spectral Clustering):

Data preprocessing:

The KDD Cup 1999 was loaded into the notebook in a pandas dataframe, with number of rows equal to the number of entries in the dataset (4,898,431), and number of columns equal to the number of features. The features' names were retrieved from the dataset in order to name the corresponding columns after them.

```
training_data
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_diff_srv_rate	dst_host_same_src_port_rate	dst_host_srv_diff_
0	0	tcp	http	SF	215	45076	0	0	0	0	...	0	0.0	0.0	0.00	
1	0	tcp	http	SF	162	4528	0	0	0	0	...	1	1.0	0.0	1.00	
2	0	tcp	http	SF	236	1228	0	0	0	0	...	2	1.0	0.0	0.50	
3	0	tcp	http	SF	233	2032	0	0	0	0	...	3	1.0	0.0	0.33	
4	0	tcp	http	SF	239	486	0	0	0	0	...	4	1.0	0.0	0.25	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4898426	0	tcp	http	SF	212	2288	0	0	0	0	...	255	1.0	0.0	0.33	
4898427	0	tcp	http	SF	219	236	0	0	0	0	...	255	1.0	0.0	0.25	
4898428	0	tcp	http	SF	218	3610	0	0	0	0	...	255	1.0	0.0	0.20	
4898429	0	tcp	http	SF	219	1234	0	0	0	0	...	255	1.0	0.0	0.17	
4898430	0	tcp	http	SF	219	1098	0	0	0	0	...	255	1.0	0.0	0.14	

4898431 rows × 42 columns

To be able to operate on the data, we had to encode 4 categorical features into numerical features. As for the protocol_type, service and flag features, one hot encoding was applied. As for the target feature, label encoding was applied.

```
icmp     2833545
tcp      1870598
udp       194288
Name: protocol_type, dtype: int64
3
ecr_i          2811660
private        1100831
http            623091
smtp             96554
other            72653
                 ...
tftp_u               3
harvest              2
aol                  2
http_8001            2
http_2784            1
Name: service, Length: 70, dtype: int64
70
SF         3744328
S0          869829
REJ         268874
RSTR          8094
RSTO          5344
SH            1040
S1             532
S2             161
RSTOS0         122
OTH             57
S3              50
Name: flag, dtype: int64
11
```

The original protocol_type, service and flag features were dropped, as they were replaced with the above one hot encoded features. One hot encoding was favored to avoid any unintentional biasing of the model calculations, as all one hot encoded features only take a 0 or 1 value.
As for the categorical values of the target feature, they were replaced with the label encoded numerical ones. Label encoding was used on this feature to avoid an unnecessary increase in dimensionality. The original dataframe had 42 features, after dropping 3 features, and adding 84 features, the encoded dataframe has a total of 123 features.

```
encoded_data = encode(training_data)
encoded_data
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	...	flag_REJ	flag_RSTO	flag_RSTOS0	flag_RSTR	flag_S0	flag_S1	flag_S2	flag_S3	flag_SF
0	0	215	45076	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	1
1	0	162	4528	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	1
2	0	236	1228	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	1
3	0	233	2032	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	1
4	0	239	486	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4898426	0	212	2288	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	1
4898427	0	219	236	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	1
4898428	0	218	3610	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	1
4898429	0	219	1234	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	1
4898430	0	219	1098	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	1

4898431 rows × 123 columns

Analysis of the target feature:

The target feature which contains the labels for each row of data, we are informed that our data is classified into a total of 23 classes, but not all classes are equally represented as shown in the figure. According to the paper "Reducing Redundancy in the KDD CUP 1999 Data Set" by Ali et al. (2011), the percentage of redundancies in the original KDD Cup 1999 dataset was found to be approximately 70%. This high percentage of redundancies can have a negative impact on the performance of machine learning models trained on the dataset. Although redundancies are important to consider, we had to remove them, and work on a small portion of the remaining data to be able to complete the clustering task.

```
smurf.              2807886
neptune.            1072017
normal.              972781
satan.                15892
ipsweep.              12481
portsweep.            10413
nmap.                  2316
back.                  2203
warezclient.           1020
teardrop.               979
pod.                    264
guess_passwd.            53
buffer_overflow.         30
land.                    21
warezmaster.             20
imap.                    12
rootkit.                 10
loadmodule.               9
ftp_write.                8
multihop.                 7
phf.                      4
perl.                     3
spy.                      2
Name: target, dtype: int64
```

train_test_split:

We used the sklearn function to get a training set containing 0.004 of the data (X_train), and the target feature was used as the (Y_train). However, we still had to remove redundancies in them. To be able to do this, and ensure that both their rows still correspond, they were combined in a single dataframe, and re-split into Xtrain_unique and Ytrain_unique. This leaves as with 7974 rows of data on which we apply the clustering.

```
#Combine to remove corresponding duplicates
combined = pd.concat([X_train, y_train], axis=1)
combined = combined.drop_duplicates()
combined
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	...	flag_RSTO	flag_RSTOS0	flag_RSTR	flag_S0	flag_S1	flag_S2	flag_S3	flag_SF	flag_SH
2451599	0	1032	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0
999149	0	1032	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0
4477733	0	145	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0
654692	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0
3418860	0	45	115	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
374455	0	415	10439	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	1	0
3651134	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0
636658	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0
3872637	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0
3547162	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0

7974 rows × 123 columns

Setting the # of clusters to the # of existing classes in the remaining data:

The remaining data consists of only 11 classes of the original 23. However, classes are still unequally represented, as the most represented class is represented almost equally to the sum of all other remaining classes.

9	3938
11	3758
18	123
17	57
5	47
15	27
0	9
10	6
20	4
21	4
14	1

Normalized Cut Algorithm:

Algorithm 16.1: Spectral Clustering Algorithm

SPECTRAL CLUSTERING (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. (16.23)

$\mathcal{C} \leftarrow \{C_1, \ldots, C_k\}$ via K-means on $\mathbf{Y}$

This pseudocode from the textbook was followed to apply spectral clustering:

The similarity matrix was computed using sklearn's rbf_kernel which computes the Guassian kernel. The degree matrix was then computed, and the Laplacian matrix was then computed as the difference between the similarity and degree matrix. To apply normalized cut, we computed the asymmetric Laplacian matrix using the following equation:

$$\mathbf{L}^a = \mathbf{\Delta}^{-1}\mathbf{L}$$

As this is an asymmetric matrix, eig was used to compute the eigenvalues and eigenvectors U. Only the real components of U were kept, and they were then normalized to return the unit vector Y, on which Kmeans will be applied.

Evaluation:

precision:

```
[0.9583741429970617,
 0.9518779342723005,
 0.9905660377358491,
 0.7651515151515151,
 0.9578256794751641,
 0.9915966386554622,
 0.9558011049723757,
 0.9415384615384615,
 0.8939051918735892,
 0.9739130434782609,
 0.9590643274853801]
```

recall:

```
[0.9939055358049771,
 0.21580627993613624,
 0.02794039382650346,
 0.08062799361362427,
 0.271953166577967,
 0.03139968068121341,
 0.04603512506652475,
 0.08142629058009579,
 0.10537519957424162,
 0.05960617349654071,
 0.04364023416711016б]
```

F1 score = 0.23949682543756703

Conditional entropy: 0.3464060044294866

We can see that except for the precision, the evaluation scores are affected due to the low quality and noisiness of the data upon which spectral clustering was applied. After all, a model is only as good as the data it trains on.

Further analysis:

We asked the spectral clustering algorithm to cluster the data into different numbers of clusters (7, 9, 13, 15).
For the numbers of clusters that are smaller than than the number of remaining classes (7, 9) in the data, an improved score of F1 and conditional entropy were recorded.

```
K = 7
F1 score = 0.35974169308547094
Conditional entropy: 0.4205720755700158
K = 9
F1 score = 0.29010667036097765
Conditional entropy: 0.3551763509243745
```

The opposite was recorded as the numbers of clusters exceeded the number of remaining classes (13, 15).

```
K = 13
F1 score = 0.20732136012072405
Conditional entropy: 0.32136218488940527
K = 15
F1 score = 0.18119829877015414
Conditional entropy: 0.30070067845164533
```

DBSCAN

I)Introduction)

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a popular unsupervised machine learning algorithm used for clustering data points in a dataset. It is designed to group together data points that are close to each other based on their density, while identifying outliers or noise points that do not belong to any cluster.

The algorithm works by defining a neighborhood around each data point, based on a specified radius. The density of each point is then calculated based on the number of neighboring points within that radius. A point is considered a "core point" if it has at least a minimum number of neighboring points within that radius.

Starting with a core point, the algorithm expands the cluster by iteratively adding neighboring points that also meet the density requirement, until there are no more qualifying points to add. This process is repeated until all points have been assigned to a cluster, or marked as noise points.

The algorithm is flexible in that it can handle clusters of arbitrary shapes, and does not require the number of clusters to be specified beforehand. However, it does require the setting of two parameters: the radius of the neighborhood (epsilon) and the minimum number of neighboring points required for a point to be considered a core point (min_samples).

Overall, DBSCAN is a powerful clustering algorithm that is widely used in a variety of applications such as image segmentation, anomaly detection, and customer segmentation in marketing.

Before using the algorithm, some data analysis was done on the data and the result was:

a. Reduced the dimensions of the dataset from 42 to 30
b. Dropping all the redundancy from the dataset
c. Using only 0.005 from the data

Pseudocode:

Algorithm 15.1: Density-based Clustering Algorithm

DBSCAN ($\mathbf{D}, \epsilon, minpts$):

$Core \leftarrow \emptyset$

foreach $\mathbf{x}_i \in \mathbf{D}$ **do** `// Find the core points`

- Compute $N_\epsilon(\mathbf{x}_i)$
- $id(\mathbf{x}_i) \leftarrow \emptyset$ `// cluster id for` $\mathbf{x}_i$
- **if** $N_\epsilon(\mathbf{x}_i) \geq minpts$ **then** $Core \leftarrow Core \cup \{\mathbf{x}_i\}$

$k \leftarrow 0$ `// cluster id`

foreach $\mathbf{x}_i \in Core$, *such that* $id(\mathbf{x}_i) = \emptyset$ **do**

- $k \leftarrow k + 1$
- $id(\mathbf{x}_i) \leftarrow k$ `// assign` $\mathbf{x}_i$ `to cluster id` k
- DensityConnected $(\mathbf{x}_i, k)$

$\mathcal{C} \leftarrow \{C_i\}_{i=1}^{k}$, where $C_i \leftarrow \{\mathbf{x} \in \mathbf{D} \mid id(\mathbf{x}) = i\}$

$Noise \leftarrow \{\mathbf{x} \in \mathbf{D} \mid id(\mathbf{x}) = \emptyset\}$

$Border \leftarrow \mathbf{D} \setminus \{Core \cup Noise\}$

return $\mathcal{C}, Core, Border, Noise$

DensityConnected ($\mathbf{x}, k$):

foreach $\mathbf{y} \in N_\epsilon(\mathbf{x})$ **do**

- $id(\mathbf{y}) \leftarrow k$ `// assign` $\mathbf{y}$ `to cluster id` k
- **if** $\mathbf{y} \in Core$ **then** DensityConnected $(\mathbf{y}, k)$

As seen in the pseudocode, We have 2 hyperparameters to tune which are epsilon and minpts.

We chose minpts according to the following equation:

minspts = D + 1

Where D is the number of dimensions in our dataset.

Now we need to get the epsilon value KNN Algorithm from Sklearn was used was a value of K equal to the number of features and we got 0.5 for epsilon.

Using those 2 hyperparameters we got the following results:

```
Presicion= [0.8114334470989761, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Recall= [0.40832975525976817, 0.765142150803461, 0.734375, 0.5021468441391155, 0.18149979398434282, 0.04229282954057535, 0.0098754830399313, 0.6438356
164383562, 0.01051953628166595, 0.012666380420781451, 0.00622584800343495, 0.00794332331472735]
F_Measure= 0.3492410127614332
Entropy= 0.25905530128112064
```

References:

1- http://www.ccs.neu.edu/home/vip/teach/DMcourse/2_cluster_EM_mixt/notes_slides/revisitofrevisitDBSCAN.pdf
2- https://link.springer.com/article/10.1023%2FA%3A1009745219419
3- https://www.datanovia.com/en/lessons/dbscan-density-based-clustering-essentials/
4- https://towardsdatascience.com/machine-learning-clustering-dbscan-determine-the-optimal-value-for-epsilon-eps-python-example-3100091cfbc
5- https://dataminingbook.info/book_html/chap15/book.html