CSCI316 Assignment 1

Minh Nguyen – 5662540 – qmn987

Task 1:

1.1. Preprocessing:

From data in "kddcup.data_10_percent." , we can have some of following preprocessing steps to prepare for SOM:

- First of all, there are two columns: "is_host_login" and "num_outbound_cmds" that only have value 0
  - ⇨ Those two columns would not show anything about problem domain
  - ⇨ We can drop these two columns

- After that, there 3 columns with symbolic values: "protocol_type", "service" and "flag"
  - ⇨ Because SOM training algorithm requires numerical data, we need to preprocess all symbolic columns to numerical
  - ⇨ In order to do that, we use one hot encoding technique because this technique will not only convert symbolic columns to numerical columns but also treat the dataset equally
  - ⇨ In another word, one-hot encoding will convert the values to numerical ones that share the property of equidistance with the original value

- In addition, we can reduce data dimension by removing columns that have strong correlation (which has corr() > 90%) and only keeping one of them.
  - ⇨ Because columns that are strongly correlated would show huge similarities, so assessing those columns would be the same as one representative of them.
  - ⇨ We can take the advantage of strong correlation.

- Moreover, we create new dataset that is used for training purpose
  - ⇨ Because the next task would implement SOM algorithm which has time complexity based on the output dataset, so the size of training set should be appropriate to assure efficiency for task 2
  - ⇨ And the appropriate size is 100,000 samples for training set
  - The training set is just for training purpose but it also needs to be a representative for the whole dataset in order to increase correctness of the algorithm
  - ⇨ That is why I choose 50,000 samples for "normal." class and 50,000 samples for attack classes

- Finally, because SOM is an unsupervised method, so we do not need to have label of column.
  - ⇨ We can do that by exporting the training dataset to file "task1.csv" without header

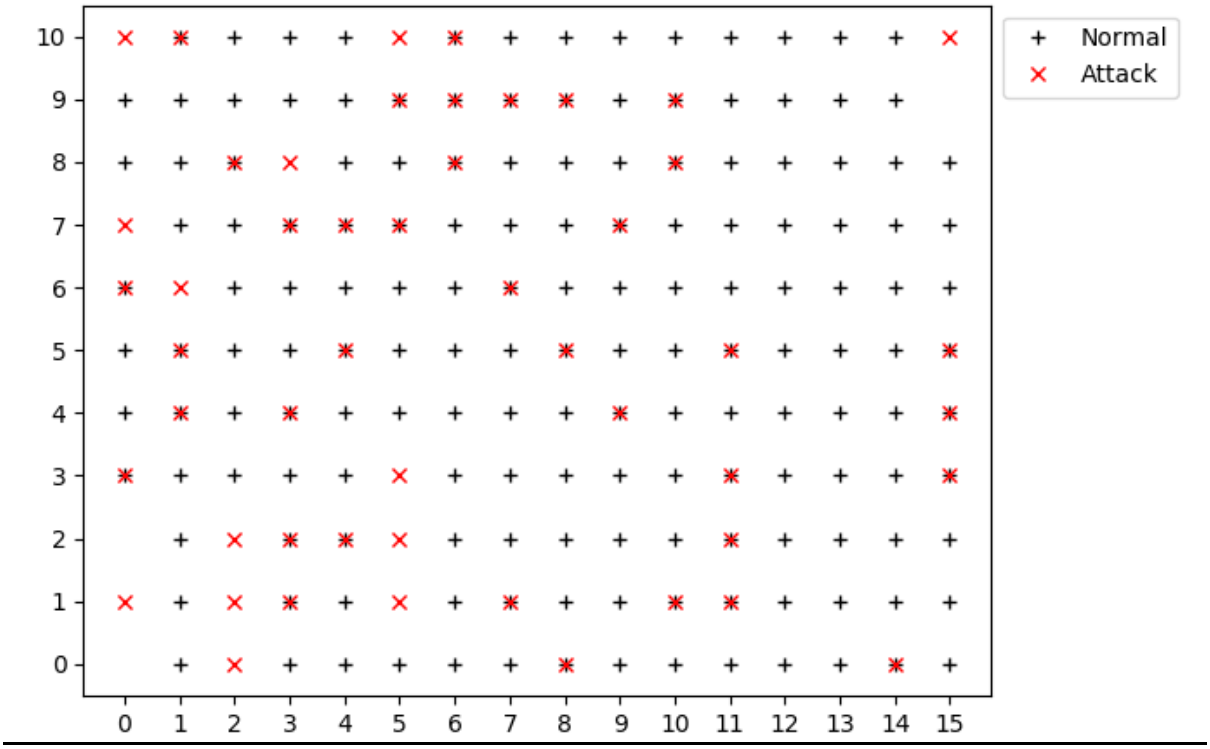
Task 2:

- The largest mapsize that I use to train my SOM is [16, 11]

- Total training time = 3696.6600892543793
  Because average training time for each iteration is 12s
  ⇨ For 300 iterations, total training time would be about 3600s = 60 minutes
  ⇨ Total training time would satisfy requirements.

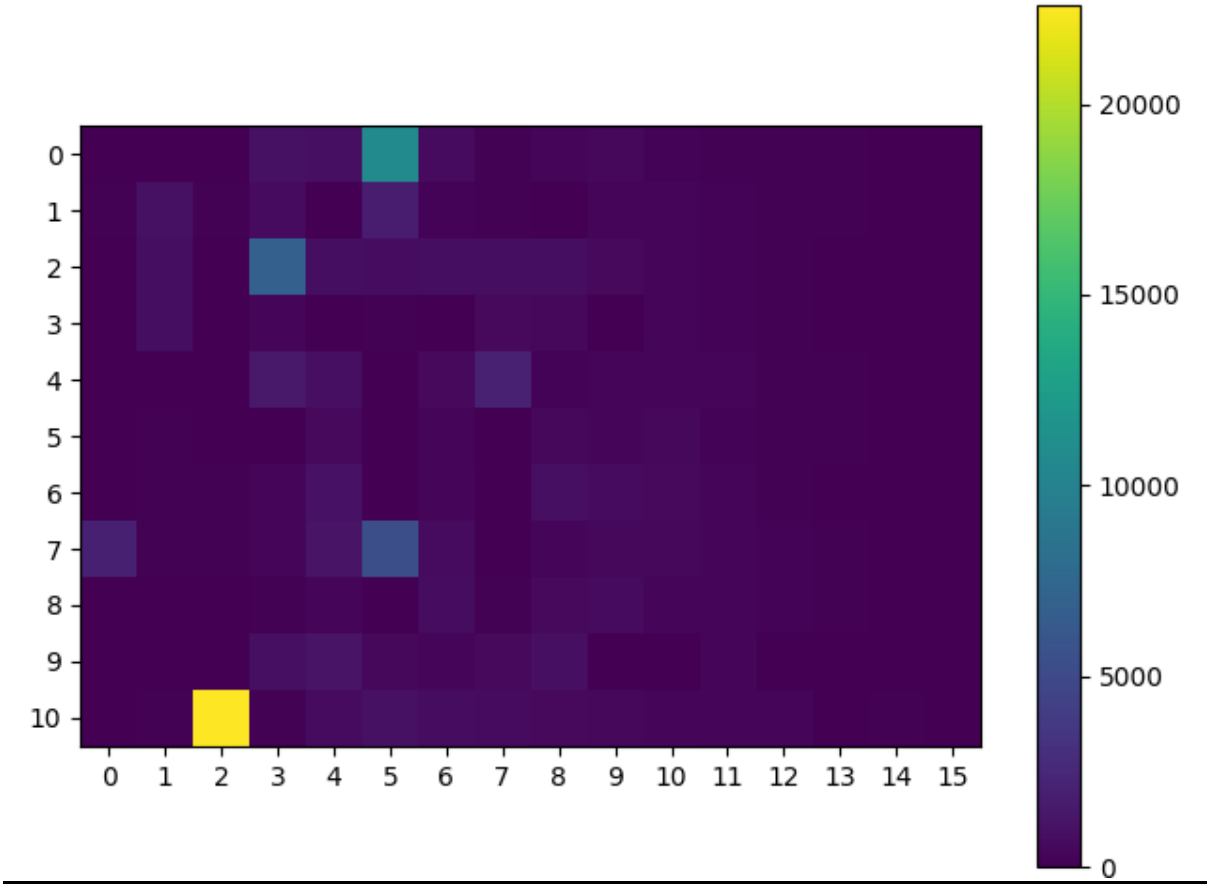
- After training, mean quantization error is 94.90565194099534

1. Visualize the mappings of the training data by class using the symbol '+' for samples that belong to class normal and using the symbol 'X' for classes that belong to one of the attack classes



As can be seen from this visualization, classes that mapped to data does not have any particular pattern and there are some points that is mapped by both classes.
These chaotic mappings may be results of quite small mapsize
However, although mapsize is quite small, there is some blanks where there are no mapped data points.

2. Visualize the mapping density by using a heatmap plot

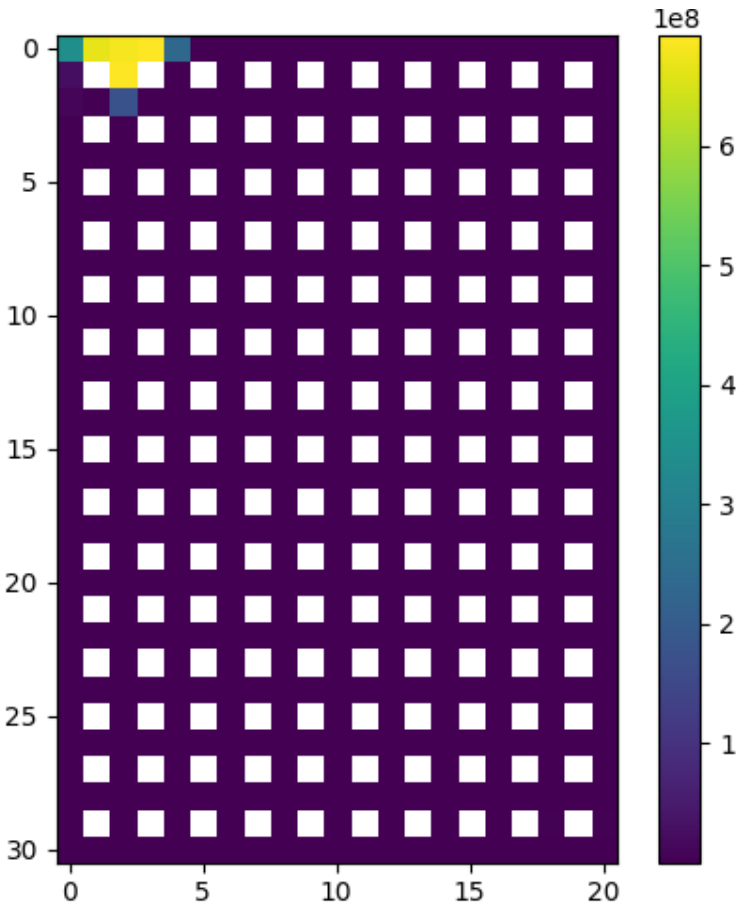


As can be seen from this visualization, density of the whole map is almost low (< 5000). However, there are some exceptions that distinguish from the others:

- There are 2 blue points, which means their densities are slightly bigger than 5000
- There are 1 slight green point, which mean its density are slightly bigger than 10000
- Especially, there is one yellow point, which mean its density is very high (much bigger than 20000)

⇨ A huge number of samples are mapped to this point, which may explain why the density of other points is very low.

3. Visualize the umatrix of the map



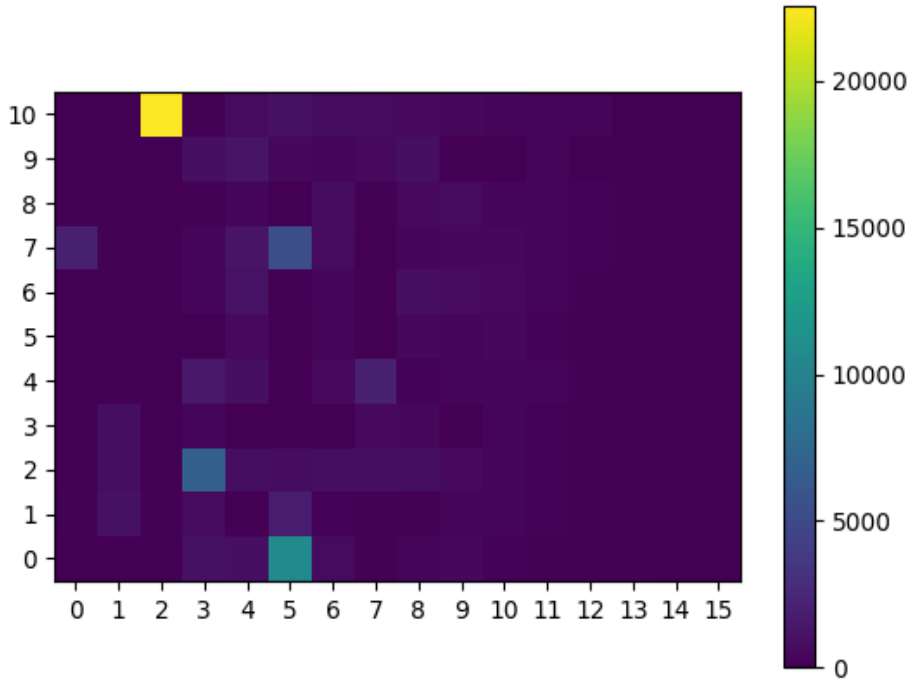
Umatrix visualizes distances between neurons.

- Blank cell will indicate that there is no adjacency
- Other cells will be expressed as colors for the distance

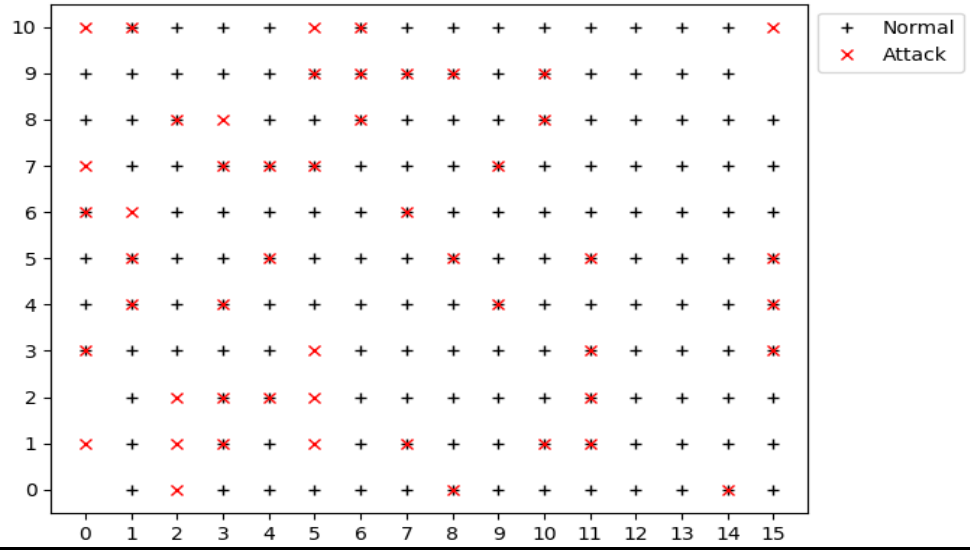
As can be seen from this visualization, almost codebook vectors are close to each other in the input space except a small area that show very large distance among neurons.

4. A visualization technique that offer a great insight into the mapping data is annotated:

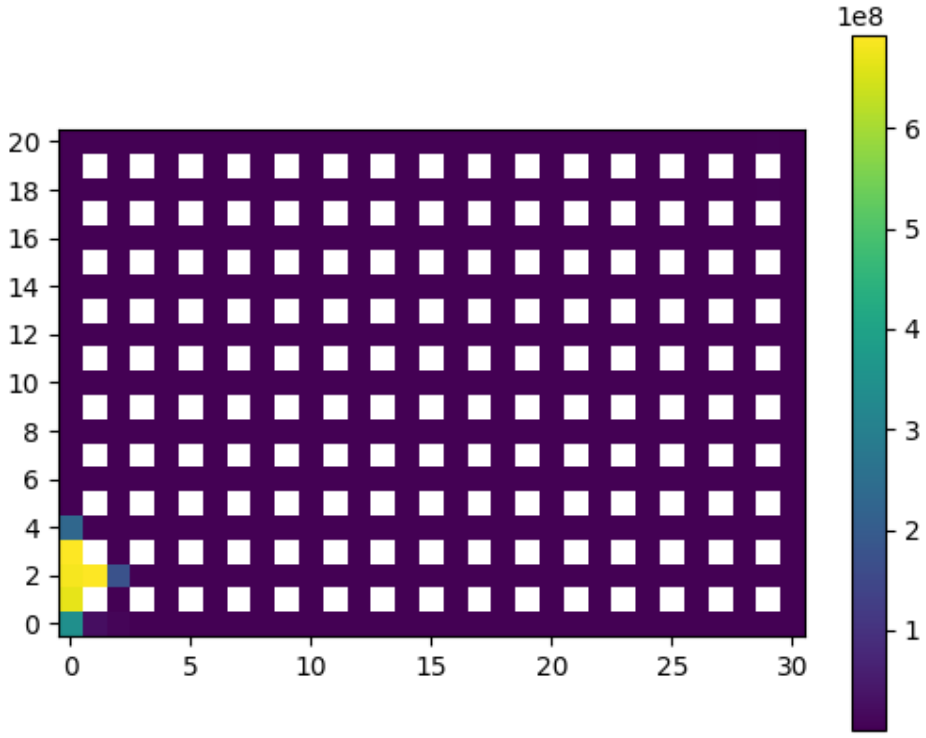
- Annotated density mapping



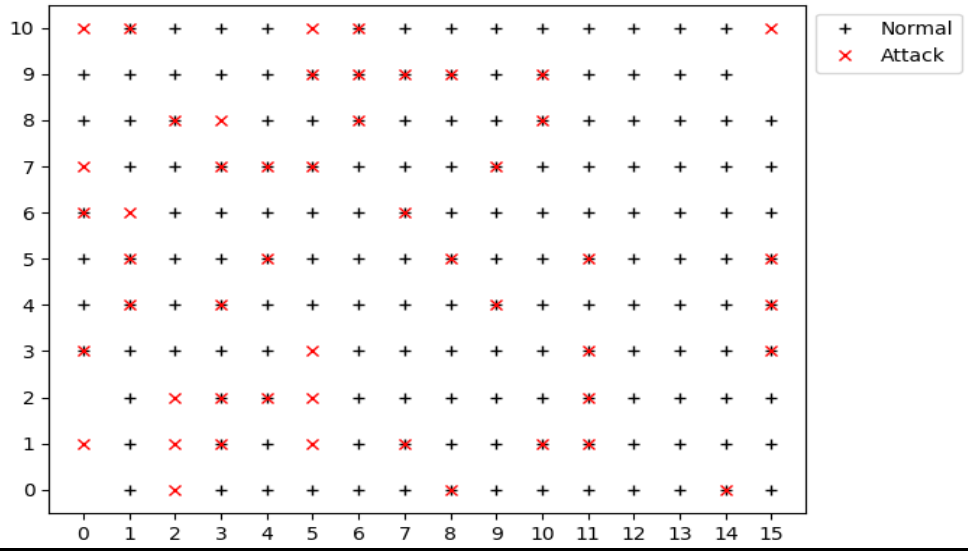
By using this annotated density mapping, we can easily see both density and classes in the mapping points when refer back to class mapping visualization:



- Annotated umatrix mapping:



When refer back to class mappings, we can immediately know why neurons (0,0), (0,1), (0,2) and (1,1) in the class mappings have huge distance between each other.
It is because neurons (0,0) and (0,2) has no mapped data points



5. My approach to implementing SOM in python:

- First of all, instead of using 2D array of codebook vectors, I use 1D array of codebook vectors in order to easily handle matrix multiplication and increase efficiency
- After that, we go to SOM algorithm that for each sample, we have 2 steps: Competitive and Cooperative
- In Competitive step, I take advanges of np.einsum() and argmin() to find index of winning neuron

⇨ Thanks to expressive power and smart loops, np.einsum() would outperform in terms of speed and memory efficiency. Therefore, after calculate difference between each elements in sample vector and corresponding elements in codebook vectors, I used np.einsum() to calculate square of distances between sample and codebooks. Then, we can easily find the index of winning neuron by using argmin()

- In Cooperative step, it is handled more delicate.

⇨ Because in cooperative step of any loop, we always need to calculate h(i,j), which is related to distance between neuron i and neuron j

⇨ Before any training iteration happen, I initialize an 2D-array "r" that r[i, j] is distance between neuron i and neuron j and multiply to a constant that is related to the formula in Gaussian neighborhood function

⇨ Due to this initialization, at the beginning of any iteration, after sigma is updated that is sigma(t), with array r[i][j] , we can immediately calculate:

h(i, j) = e ** (r(i, j) / (sigma(t) * sigma(t))

=> Then, we can calculate alpha(t) * h(i,j)

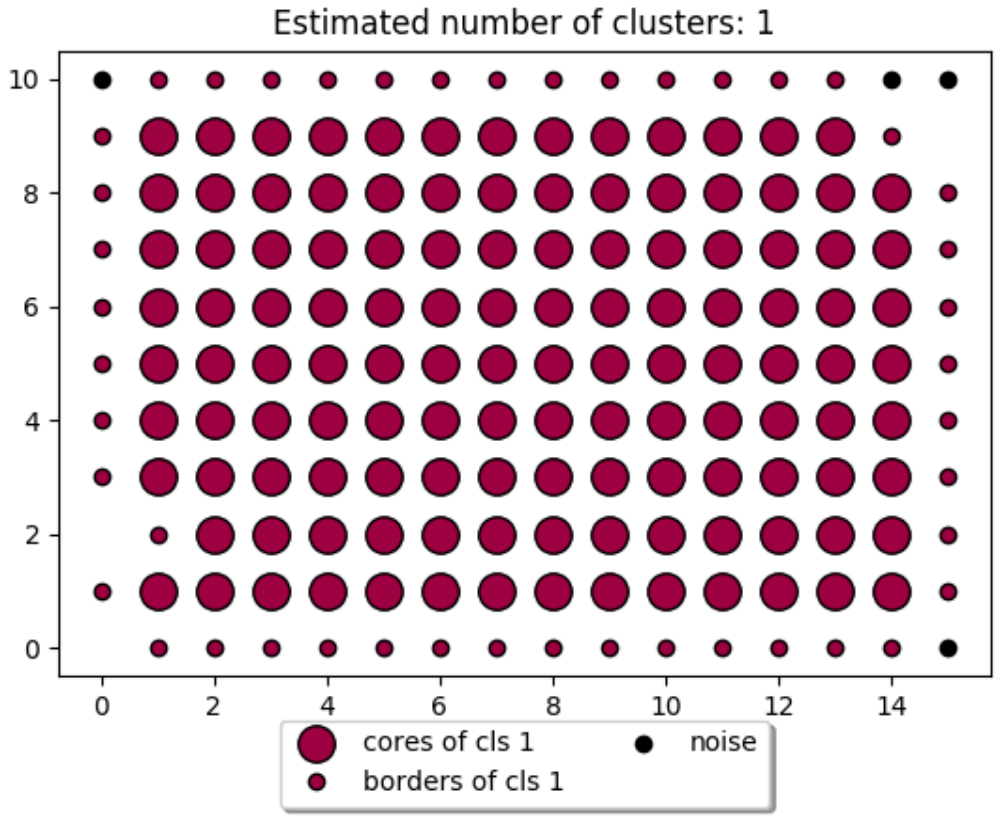
⇨ Now, we only need to take advantages of np.einsum() again to calculate delta, which is codebook(t + 1) – codebook(t)

⇨ Then, we use delta to update codebook vectors

- When we finished updating codebook vectors, we start to calculate quantization errors by using competitive step again to get the winning neuron and calculate distance between them using np.einsum()
- In general, my approach is using np.einsum(), argmin() and initialized r[i,j] to significantly increase speed and efficiency

Task 3:

In this task, if I choose eps = 1 and minPts = 5
The visualization of results would be:



- In this visualization,
  - core points are plotted as big red ones
  - border points are plotted as small red ones
  - noises are plotted as small black ones

- Because data points have integer coordinate, distance between adjacencies = 1

- Because mapsize is small and with eps = 1 = distance between adjacencies and minPts = 5, it is quite easy to cluster

⇨ There is only one cluster cover almost the whole map

- However, there are still some blanks, where there is no data points and there are some noises, where the point is neither core point nor border point