Texas A&M University at Qatar

ECEN 404 - Electrical Design Lab 2

Semester: Spring 2020

Progress Report

Anomaly detection in BACnet protocol systems

Team Members: Sofian Ghazali

Muhammed Zahid Kamil

Rahul Balamurugan

Project Mentors: Dr. Hussein Al Nuweiri

Mr. Salah Hessien

Submission Date: 30/03/2020

"An Aggie does not lie, cheat, or steal, or tolerate those who do."

Abstract

Building Automation Networks/Systems (BAN/BAS) are networks that control and monitor utilities and equipment in smart buildings. The BACnet protocol is one of many communication protocols for smart networks and is widely used as a BAS communication standard. As the number of utilities and buildings connected by such Wide Area Networks increases, the need for robust cyber-physical security measures also increases. Although much research has been done on the subject, constant developments in BAS technology implies similar need for advancement in security as well.

In this project, we propose to emulate a real BAN on Raspberry Pi stacks and generate synthetic network traffic. This will help us in developing an anomaly detection system using unsupervised machine learning technique. The anomalies could be synthetic attacks or malfunctions in BAN services. This shall build upon concepts developed by other researchers in the field, specifically the THE classification of network traffic developed by Zheng et. al, and machine learning techniques by various researchers. Although THE concerns the Time, Human and Event aspect of network traffic, we will only focus on the temporal aspect of our dataset and observe its effect on network data. The end product should be an anomaly detector that is resilient, adaptive, responsive to zero-day attacks, and has a small false-alarm rate. The main focus is on reducing labeling work and improving accuracy by applying clustering techniques to create a unique program that detects novel or unusual data. The network anomaly detector shall be well tested by subjecting the BACnet stack to synthetic attacks.

1. Objectives

The specific list of objectives that we intend to meet are:

1. To measure the accuracy rate of our algorithm and evaluate how well our algorithm works against novel data

To achieve this, we aim to optimize the system to meet a certain threshold accuracy to be deemed fit for use in real-time BAS traffic systems.

The objectives we have met already are:

1. To examine and understand traffic behavior in BAS systems
2. To assess the feasibility of the ML algorithm in traffic network data.
3. To identify methods of classifying and inspecting data packets via different types of ML models.

To address these objectives, we aimed the following tasks which included 1.) Gathered data necessary for the training phase of the algorithm; 2.) Investigated the algorithms that can best model the data behaviour and to achieve an anomaly detector system.

2. Simulation Results, Visual Prototyping & Analysis

2.1 Machine Learning Algorithm

There are many Machine Learning algorithms we decided to give a try. So firstly, we decided on our objectives with regards to the ML algorithm and how it should detect our anomaly. Our objective for ML algorithm was as follows:

1. We have features x1, x2, and x3. We will check if they exhibit any common pattern. We will build a model that will cluster this unlabelled into different groups of similar properties. Once the model is developed, we will visualize the pattern to see which observations are at the boundary level or outside whom the model identified as outliers.

2. The GUI should take input x1, x2, and x3 (exactly the same features that I used to develop the model). Then we run the algorithm to see if they belong to the same cluster or not. The idea is to basically calculate the dissimilarity measure between our input and each of the clusters. If this dissimilarity measure goes above a certain probability score threshold, then the input is classified as an anomaly.

3. Then, we would try to visualize the location of the new observation that the user input on top of the normal data.

From the above algorithms we realized that we needed to use unsupervised learning approaches. This is because we do not have labels to feed into our algorithm, which would in this case be anomalies or not. Therefore, clustering based approaches are very useful in unsupervised learning where it can learn on its own to discover hidden patterns in data[6]. Based on these objectives, our option was inclined towards using the K-Means algorithm as it is an efficient algorithm to work on datasets without labels and will find patterns for us. However, most of our data are categorical in nature, so using the K-means algorithm wasn't feasible since this depended on numerical attributes to calculate the euclidean distance between the values and find the average. Instead, we decided to work with a K-modes algorithm which works well on categorical data. The idea for using K-modes algorithm came to life when a PhD student in mathematics used it to break the okCupid (Dating App) algorithm to get more matches with women [1].

K-modes algorithm works in the following way:

i. Pick an observation at random and use that as cluster
ii. Compare each data point in the cluster centroid to each observation data point, and any elements that are not equal we add a counter of +1 and if they are equal, we keep it as ‘0’
iii. Assign each data point to each closest centroid based on the most least score made in step ii
iv. Update the centroids based on the most common values found from the observation data points.
v. Repeat steps ii-iv until the centroid values converge or doesn’t update.

2.2 Pre-processing

We already had an existing dataset given to us by our mentor. Pre-processing just means cleaning up the data so that the Machine Learning model can make sense of it. This means removing null/missing values, taking out special character values.

Below is the raw Dataset we had to work with:

Time	Source	Destination	Protocol	Length	Info
0.0	10.10.10.43	10.30.10.12	BACnet-APDU	187	Complex-ACK readPropertyMultiple[87]
0.022244	10.30.10.12	10.10.10.44	BACnet-APDU	187	Confirmed-REQ readPropertyMultiple[93]
0.036095999999999996	Invensys_05:6e:9a	Broadcast	ARP	60	Who has 10.30.10.21? Tell 10.30.10.25
0.038743	10.10.10.44	10.30.10.12	BACnet-APDU	261	Complex-ACK readPropertyMultiple[93]
0.09819800000000001	10.30.10.12	10.10.10.61	BACnet-APDU	151	Confirmed-REQ readPropertyMultiple[181]
0.111526	10.10.10.61	10.30.10.12	BACnet-APDU	194	Complex-ACK readPropertyMultiple[181]
0.22218400000000002	10.30.10.12	10.30.10.62	BACnet-APDU	124	Confirmed-REQ readPropertyMultiple[14]
0.233449	10.30.10.62	10.30.10.12	BACnet-APDU	155	Complex-ACK readPropertyMultiple[14]
0.348248	10.30.10.12	10.10.10.54	BACnet-APDU	232	Confirmed-REQ readPropertyMultiple[198]
0.414095	Dell_25:9a:40	Broadcast	ARP	42	Who has 10.30.10.22? Tell 10.30.10.12
0.483965	10.10.10.54	10.30.10.12	BACnet-APDU	311	Complex-ACK readPropertyMultiple[198]
0.487171	10.30.10.12	10.30.10.34	BACnet-APDU	133	Confirmed-REQ readPropertyMultiple[95]
0.505225	10.30.10.12	10.30.10.33	BACnet-APDU	169	Confirmed-REQ readPropertyMultiple[217]
0.518932	10.30.10.34	10.30.10.12	BACnet-APDU	171	Complex-ACK readPropertyMultiple[95]
0.526172	10.30.10.12	10.30.10.31	BACnet-APDU	196	Confirmed-REQ readPropertyMultiple[164]
0.536479	10.30.10.33	10.30.10.12	BACnet-APDU	220	Complex-ACK readPropertyMultiple[217]
0.5450020000000001	10.30.10.31	10.30.10.12	BACnet-APDU	278	Complex-ACK readPropertyMultiple[164]
0.6372140000000001	10.30.10.12	10.30.10.73	BACnet-APDU	232	Confirmed-REQ readPropertyMultiple[24]

Figure 1: Raw Network Traffic Dataset

Time was numerical and continuous and Length was numerical as well, so to make it categorical we grouped it into time intervals. Source, Destination, Protocol and Message are categorical in nature, so in order to better represent the vast array of values within each column, we wanted a way to encode this information in a way the ML can understand. One hot encoding was the technique we used which is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. So categorical is converted to a numerical value[2].

Every column consists of a binary vector of 1s and 0s corresponding to the active and inactive regions of each row, thereby increasing the model's performance. The Length intervals were grouped from 0-100, 100-200 while Time intervals were grouped in 300 second intervals. Code is shown below which shows how we preprocessed.

Code for pre-processing:

In [1]:
```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar 28 15:34:07 2020

@author: ecen
"""


import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from pandas import DataFrame
from kmodes.kmodes import KModes
from kmodes.kprototypes import KPrototypes
import seaborn as sns

dataset= pd.read_csv('Data_SDP_categorical.csv',header =0, index_col=0,parse_dates=True, squeeze=True)

pd.set_option('display.max_columns',227)
pd.set_option('display.max_rows',100)

print(dataset[['Length']].describe())

# Copying dataset
dataset_new = dataset.copy()
# Creating length intervals
dataset['Length_Intervals'] = pd.cut(dataset['Length'], [0, 100,200,300,400,500,600,700,800,900,1000,1100,1200,1300],
                             labels=['0-100', '100-200', '200-300', '300-400','400-500','500-600','600-700',
                                     '700-800','800-900','900-1000','1000-1100','1100-1200','1200-1300'])
```

```
Using TensorFlow backend.
```

Figure 1: Pre-processing Code

Figure 2: Pre-processing Code part 2

	Time_Intervals	Length_Intervals	Source_10.10.10.21	Source_10.10.10.23	Source_10.10.10.24	Source_10.10.10.43	Source_10.10.10.44	Source_1
0	(0, 300]	100-200	0	0	0	0	0	
1	(0, 300]	0-100	0	0	0	0	0	
2	(0, 300]	200-300	0	0	0	0	1	
3	(0, 300]	100-200	0	0	0	0	0	
4	(0, 300]	100-200	0	0	0	0	0	
...	...	...	...	...	...	...	...	
248063	(12000, 12300]	0-100	0	0	0	0	0	
248064	(12000, 12300]	0-100	0	0	0	0	0	
248065	(12000, 12300]	0-100	0	0	0	0	0	
248066	(12300, 12600]	0-100	0	0	0	0	0	
248067	(12300, 12600]	0-100	0	0	0	0	0	

248068 rows × 227 columns

Figure 3: Processed Dataset

2.3 Data Analysis

To understand the nature of our data and how it behaves, we performed a few data visualizations such as pairwise plotting and time-series plotting. As shown in Figure 5 below, some pairwise plots showed some interesting characteristics about the data and how each message type behaved with respect to the 5 features: Time, Source IP, Destination IP, Message Length, and Protocol. Some messages exhibited recurring patterns with a few IP addresses only and this was interesting. It showed how some IP addresses only share a certain range of messages. We also wanted to

see upclose how data behaved for time. Figure 4 shows a periodic data pattern and this prompted us to learn more about this time pattern.

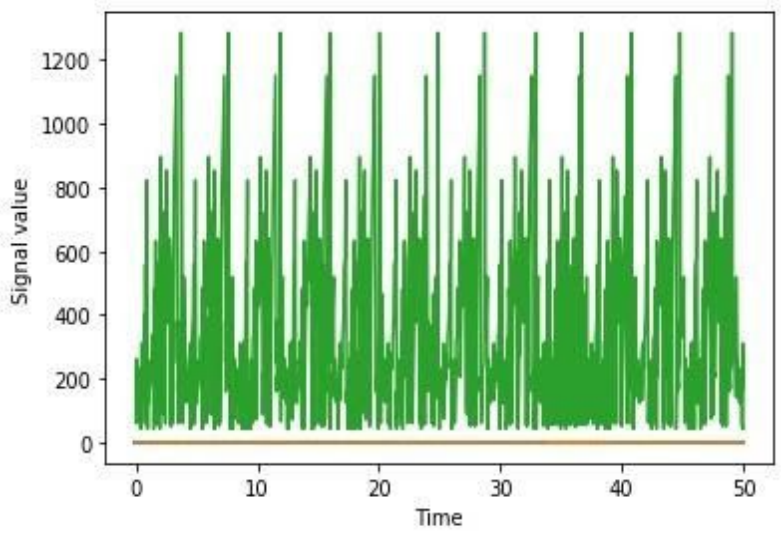


Figure 4: Time-Analysis of data

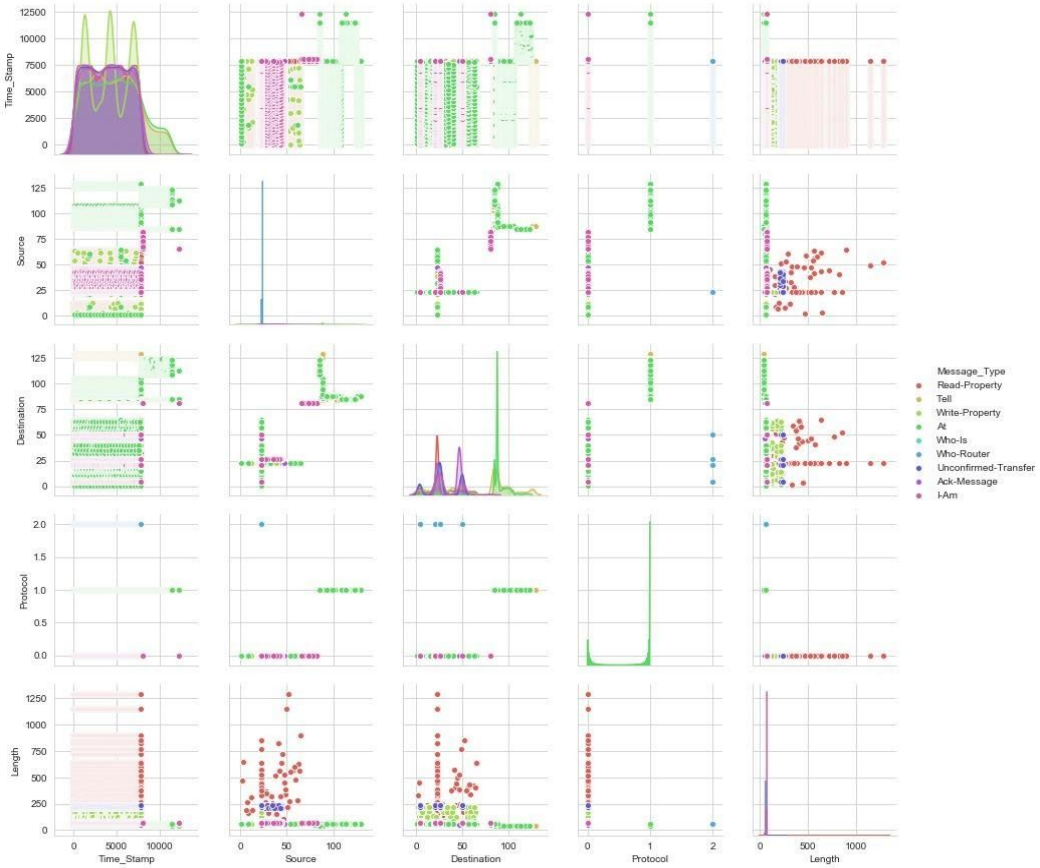


Figure 5: Pairwise Plot to showcase relationship between the variables

2.4 Flowchart of K - Mode Algorithm

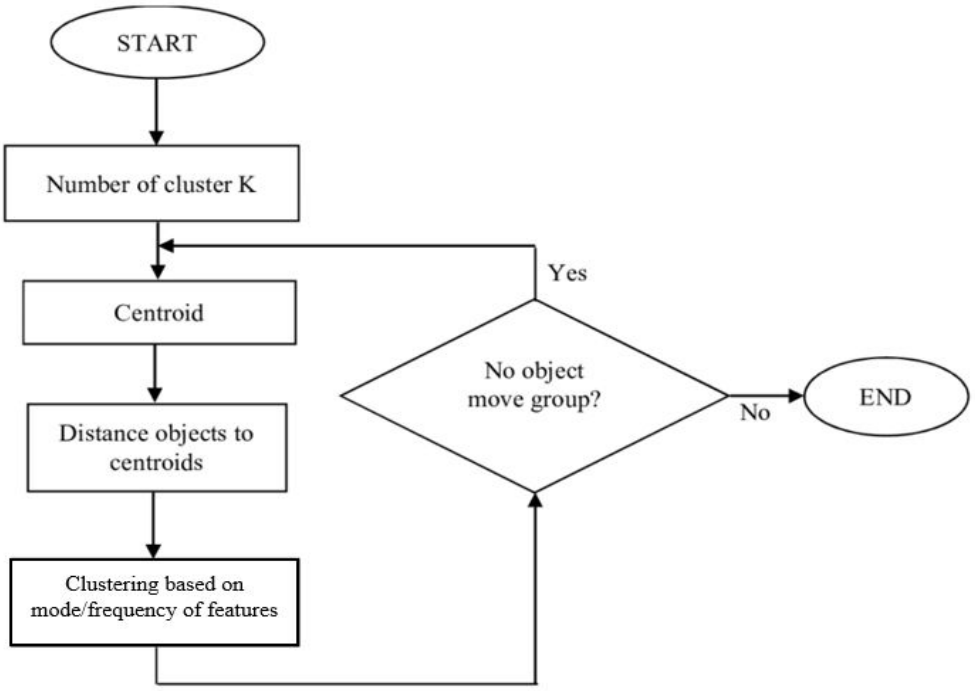


Figure 6: Pairwise Plot to showcase relationship between the variables

2.5 K-Mode Algorithm and Analysis

Choosing the number of clusters either 5,10,15,20

Specify random_state_no - If int, random_state is the seed used by the random number generator

init_no - Number of time the k-modes algorithm will be run with different centroid seeds.

##The final results will be the best output of n_init consecutive runs in terms of cost.

init_method - Type of algorithm, mostly either Cao (Method in Cao et al. [2009]) or Huang (Method in Huang [1997, 1998])

Max_iter = 100 is Maximum number of iterations of the k-modes algorithm for a single run.

```
In [4]: def makeClusters(data, num_clusters,max_iter, init_method, init_no, random_state_no):

            km_cao = KModes(n_clusters= num_clusters, init = init_method, n_init = init_no, verbose=1, random_state
            data = Finalized_Data.values # or Finalized Data not sure?
            fitClusters_cao = km_cao.fit_predict(data)

            clusterCentroidsDf = pd.DataFrame(km_cao.cluster_centroids_)
            clusForInfo = clusterCentroidsDf.copy()
            clusterInfo = clusForInfo
            clusterInfo.columns = Finalized_Data.columns #clusterInfo shows the cluster information

            # Concatenating the data for visualization
            clustersDf = pd.DataFrame(fitClusters_cao)
            clustersDf.columns = ['cluster_predicted']
            combinedDf = pd.concat([Finalized_Data, clustersDf], axis = 1).reset_index()

            return clusterInfo, clusterCentroidsDf, combinedDf, fitClusters_cao
```

```
In [26]: def plotMessages(data):

             f, axs = plt.subplots(4,3,figsize = (15,20))

             sns.countplot(x=data['Messages_Who-Is'],order=data['Messages_Who-Is'].value_counts().index,hue=data['cl
             sns.countplot(x=data['Messages_Read-Property'],order=data['Messages_Read-Property'].value_counts().inde
             sns.countplot(x=data['Messages_I-Am'],order=data['Messages_I-Am'].value_counts().index,hue=data['cluste
             sns.countplot(x=data['Messages_Ack-Message'],order=data['Messages_Ack-Message'].value_counts().index,hu
             sns.countplot(x=data['Protocol_BACnet-APDU'],order=data['Protocol_BACnet-APDU'].value_counts().index,hu
             sns.countplot(x=data['Protocol_ARP'],order=data['Protocol_ARP'].value_counts().index,hue=data['cluster_
             sns.countplot(x=data['Protocol_BACnet-NPDU'],order=data['Protocol_BACnet-NPDU'].value_counts().index,hu
             sns.countplot(x=data['Messages_Write-Property'],order=data['Messages_Write-Property'].value_counts().in
             sns.countplot(x=data['Messages_Who-Router'],order=data['Messages_Who-Router'].value_counts().index,hue=
             sns.countplot(x=data['Messages_Unconfirmed-Transfer'],order=data['Messages_Unconfirmed-Transfer'].value_
             sns.countplot(x=data['Messages_At'],order=data['Messages_At'].value_counts().index,hue=data['cluster_pr
             sns.countplot(x=data['Messages_Tell'],order=data['Messages_Tell'].value_counts().index,hue=data['cluste

             plt.tight_layout()
             plt.show()
```

```
In [29]: clusterInfo_12, clusterCentroidsDf_12, combinedDf_12, fitClusters_cao_12 = makeClusters(data = Finalized_Data,
                                                                                                num_clusters = 12,
                                                                                                max_iter = 100,
                                                                                                init_method = 'Cao',
                                                                                                init_no = 1,
                                                                                                random_state_no = 22)
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 21106, cost: 651987.0
Run 1, iteration: 2/100, moves: 2218, cost: 651987.0
```

Figure 7: Code for K-Mode Algorithm

Now the issue we ran into was deciding on the number of clusters that will best capture the data. If we were to try every cluster size by trial and error, this will be time consuming and often a cumbersome process. Assigning too many clusters will render the clustering process futile since there will be overlap of centroids and discernable patterns will be found. Setting cluster size too low will result in

misrepresentation of data. The best option is to make use of the Elbow Method[5]. Usually, in clustering, the distance between the data points in the clusters and the centroid is often referred to as the 'Within Cluster Sum of Squares' (WCSS). We basically want to minimize this distance while making sure we get a reasonable amount of clusters. This helped us in determining the right number of clusters as shown below which was from 3-5 clusters. Of course, this cluster decision is going to change since we are going to investigate the right number of clusters by tweaking it in addition to what the elbow method showed us. Below is figure 8 which shows the elbow method graph we generated. In the figure below, the x-axis represents the number of clusters and y-axis represents the WCSS value. So, it looks the right distance between centroids and cluster points and the respective cluster amounts that can be formed. The tiny bump at x=3,4 indicates to us that clusters above this value and until 6 are optimal. However, these clusters are not suitable

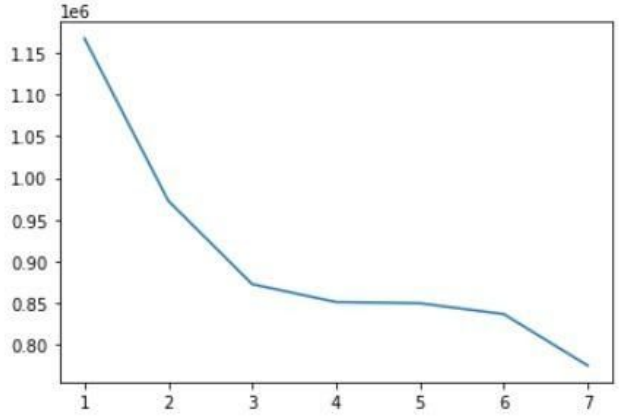


Figure 8: Elbow Method for optimal clusters

So, we run the algorithm once to find out optimal clusters using the above method. Then we use the same algorithm to run using the cluster number obtained from first run to observe if those clusters capture the data patterns. Figure 9 below shows the results of running the k-mode algorithm. We made two plots by placing a message and an IP address input. The K-mode gives us a cluster plot in terms of graph to showcase which cluster the inputs will belong to. The y-axis shows the frequency of occurrences with the respective clusters and the x-axis shows the binary output of those messages.

(Write here about silhouette and the optimal cluster found)

```
In [133]: f, axs = plt.subplots(1,2,figsize = (15,5))

sns.countplot(x=combinedDf_10['Messages_Tell'],order=combinedDf_10['Messages_Tell'].value_counts().index,hue=combinedDf['clus
sns.countplot(x=combinedDf_10['Source_10.10.10.23'],order=combinedDf_10['Source_10.10.10.23'].value_counts().index,hue=combin

plt.tight_layout()
plt.show()
print(axs[0])
```

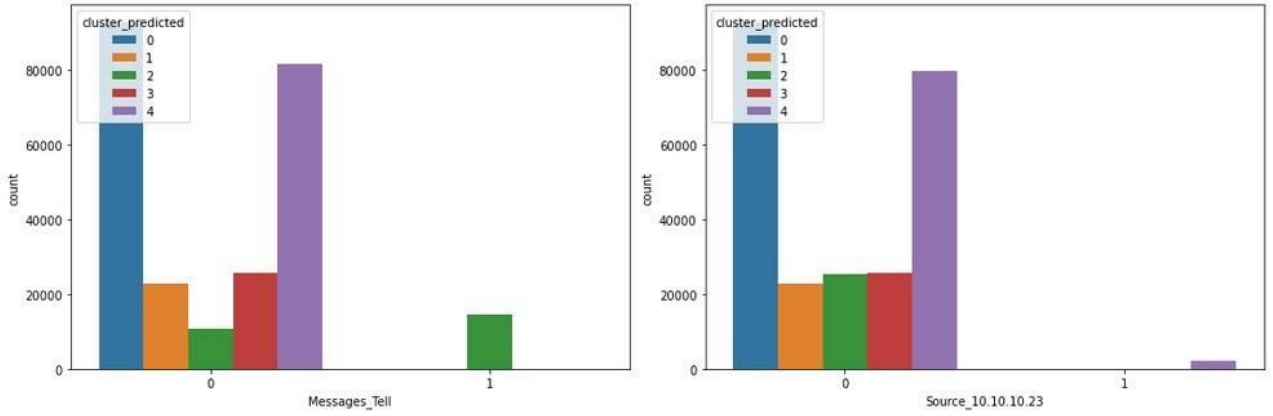


Figure 9: Cluster Plot

3. Functional Prototyping, testing, troubleshooting

3.1 Troubleshooting and Issues

First issue that we'd like to point is regarding data generation. Our second objective related to data generation which was in addition to the main objective of doing ML for our network traffic. We faced some issues related to the second objective since we couldn't bring all the equipment from the lab due to the recent closure. Our project mentor, Dr. Hussein Al-Nuweiri, recommended to us that we only focus on the ML aspect of the project with the existing dataset he gave us. We also ran into quite a lot of issues regarding our algorithm since most of the time we couldn't make sense of what the algorithm is giving as an output. Initially, we relied on Neural Networks, Recurrent Neural Networks in particular, but we didn't proceed with our algorithm since it required a lot more datasets with more features to be able to extract useful information. We ran mostly into overfitting problems in the case of Neural Networks, and hence we omitted it. We wanted to make sense of our data since our project is on detecting anomalies but our data was perfectly normal. Our choice was K-Mode and it served our needs well. Since it trains on normal data and clusters it according to the centroids, any dissimilarity between the input and the centroids will be highlighted on the cluster graph; the anomalous point will appear farther away from the centroids.

3.2 Testing the Model

To test the k-modes model, we ran the model using the combined data from the QU dataset and our own collected data from the emulated BAS. The clusters generated are shown below:

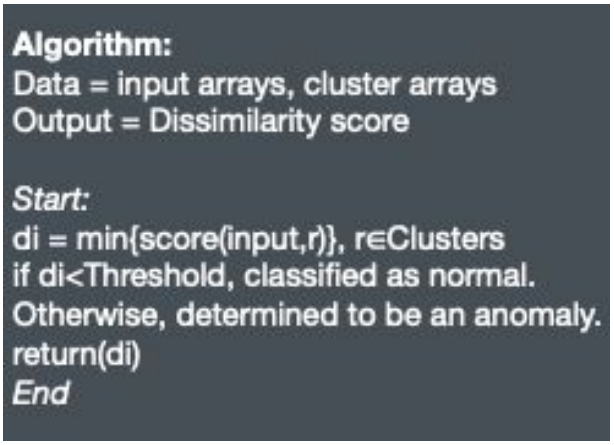
	Protocol_ARP	Protocol_BACnet-APDU	Protocol_BACnet-NPDU	Messages_Ack-Message	Messages_At	Messages_I-Am	Messages_Read-Property	Messages_Tell	Messages_Who-Is	Messages_R
0	0	1	0	0	0	0	1	0	0	
1	0	0	1	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	1	0	
3	0	1	0	0	0	0	0	0	0	
4	0	1	0	0	0	0	0	0	0	
5	0	1	0	0	0	1	0	0	0	
6	1	0	0	0	1	0	0	0	0	
7	0	1	0	0	0	0	1	0	0	
8	0	1	0	0	0	0	0	0	0	
9	0	1	0	0	0	1	0	0	0	
10	0	1	0	0	0	0	0	0	0	
11	0	1	0	0	0	0	0	0	1	

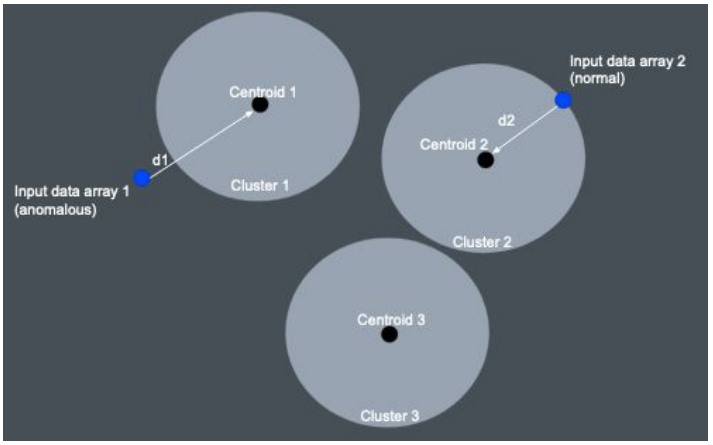
Figure 10: Feature count in each cluster

The Number of occurrences (count) of each category in each cluster was plotted for the model. From this plot, we could observe for which number of clusters there was lesser repetition and thus more data coherence. To get the result deterministically, we planned to use the Jaccard Similarity Index since our dataset is categorical in nature. The Jaccard index is a simple ratio between the number of intersecting elements in two sets and the number of elements in the union of the two[4] as shown in the formula below:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

Figure 11: Jaccard-Needham Dissimilarity Formula





This can help us identify two things: 1) the most important features, and 2) which number of clusters is most efficient or accurate in representing our data. Other methods of doing the same exist, however we will only try those if the Jaccard Index shows conflicting results. We were warned that might happen if the data sample size was too small, since the Index is highly sensitive.

To test the anomaly detection capability of the model, we must first identify the most important features. Then, we can set the threshold on the cost function, which in our case is the distance between the farthest outlier and the centroid of the cluster. Any data point beyond the threshold would be classified as an anomaly.

Figure 13 above shows simplistically how the algorithm works. The blue dots are the new inputs, while the three cluster centroids represent the entire data in this figure. The dissimilarity score for an input is the distance of input from the closest centroid. Training the detector over our normal dataset sets the threshold, which is the boundary of the cluster. The algorithm decides it is an anomaly if the dissimilarity is greater than the threshold. Figure 12 shows the code we implemented to perform the Jaccard dissimilarity scoring.

3.3 Threshold settings

Setting the threshold is the same thing as training the detector using our existing normal dataset. The maximum score for our dataset was chosen as the threshold. The table shows the various distance/dissimilarity scores for each row in the normal dataset. As mentioned previously, we chose the Jaccard score as it has been known to work very well for categorical data. For this measure, our threshold was found to be 0.45.

Table 1: Anomaly Scores

	Jaccard	Dice	Hamming	Rogers-Tanimoto	Sokal-Michener	Sokal-Sneath	Yule
0	0.166667	0.0909091	0.0526316	0.1	0.1	0.285714	0.00759013
1	0.125	0.0666667	0.0394737	0.0759494	0.0759494	0.222222	0
2	0.0588235	0.030303	0.0131579	0.025974	0.025974	0.111111	0
3	0.0454545	0.0232558	0.0131579	0.025974	0.025974	0.0869565	0
4	0.12	0.0638298	0.0394737	0.0759494	0.0759494	0.214286	0.00355872
...	...	...	...	...	...	...	...
248063	0.32	0.190476	0.105263	0.190476	0.190476	0.484848	0.0340136
248064	0.32	0.190476	0.105263	0.190476	0.190476	0.484848	0.0340136
248065	0.384615	0.238095	0.131579	0.232558	0.232558	0.555556	0.0534351
248066	0.25	0.142857	0.0789474	0.146341	0.146341	0.4	0.0169492
248067	0.32	0.190476	0.105263	0.190476	0.190476	0.484848	0.0340136

248068 rows × 7 columns

Each of the scoring functions above different ways of finding the dissimilarity in the data:

Jaccard:
Dice:
Hamming:
Rogers-Tanimoto:
Sokal-Michener:
Sokal-Sneath
Yule:

Chosen Scoring function and threshold:
Jaccard-Needham Dissimilarity: 0.45 (**Any score above a score of (Jaccard, 0.45) is determined to be an anomaly.)**

3.4 Model Testing

We began making our own anomalous dataset since we didn't have any anomalies with our actual dataset. This step will be essential in order to detect our anomalies and classify them as ‘anomaly’ or not. We used the threshold we obtained from Jaccard similarity when training on the normal dataset. This threshold was then applied to the anomalous dataset; when any datapoint goes above the threshold value of 0.45, the data is classified as an anomaly. To generate the abnormal dataset, new IP addresses were generated by scrambling existing normal dataset.
We even scrambled all the protocol and message columns by randomizing it. The purpose of randomizing it is that the anomalous data packets may include some instances of normal data as well so that there’s a good mix of two datasets, but overall, the dataset will be considered anomalous

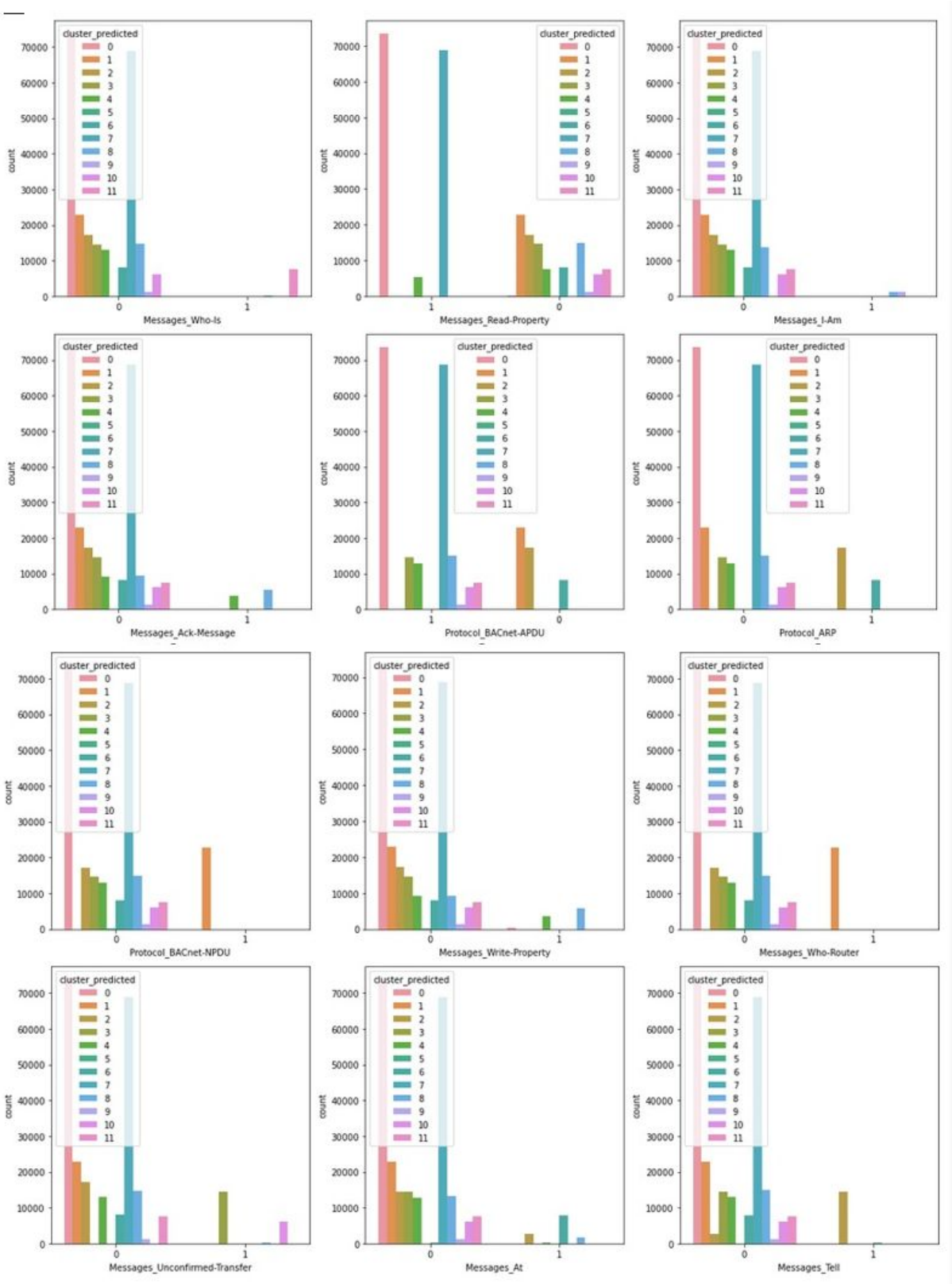


Figure: Plot of Feature counts in each cluster

4. Progress

The timeline of the project since its inception in August to its current progress (blue triangle marker) is shown in the figure above.

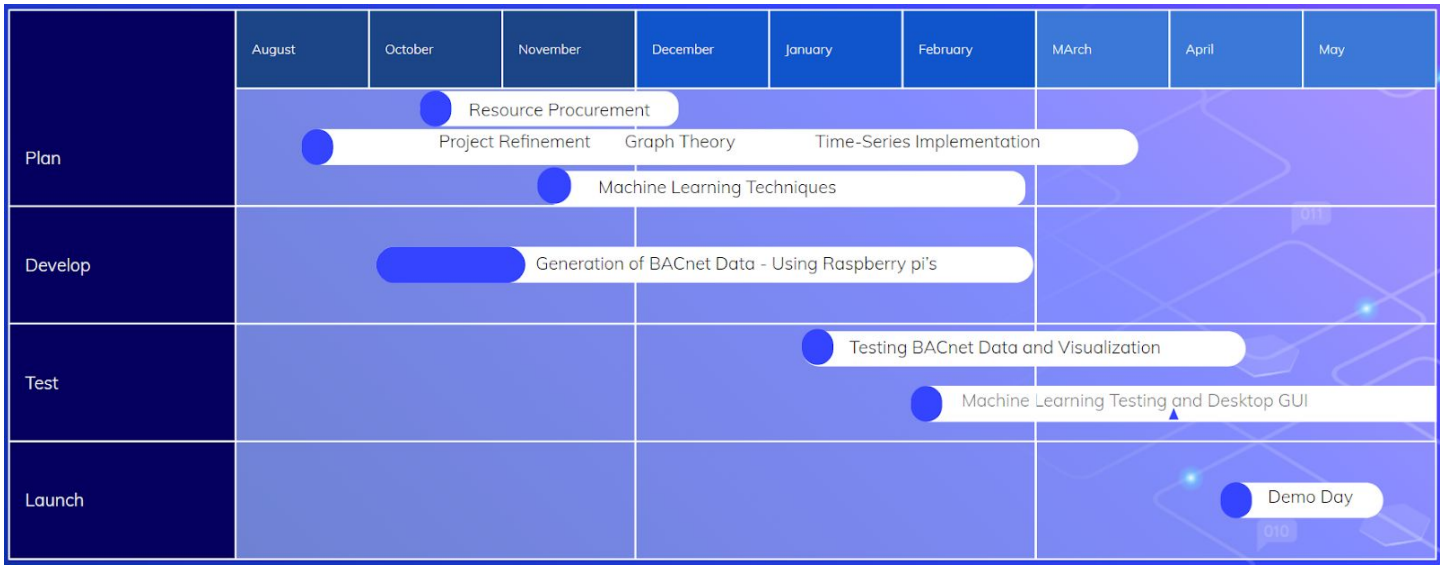


Figure: Timeline of the project

5. Conclusion

Despite the hiccups caused by social distancing, the project is close to completion. The focus of the project, however, changed from emulating a BACnet system using Raspberry Pis to training a machine learning model. The K-modes algorithm was proved to be an effective method to train a detector for comparatively smaller datasets (~250,000 data points). In addition, a prototype GUI was created to allow users to track anomalies and generate BACnet protocol messages to test the model. With only the actual detection part and model testing remaining, we hope to reach our minimum goal in a week or so.

References

[1] K. Poulsen, “How a Math Genius Hacked OkCupid to Find True Love,” *Wired*, 25-Sep-2018. [Online]. Available: https://www.wired.com/2014/01/how-to-hack-okcupid/. [Accessed: 30-Mar-2020].

[2] D. Masse, "Unsupervised Learning for Categorical Data", *Medium*, 2020. [Online]. Available: https://medium.com/@davidmasse8/unsupervised-learning-for-categorical-data-dd7e497033ae. [Accessed: 26- Mar- 2020]

[3] N. Vos, "nicodv/kmodes", *GitHub*, 2020. [Online]. Available: https://github.com/nicodv/kmodes/tree/master/kmodes. [Accessed: 23- Mar- 2020]

[4] R. Joseph, "Jaccard Similarity: The Jaccard similarity of sets is the ratio of the size of the intersection of the sets to the size of the union. This measure of similarity is suitable for many applications, including textual similarity of documents and similarity of buying habits of customers.", *Github*, 2020. [Online]. Available: https://gist.github.com/Renien/9672f174e31b6f96f356da09eb481d2c. [Accessed: 28- Mar- 2020]

[5] D. Shapiro, "Elbow Clustering for Artificial Intelligence", *Medium*, 2020. [Online]. Available: https://towardsdatascience.com/elbow-clustering-for-artificial-intelligence-be9c641d9cf8. [Accessed: 27- Mar- 2020]

[6] S. Mishra, "Unsupervised Learning and Data Clustering", *Medium*, 2020. [Online]. Available: https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a. [Accessed: 24- Mar- 2020]