Cyber Data Analytics Lab 3

Xin Li 4721101
Po-Shin Chen 4703308

June 24, 2018

1 Sampling Task

In this section, we use CTU-Botnet-46(scenario 5) with 455540 rows for sampling task, which has fewer observation than others. We tend to know what IPs the infected host connected to. After filtering out all of the hosts, the 10 most frequent IPs that were connected by the infected host with the percentages are shown in Table 1.

Connected hosts	**# of connections**	**Percentage(%)**
212.117.171.138	**381**	**0.22**
46.4.36.120	**275**	**0.13**
147.32.80.9	**46**	**0.06**
64.4.2.109	**42**	**0.05**
65.55.72.7	**34**	**0.05**
65.55.40.23	**32**	**0.03**
91.220.0.52	**28**	**0.03**
65.54.186.10	**26**	**0.03**
94.63.150.52	**26**	**0.03**
209.85.148.147	**24**	**0.02**

Table 1: Top 10 frequent IPs in CTU-Botnet-46

Next, we use min-wise sampling method to see whether the sampling result is similar to the real distribution. First we uniformly sampled the IPs into a reservoir with size 10. We then generate a random value $h \in [0, 1]$ for each row(therefore, there will be 455540 numbers) and for the first 11 rows, we select 10 items with the lowest random values into the reservoir and remove the highest one. Next, the coming row was added and was compared to those 10 items again. By looping through all of the rows, we are able to get an estimated distribution of the IPs.

As a result, the performance is shown as below. It is easy to notice that the distribution is quite different from the true distribution when the reservoir size(K) is small and approaches to the true distribution when the reservoir size increases. The reason of this result is that we randomly assign a probability to each row and then the IPs with higher frequency would get higher value. However, it is possible that a IP with lower frequency gets a randomly high value. In addition, except for the first two frequent IPs, the other IPs have similar percentage as shown in Table. 1, so they couldn't increase the chance of getting higher value based on their frequency. The result will turn out to be random.

Connected hosts	**K = 100**	**Connected hosts**	**K = 400**	**Connected hosts**	**K = 1000**
212.117.171.138	0.22	**212.117.171.138**	0.2325	**212.117.171.138**	0.252
46.4.36.120	0.13	**46.4.36.120**	0.195	**46.4.36.120**	0.204
64.4.2.109	0.06	**64.4.2.109**	0.0375	**147.32.80.9**	0.033
65.55.40.23	0.05	**147.32.80.9**	0.035	**64.4.2.109**	0.028
60.190.223.75	0.05	**65.55.72.7**	0.0275	**65.55.72.7**	0.027
65.54.234.78	0.03	**74.3.164.224**	0.025	**65.55.40.23**	0.024
74.3.164.223	0.03	**65.54.186.10**	0.0225	**65.54.186.10**	0.023
147.32.80.9	0.03	**65.55.40.23**	0.02	**94.63.150.52**	0.022
94.63.149.150	0.03	**94.63.150.52**	0.02	**209.85.148.147**	0.02
94.63.150.52	0.02	**65.55.75.231**	0.0175	**74.3.164.224**	0.019

Table 2: Distribution after min-wise sampling

2 Sketching task

In this section, we implement Count-min sketch method to calculate the frequency. A 3×1000 frequency table are created and each row corresponds to each different hashing method. Here we implement md5, sha1, sha224 for hashing. Each IP will be hashed into three values from the above hash functions and the counts of the corresponding coordinations in the table from the returned value will increase. When retrieving the frequency from an IP, the lowest count among three corresponded counts from the hashed coordinates is returned. Since there are three various hashing functions, it overcomes the issue of overcounting the IPs due to collisions and the result would be precise.

The goal of the count–min sketch is to consume a stream of events, one at a time, and count the frequency of the different types of events in the stream. Furthermore, the sketch can be queried for the frequency anytime. As as result shown in Table 3, we can see that the distribution is the same as the true distribution and the counted value is the same as well because of the multi-implementations of the hashing functions.

As for the computational time, we compute the time from creating the table to printing the results. For the min-wise sampling, the elapsed time is 0.349 - 0.576 second depending on the reservoir size. However, it only takes 0.03 second for count-min sketch method. Therefore, we can also prove that the count-min sketch method is not only accurate but also faster than min-wise sampling.

Connected hosts	**# of connections(Count-min-sketch)**	**# of connections(True)**
212.117.171.138	**381**	**381**
46.4.36.120	**275**	**275**
64.4.2.109	**46**	**46**
147.32.84.138	**42**	**42**
65.55.72.7	**34**	**34**
65.55.40.23	**32**	**32**
91.220.0.52	**28**	**28**
65.54.186.10	**26**	**26**
94.63.150.52	**26**	**26**
209.85.148.147	**24**	**24**

Table 3: Performance of Count-min sketch

3 Botnet flow data discretization task

In this section, we select two features believed as relevant for discretization, which are *Protocol* and $Packets_Bytes$. First, the protocol distribution of an infected host and a normal host is compared in Fig. 1. As we can notice, in infected hosts, there might be relatively more ICMP protocol existed in the data and there might be more TCP protocol in a normal host. Second, for the $Packets_Byte$, we change this quantitative data into qualitative data by setting the regions for the packets(0 if p $<$ 1000, 1 if 1000$<$ p $<$5000, 2 if 5000$<$ p $<$10000,3 else). After transforming them into qualitative data, we plot the distribution as shown in Fig. 2. We are able to see that for the infected host, the value of $packets_Byte$ tends to stay in region 1, and region 0 for the normal host. Thus, it is trust worthy that the infected host and be distinguished through the encoding.

We implement attribute mappings for Netflow encoding. Since we selected *Protocol* and $Packets_Byte$ as features, we will implement both quantitative mapping method and qualitative mapping method. As described in the paper[Pel17], we factorize the categories into 0,1,2 and $|M_i| = 3$. As fort the qualitative feature($Packets_Byte$), we assume that we are interested in 20% and 80% percentiles, then we categorize the values to the corresponding regions. However, it is interested that for our selected infected host, the corresponded 20% and 80% values are the same(1066). Therefore, we only categorize the data into two values(0 if p$<$1066 and 1 if p$>=$ 1066,

which means $|M_j| = 2$).

Next, we apply the encoding formula according to the paper[Pel17] : $code = P * |M_j| + B$, where P,B indicates the quantified value of $Prot$ and $Packets_Byte$, individually. The distribution of the encoding results in each non-infected/infected host is displayed in Fig. 3. It is clear that we are able to discrete the 5 normal hosts and 10 infected provided by the website based on these encoding values.

We classified the hosts as infected hosts if there exists a high-frequency behavior on encoding 5. As a result, for the known hosts, we got zero FP and FN, as shown in Fig. 4a. However, the performance of this encoding method into all hosts in scenario 10 is not good enough(Fig.4b). There are two reasons behind this phenomenon. First is that there are lots of Botnets having few behaviors(lower than 50), thus we don't have enough data for observations. Second, since there are only 10 hosts in Botnet are known as infected hosts, we could not assure whether those Botnets classified as non-infected hosts are really infected or not. These reasons will increase the uncertainty and make us hard for classification.

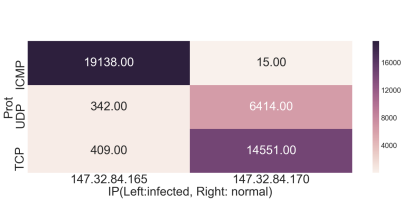


Figure 1: Descritization based on the *protocol*

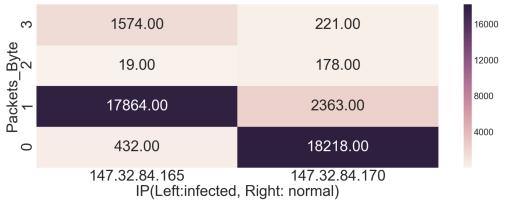


Figure 2: Descritization based on the $Packets_Byte$

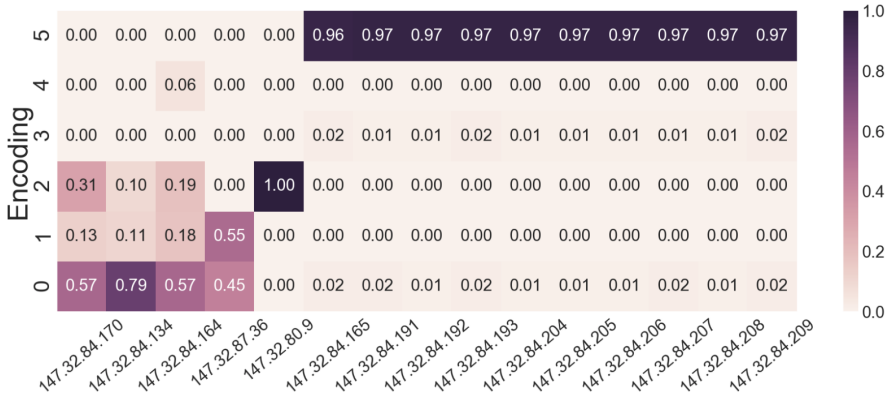


Figure 3: Descritization after combining *protocol* and $Packets_Byte$)

Actual \ Classified	**Infected**	**Normal**
Infected	10	0
Normal	0	5

(a) Detection performance of infected hosts and normal hosts

Actual \ Classified	**Botnet**	**Normal**
Botnet	11	56
Normal	0	455

(b) Detection performance of Botnets and Legitimates hosts

Figure 4: Detection result of the encoding algorithm

4 Botnet profiling task

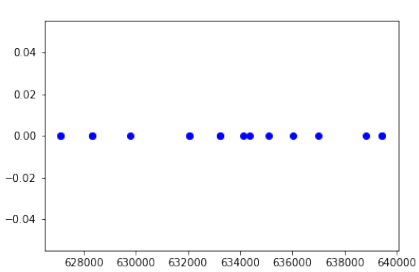
For the profiling task, we use the HMM model. The dataset is based on the result of previsous section 3. And the dataset is grouped by the source IP address. In order to obatain a sequential data, the sliding window method has been applied to the data of each source IP address seperately. Empirically, we choose the value of window size as 4. If the last sequence produced by sliding window is shorter than the window size, it wil be deleted for the format of HMM input.

For the training of Gaussian HMM model of 10 states, the sequential data of source IP address 147.32.84.165 is applied. The number of states is choosen empirically . After that, we can use the training data samples, which include 9 botnets and 5 normal IPs, and compute their log probability under the Based on the log probability of these samples, a threshold can be found. In this case, we found $threshold = 0$ is suitable.

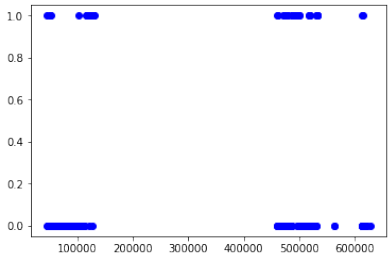
For the testing part, the remaining source IP data of scenario 10 is used as the test set after applying sliding window method with the same window size as before. If the amount of the data of certain source IP is less than the window size, the data will be dropped for the same format reason. Based on the log probability produced by HMM model and the threshold, we can easily identify the infected IP and normal IP. As the table 4 shows, 15 new infections are found by our method while 66 false positive are produced. For the infected source IP address with few data, they are hard to detected for our method. Because we are using the source IP 147.32.84.165 to build the HMM model, the behaviors like the pattern shown in figure 5a will be detected by our profile. If the behavior shows a huge difference compared to the 147.32.84.165, like the figure 5b ,it would not be detected by our profile.

Table 4: The confusion matrix of HMM model

	Infected (Truth)	Normal (Truth)
Infected (Detection)	15	66
Normal (Detection)	19	264



(a) The pattern of source IP 147.32.84.165



(b) An example of the pattern of infected source IP that can not be detected.

Figure 5: The behavior of source IP.

5 Insight

To sum up, based on the results presented in the previous sections, we tend to select *Protocol* and $Packets_Byte$ as features for Botnet detection. In section 3 features are encoded into one value for discretization. Although the known infected hosts and normal hosts can be detected well, most of the unknown Botnet hosts couldn't be detected because they don't have the indicative behavior("ICMP",byte>1066). Therefore, in section 4 the HMM model is used for profiling task. By considering the sequential behaviors of a certain host on the encoding feature, we are able to know whether the host is infected or not based on the existence of their sequential behaviors instead of only considering the abnormal code. As a result, some of the Botnet could be detected successfully, which is better than the previous one.

6 Github

The code and tables of above methods can be retrieved on: Github
https://github.com/Po-Shin/Cyber_data_analysis/tree/master/Assignment3

References

[Pel17] et al Pellegrino, Gaetano. Learning behavioral fingerprints from netflows using timed automata. 2017.