CDA Lab 3: Streaming and profiling

Cas Bilstra 4381084
Roy Graafmans 4299442

https://github.com/Roy-Graafmans/CDA_CS4035

June 26, 2019

Please read the readme.md file for instructions for running the code. Please download the applicable data-sets (6 and 10) yourself, as their size is too big to be allowed when uploading to the peer review system. The labeled unidirectional network files should be put in the data folder and be named 'scenario6.labeled' and 'scenario10.labeled' respectively.

1 Sampling

Scenario 6 (CTU-Malware-Capture-Botnet-47) was used for the sampling. There are 1582 different IP addresses the infected IP (147.32.84.165) sends packets to. Traffic in the other direction (hosts which send packets to the infected IP) was not taken into account, as it seems from the assignment description that only the traffic from the infected host to other hosts should be looked at. The code would not change much though, just that instead of only 'from_infected' the variable 'to_infected' should be taken into account for the reservoir sampling as well (see main.ipynb, simply combine them to one big list before the sampling).

In fig. 2(a) the real distribution (obtained by simply keeping one counter per IP) of the 300 most occurring IP addresses can be seen, together with the estimated distribution (by using reservoir sampling with one pass) with reservoir size 300. They have both been normalized to the total IPs seen (300 for reservoir, as only 300 are saved and 1582, or all unique IP addresses, for the real distribution) to be able to compare them. It can be seen that their shape is similar, so characteristics such as the general distribution seem to be preserved.

In fig. 1, the real and estimated (with reservoir size 300) 10 most frequent addresses with their counts can be seen. There are 7 addresses which are estimated to be in the top 10 correctly, three of them should not be in the top 10. The most frequent item was estimated to be present in 6% (18/300)of the rows, while the real most frequent item occurrence is in 0.0671% (384/5724) of the rows. This is a difference of for about 12%.

n	real	real count	estimated	estimated count
1	91.212.135.158	384	91.212.135.158	18
2	64.59.134.8	194	64.59.134.8	13
3	24.71.223.11	160	216.32.180.22	9
4	216.32.180.22	154	65.55.92.152	8
5	65.55.37.72	96	65.55.92.136	7
6	65.55.37.88	94	65.55.92.184	7
7	65.55.92.136	93	65.55.37.72	7
8	65.55.92.168	90	65.55.37.104	6
9	65.55.92.152	90	65.55.92.168	6
10	202.108.252.141	89	65.54.188.110	5

Figure 1: Real and estimated top 10

As it would be too much to show the estimated frequencies for all reservoir sizes, we calculated the overlap (defined as the number of IPs which are present in both the estimated and real top 10 most occurring IPs) averaged over 500 runs (as the results are not deterministic) for reservoir sizes between 50 and 700, which can be seen in fig. 2(b). As the overlap estimate becomes better, the distribution of the frequencies of the individual IPs are also estimated better on average. It can be seen that until a reservoir size of 150 the overlap grows quite steady, after which it only grows slowly. Depending on the amount of memory available, the 'sweet spot' should be chosen, looking at the trade-off between the amount of memory used and size of the detected overlap (or other metrics).

The algorithm does not know the real count for each IP (to save memory space), but only keeps reservoir_size samples. This means not every occurrence of an IP will be saved. The chance that an IP is saved is uniform over the sample, thus $\frac{1}{sample\_size}$. As it is only a probability that some IP is saved, it depends on a chance if the top 10 IP addresses will be estimated correctly. When the reservoir size grows, this chance increases as more IPs can be saved, which can clearly be seen in the figure because the overlap increases on average. Because the chance that some element in the reservoir is swapped is random, results are not deterministic which is why the experiment was run 500 times per reservoir size. When choosing reservoir size 5724 (the total number of applicable rows) the overlap score is 10 (everything correct) as expected because then every IP occurrence is saved.

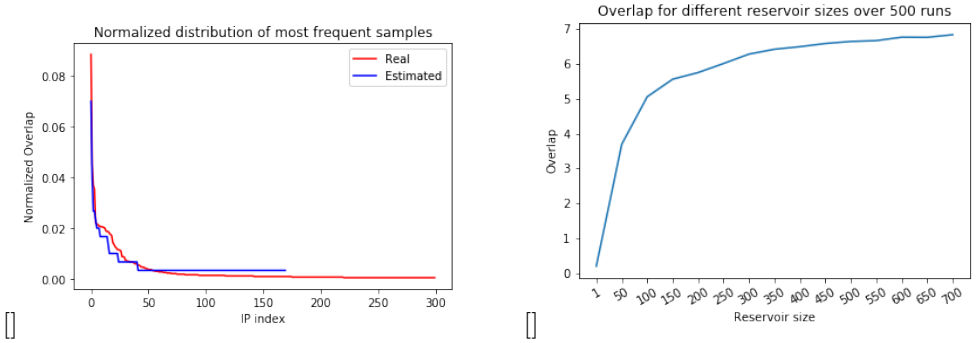


Figure 2: Sampling plots

2 Sketching

Width	Depth	Overlap	Estimate for most occurring IP	Avg Time (s)
10	1	0	1018	0.00940
10	5	1	705	0.0278
10	10	4	685	0.0500
10	50	6	593	0.223
10	100	8	593	0.436
10	1000	7	584	4.35
50	1	0	428	0.00908
50	5	7	428	0.0270
50	10	9	428	0.0487
50	50	10	407	0.224
50	100	10	405	0.441
50	1000	10	405	4.49
100	1	0	411	0.00912
100	5	9	411	0.0271
100	10	9	403	0.0489
100	50	9	395	0.223
100	100	9	390	0.442
100	1000	9	388	4.47
1000	1	2	395	0.00929
1000	5	9	384	0.0282
1000	10	10	384	0.0512
1000	50	10	384	0.235
1000	100	10	384	0.468
1000	1000	10	384	4.90

Figure 3: Estimates and overlaps for different widths and depths (heights)

In fig. 3, the overlap for the 10 most frequent IP addresses such as discussed in section 1 and estimate for the most occurring IP can be seen for different widths and depths(heights) of the count-min sketch. Interesting to see is that increasing the width works better than increasing the depth with the same amount of data storage (saving 100*5 works better than 50*10, 1000*10 works better than 100*100, etc.). This is not the case for a depth of 1, as then no minimal value of multiple hash functions can be taken for an estimate of the count. Looking at fig. 2(b), when a reservoir size of 250 is used (with thus 250 IP addresses in memory), the average overlap is for about 6. For sketching with width 50, depth 5 this is 7. Looking at 500 memory spaces, the overlap is for about 6.5 for sampling and 9 (width=100, depth=5) for the sketching. CM sketching works better than sampling from for about 250 used memory spaces. Only with 50 memory spaces sampling clearly outperforms sketching with on overlap of almost 4 opposed to 1 for width=10, depth=5. Space efficiency is thus better for the sketching method as a higher overlap can be reached with the same amount of memory addresses. For the accuracy, we compared with the results for reservoir size 300 (12% off) and 500 (7% off). For 250 memory spaces, sketching was $(428-384)/384 = 11\%$ off and for 500 memory spaces, sketching was $(411-384)/384 = 7\%$ off as well. We thus did not observe a big difference on accuracy for measuring the occurrence of values.

The runtime of the sampling technique can be seen in fig. 4. The runtime of the sketching algorithm can be seen in the last column of fig. 3. This runtime is the averaged runtime over 50 runs, and it can be seen that the runtime mainly depends on the depth (number of hash functions). As calculating a hash takes some time, this is also expected. The width is implemented by a simple module operation, which only adds a negligible amount of overhead when increasing. The longest runtime of the sampling algorithm is four times as fast as the fastest runtime of the sketching algorithm. The sampling algorithm is thus a lot faster than the sketching algorithm, and also for the runtime is becomes clear that the better option is to increase the width than it is to increase the depth (except for depth = 1, which just is a bad idea as described before).

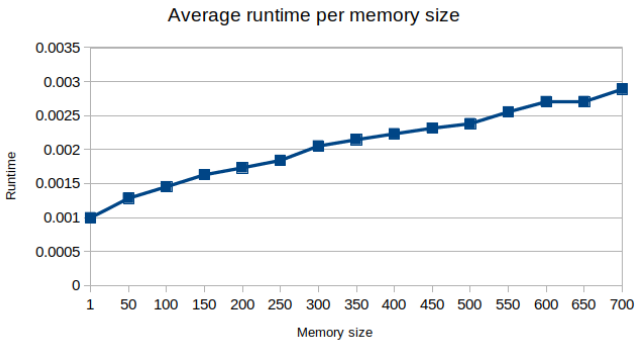


Figure 4: Sampling runtime per memory size

The approximation errors we observe in the estimate for the most occurring IP can be explained by the way the COUNT-MIN algorithm works. As the 1582 unique IP addresses are mapped to a smaller space of *width* values, some IPs will be mapped to the same number (a collision). This means the bucket in the CM matrix might be updated by more than just one IP address. Their counts are then added together, which of course produces too many estimated occurrences. This is why it is better to use multiple hash functions, such that the hash function with the least collision for a certain IP can be selected as the 'leading' hash function for that IP. Because of these collisions, the estimated amount will always be bigger or equal than the actual amount. In 3 it can also be clearly seen that when increasing the depth (number of hash functions), the estimate never gets worse. As replacing a random IP in an array is faster than calculating new masks and calculating hashes, it is expected that the sampling algorithm works faster than the sketching algorithm. Performance optimizations are possible for sketching though, such as saving the used hash functions and performing updates in bulk, which could bring the runtime of the sketch algorithm down a bit.

3 Flow data discretization

For this assignment scenario 10 (CTU-Malware-Capture-Botnet-51) was used. The data contains 10 infected hosts and 6 normal hosts. For the comparison of features between normal and infected hosts we picked one infected host (147.32.84.165) and compared the mean as well as the standard deviation for every feature. This gave the following results:

Feature	Host type	Mean	Standard deviation
Duration	Normal	0.03	0.31
	Infected	0.55	1.44
Tos	Normal	0.0	0.0
	Infected	0.0	0.0
Packets	Normal	1.77	33.22
	Infected	15.84	102.15
Bytes	Normal	781.56	34258.15
	Infected	14104.12	84291.2
Flows	Normal	1.0	0.0
	Infected	1.0	0.0
Protocol category	Normal	0.94	0.24
	Infected	1.94	0.31

Table 1: Mean and Standard deviation for the features of normal and infected host data

Features where a clear difference in the statistics between normal and infected hosts can be observed were taken together and visualized, the result which can be found in appendix A. The 2 most relevant features in this stadium are probably bytes and protocol since there is a clear distinction in the protocols used and the range of bytes is larger than the range of packets which makes it easier to distinct them.

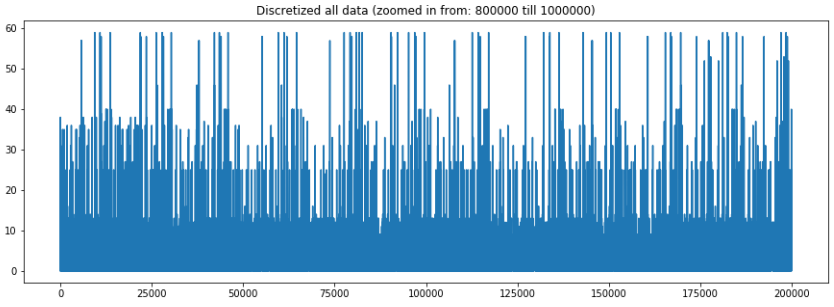
The combination of duration with packets or bytes is not very successful, as normal and infected data are not clearly separated. The combinations (Protocol, Duration), (Protocol, Bytes), (Protocol, Packets) and (Bytes, Packets) make a more clear separation to investigate further using the netflow algorithm to determine which one would be most useful. We made this comparison by plotting the netflow outcomes of our infected host as well as the normal hosts in appendix B.

The netflow algorithm combines discretized data by calculating a score for each netflow. The score is the sum of every mapped feature values times the space size divided by the mapping size. The discretization works for numerical as well as categorical data. The numerical data is discretized by defining the number of bins. For every bin the percentile boundaries are calculated. The categorical data is discretized by numbering each unique value and returning that.

For example: taking the features protocol and bytes. There are 3 protocol options: TCP UDP and ICMP. The discretized values are respectively: 0, 1 and 2. For the bytes we have 1,1,1,5,12,14,14,18,23,31. When there are 5 bins, we calculate the boundaries for the 20th, 40th, 60th adn 80th percentile. the boundaries are: 1, 5, 14, 18. So a netflow with $< TCP, 6 >$ is mapped as $< 0, 2 >$. Based on this mapping, the mapping size for protocol is 3 and the mapping size for bytes is 5. So the space size for this netflow is initialized of 15. The score is: $0 * 15/5 + 2 * 3/3 = 2$

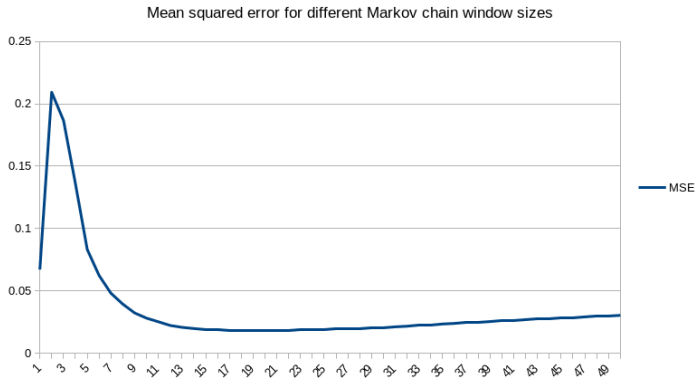
When all the netflows for an infected host are discretized and plotted against the discretized normal hosts, the combinations (Protocol, Duration), (Protocol, Bytes), (Protocol, Packets) show a clear distinction. The plots can be found in appendix B. Although the distinction for (bytes, packets) is visible, the differences are smaller. Looking at the averages of the netflow scores, the biggest distinction is visible for the combination (protocol, bytes) since the top of the normal data is the same (9) but the bottom is much lower (this can be seen in the figures as well: the y-axis starts at 0 instead of 4 in that figure). We can thus conclude that the discretization works best for the features 'Protocol' and 'Bytes' together.

When the discretization is applied to all the data, the connections are not directly visible anymore. When combined, there is a wider range for all the features and the netflow score is only one dimensional. In order to find botnets, profiling should be applied, which will be perfomed in the next section. The results for discretization of all data can be found in the figure on the right:

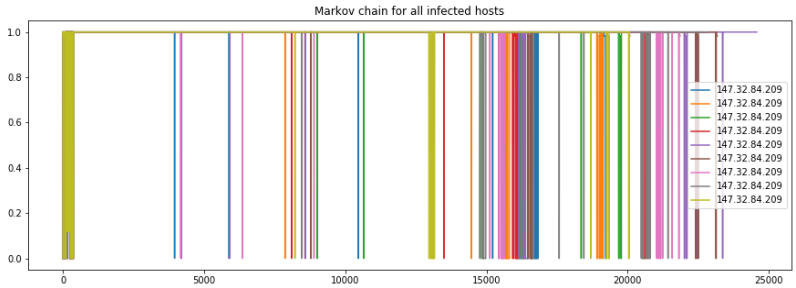


4 Botnet profiling

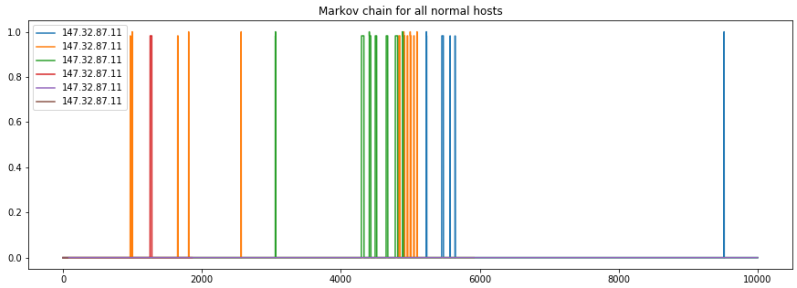
For this assignment we chose to use a Markov chain in combination with the discretized data from the previous section. We trained the Markov chain on the first infected host (147.32.84.165) and then processed the data per host. As determined in the previous section, the protocol and byte feature should work the best, so those were used for the discretized feature. For every host a plot was made. Based on the error it is very easy to say if a host is infected as can be seen in section 4 and section 4. There is a clear differences between the normal and infected averages. In order to optimize the model we calculated the mean squared error for every host from window size 1 till 50. It came clear that a window size of 20 was most optimal wit a MSE of 0.0182

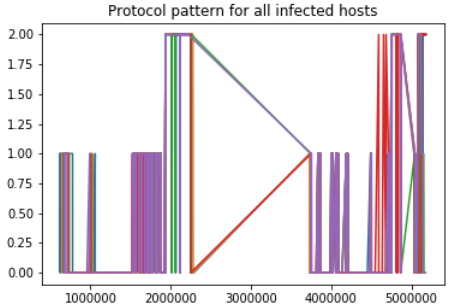


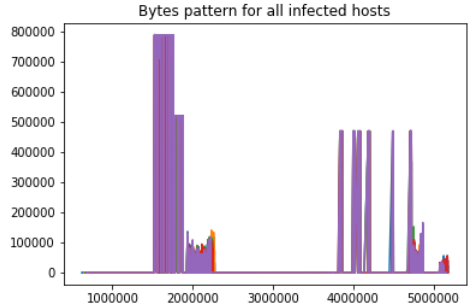
Using the Markov chain trained on an infected host, it was able to detect all other infected hosts as well using the error based threshold, (instead of summing the absolute subtraction between the likelihood and label, we used the average likelihood overall), where the lowest score was a likelihood of 98% being an infected host. The normal hosts were also clearly visible. There the highest score was 2% likelihood of being infected were 0% is the perfect score for normal hosts. There were no false positives for this scenario.



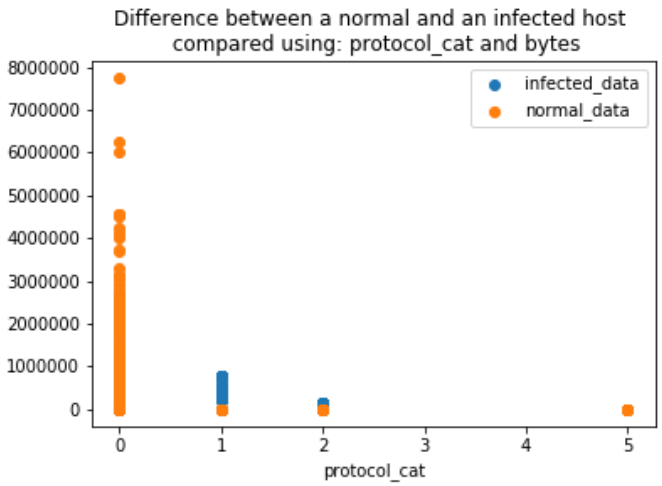
A Markov chain with a sliding window is looking for known sequences of states. When a sequence is unknown, the likelihood of being the same type as the training type drops. The behaviour our profile detects is thus abnormal behaviour in the combination of protocol and bytes. Specifically, it detects if a lot of bytes are sent with a certain protocol (mostly UDP). So when trained on an infected host, a higher likelihood means a higher chance that host being infected as well. Same goes the other way around when trained on normal data. When looking at these features over time (netflows), they look almost similar for every infected host. For the protocol type as well as the bytes send there is a clear pattern visible.

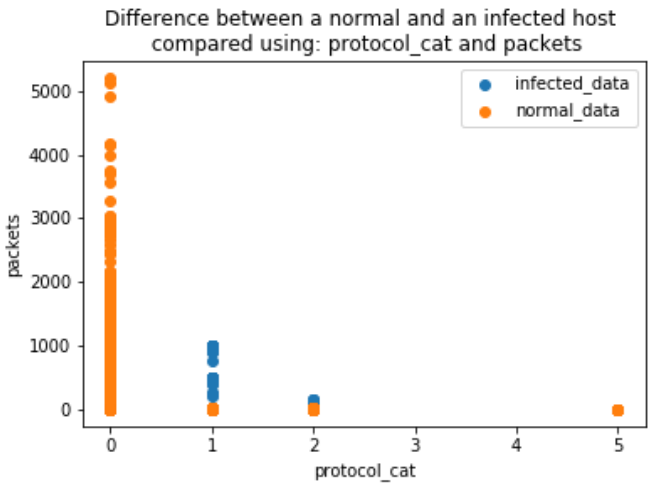


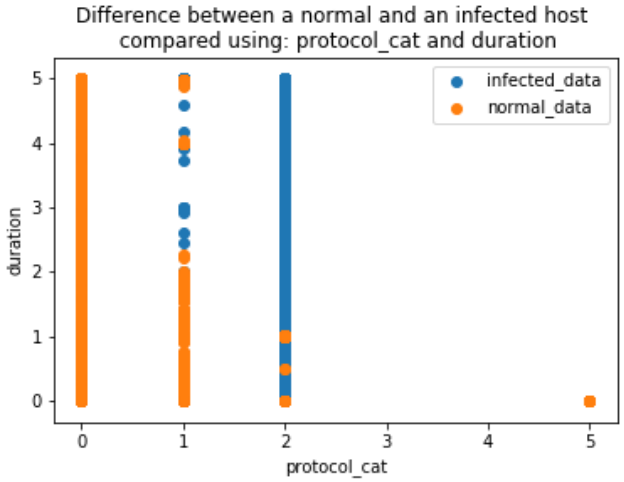


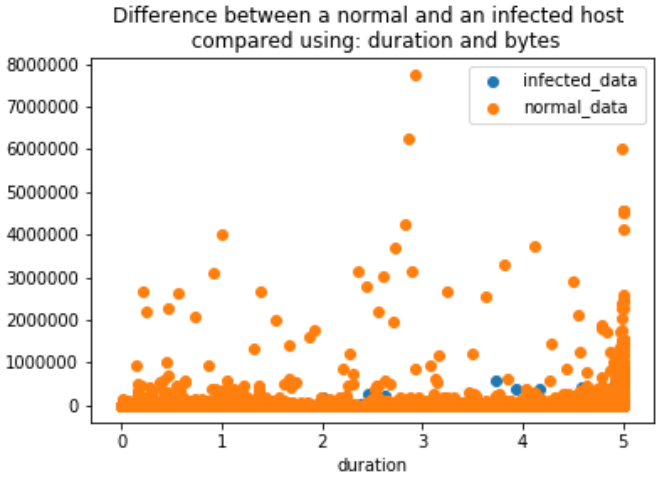


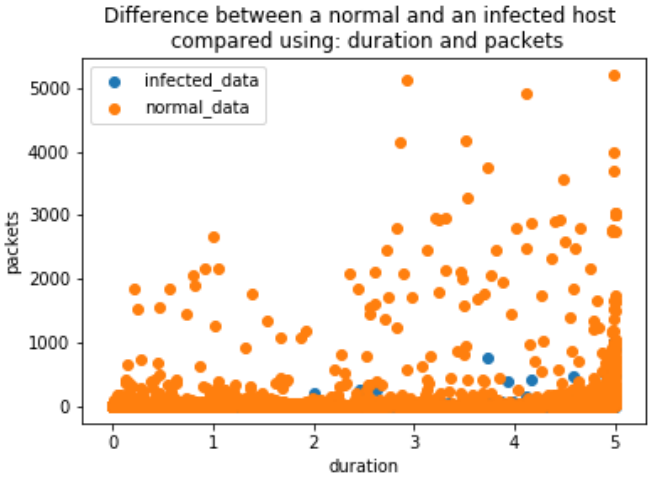
A Plots for visibility of differences between infected and normal hosts

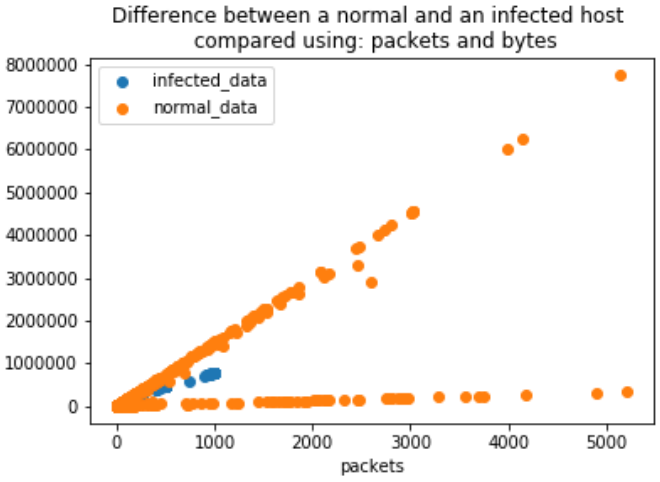












B Discretization of different feature combinations between infected and normal hosts