Delft University of Technology

Cyber Data Analytics

CS4035

Assignment 3

Authors: | *Student Numbers:*
Alina Bianca Iancu | 4930509
Gabriele Mazzola | 4815505

June 14, 2019

Contents

1 Sampling Task

For this task, we decided to work with capture 44 of the CTU-13 dataset. As seen in the documentation, there is only one infected host in this dataset, which has IP '147.32.84.165'. As an initial step, since the data contained records combining the IP and the port of the source and destination addresses, we preprocessed the data for separating these two in two different columns.

For further analysis, we streamed the data once and kept only the records where we encountered the infected host either in the source or the destination address, resulting in 75891 flows. Next, we computed the real distribution over the other IP addresses that the infected host connects with by counting their occurrences. The 10 most frequent values are the following: '178.77.71.27' (0.163708 frequency), '38.229.70.20' (0.050441 frequency), '147.32.96.45' (0.016049 frequency), '195.113.235.89' (0.009527 frequency), '109.74.55.27' (0.005231 frequency), '64.31.13.148' (0.003716 frequency), '208.86.166.68' (0.003716 frequency), '210.139.61.102' (0.002385 frequency), '208.100.127.34' (0.001186 frequency), '210.139.61.70' (0.000949 frequency).

We then employed Reservoir sampling for estimating the distribution in one pass of the other IPs that the infected host connected with. We analyzed the behaviour of the sampling approach based on different reservoir sizes. The sampling for each reservoir size value was evaluated using the recall only among the top 10 retrieved IP addresses. More specifically, we computed how many IPs from the top 10 most frequent observed in the real distribution of the data we obtained in the top 10 most frequent values observed in the sampled distribution using that specific reservoir size. In Figure 1 you can observe the recall value associated to each reservoir size that we have experimented with. Additionally, we computed the mean frequency estimation error among the top ten estimated IPs, plotted in Figure 5 included in the Appendix. What is interesting to observe in the plots is that when the reservoir sizes are too small, the recall is really low (and the mean frequency estimation error high), as there is high probability of keeping on replacing values stored in the reservoir, thus higher chances of obtaining a distribution that is not representative of the real distribution. However, the higher the reservoir size becomes, the lower the probability that the values from the reservoir are being replaced, and thus there are more chances of getting closer to a more representative distribution of the real one. As we can see in the plot in Figure 1, for reservoir size value of above approximately 10000, the recall starts getting more stable around a value of 1.

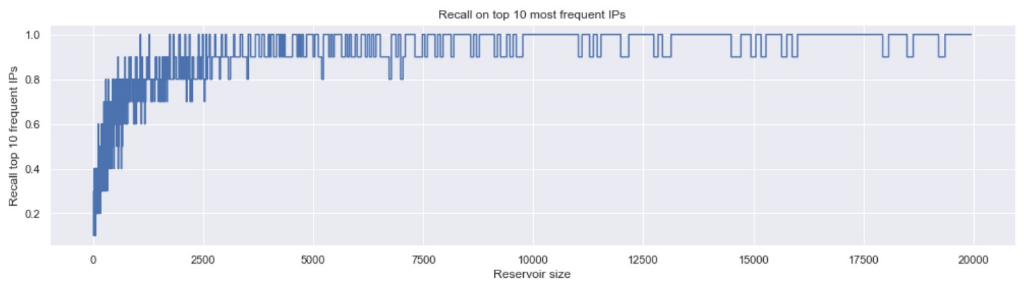


Figure 1: Values of recall for the top 10 most frequent IPs for different reservoir sizes

When performing Reservoir sampling with the reservoir size of 10000, we obtained the same 10 most frequent values among the IPs that the infected host connects as in the real distribution (thus, a recall of 1) with a slightly altered order, that can be explained by the fact that Reservoir sampling is a probabilistic approach (repeating it multiple times yields slightly different results): '178.77.71.27' (0.1634 frequency), '38.229.70.20' (0.0513 frequency), '147.32.96.45' (0.0154 frequency), '195.113.235.89' (0.0087 frequency), '109.74.55.27' (0.0064 frequency), '64.31.13.148' (0.0039 frequency), '210.139.61.102' (0.0030 frequency), '208.86.166.68' (0.0028 frequency), '210.139.61.70' (0.0012 frequency), '208.100.127.34' (0.0007 frequency).

2 Sketching Task

For this task, we implemented and employed the Count Min Sketch approach as the sampling strategy. Same data as preprocessing steps as in Task 1 were used. We started by hashing each given IP address using several independent hashing functions and adding it to the corresponding bucket for each hashing function in the Count-Min sketch matrix. Once we had the final matrix, we rehashed each IP address, identified the bucket locations for that IP address and chose the minimum value of the bucket size among the corresponding buckets as the frequency for that IP address, as it is the most reliable one.

As in the case of the Reservoir sampling, we experimented with several values for the height and width of the Count-Min sketch matrix, using the recall only among the top 10 IP values as the evaluation metric. Additionally, as before, we computed the mean frequency estimation error among the top ten estimated IPs. The values we have experimented with were heights between 2 and 15 with a step size of 1 and widths between 50 and 1000 with a step size of 25.

In order to visualize the behaviour of Count-Min sketch sampling based on different values, we plotted a heatmap showing the values of the width and height of the matrix, together with the associated recall, visualized in Figure 2. Interesting to observe is that indeed, as theoretically expected, very low numbers of hashing functions lead to a low recall (due to collisions), no matter how high the number of buckets is, as decreasing the number of hash functions results in a higher probability of bad estimate. Additionally, we can see that when the number of buckets gets higher the recall increases, independently of the number of hashing functions, as the higher the number of buckets gets, the higher the accuracy of the buckets counts becomes, as they get less biased by collisions (this happens because more buckets means higher ranges for the hashing functions).

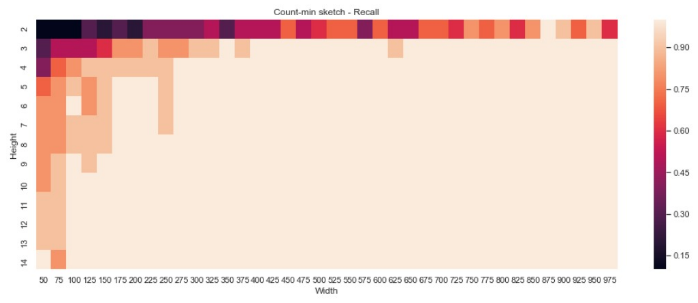


Figure 2: Values of recall for the top 10 most frequent IPs for different height and width values for the Count-Min sketch matrix

Observing the performances over the different values ranges, we chose the following parameters for the Count-Min sketch matrix: a height of 9 (so 9 independent hashing functions) and a width of 150 (so 150 buckets), as starting from around those values the recall stabilizes to a value of 1 (as it can be seen in the heatmap in Figure 2). The resulting top 10 most frequent IPs connected with the infected host are the same as in the real data and in the same order: '178.77.71.27' (0.168149 frequency), '38.229.70.20' (0.054552 frequency), '147.32.96.45' (0.019936 frequency), '195.113.235.89' (0.014099 frequency), '109.74.55.27' (0.010001 frequency), '64.31.13.148' (0.008499 frequency), '208.86.166.68' (0.007498 frequency), '210.139.61.102' (0.006747 frequency), '208.100.127.34' (0.005640 frequency), '210.139.61.70' (0.005244 frequency).

Regarding the space efficiency, Reservoir sampling is more efficient than Count-Min sketch, as in the case of the latter we also need to store the unique IPs that we see while the data is streaming. Without doing this, we would not be able to retrieve the counts for each IP address, as we need to hash the IP to retrieve the indices of the corresponding buckets. With regards to the run-time, we observed that count-min sketch runs faster than reservoir sampling (at least, with our implementations).

3 Flow data discretization Task

For this task we use capture 51 of the CTU-13 dataset (corresponding to scenario 10). As before, since the data contained records combining the IP and the port of the source and destination addresses, we preprocessed the data for separating these two in two different columns. Additionally, we removed all the records corresponding to background flows.

As stated in the assignment, we are asked to select one of the infected hosts and investigate and discretize the data. The IP of the infected host that we selected is '147.32.84.165'.

Before moving on to the discretization, we analyzed the features of the selected host in order to choose the most representative ones for modeling the behaviour of the infected host. We plotted the 'duration', 'packets' and 'bytes' features for the infected and normal data, visualized in Figure 7 included in the Appendix. We can definitely observe the different behaviour for the infected and normal data, as we observe more frequent and dense groups of peaks in the case of all three features for the infected data. We decided to further proceed with the 'duration' and 'packets' (which was highly similar to 'bytes') features.

For the discretization, we followed the approach presented in [1]. We start by performing clustering for each of the features using different number of clusters and for each of them we plot the scores; the two plots can be found in the Appendix, in Figures 8 and 9. Using the Elbow method, we identified the optimal number of clusters for each feature and we identified 2 clusters for 'duration' and 4 clusters for 'packets'. Next, we visualized the distribution of the two features and we manually chose the breakpoints such that the discretization would still represent the behavior of the original signal. The number of chosen breakpoints was determined such that the data would be divided in the number of clusters identified with the Elbow method. After the percentiles for the breakpoints were define, we proceeded as presented in [1] for computing the mapping of each data point corresponding to the two features to a discrete value depending on which cluster it belonged to. It is important to note that that the discretization using this mapping was performed on the entire data from all the hosts, not only the one used for defining the breakpoints. Next, as discussed in [1], we combined the two discrete values corresponding to each of the two features into one numerical code. Using this code, we plotted the data corresponding to infected and normal hosts in order to check whether there are any differences. For visualization purposes, we chose the host with IP '147.32.84.165' as the infected host and the host with IP '147.32.84.170' as the normal host. The visualization is shown in Figure 3 and we can observe different structures for the normal and infected host.

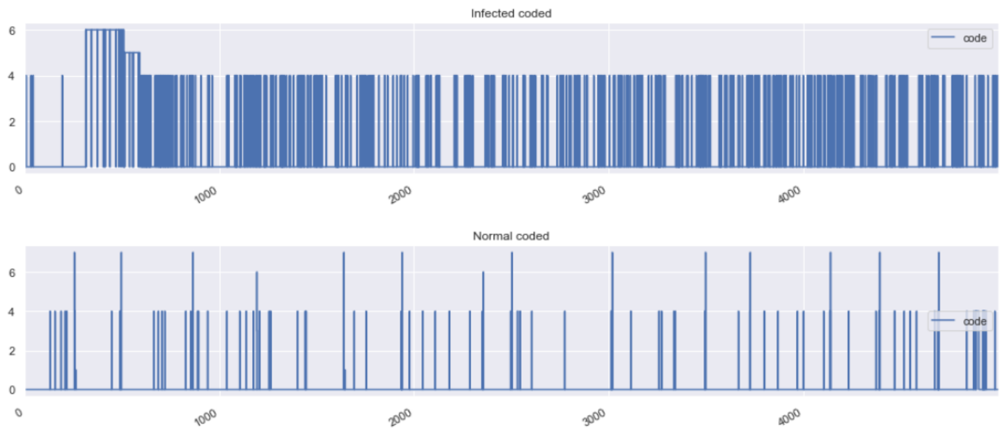


Figure 3: The discretization using a unique numerical code corresponding to an infected (IP: '147.32.84.165') and normal (IP: '147.32.84.170') host

4 Botnet profiling Task

For the profiling of the Botnet we decided to employ an Hidden Markov Model, which we train with the sequences extracted by means of a sliding window approach. The same preprocessing steps as described in Task 3 were employed.

Sliding window

By following the approach suggested in [1], we slide a window over the flows (encoded as described in the previous section) of an infected host in order to generate sequences of events. We decided to adopt both incoming and outgoing flows. Starting from a certain flow (time t), we "capture" all the events happening within the time window length (between time t and $t + w$, where w is the length of the window). Intuitively, the generated sequences have different lengths. For example, during a "burst" of data, there will be many flows happening within the window, on the other hand, many windows will contain only one event (in the case of an isolated flow). As it is explained later on in this section, we adopted a window length of 20 milliseconds, as in [1].

HMM

As stated above, we decided to perform profiling of the infected host (as well as detection of the others) using Hiddem Markov Model. We used the **hmmlearn** library of Python[1]. We employed a maximum likelihood approach in order to select the number of states of the model. As shown in Figure 4, the number of states that maximized the likelihood on the training sequences is **2**.

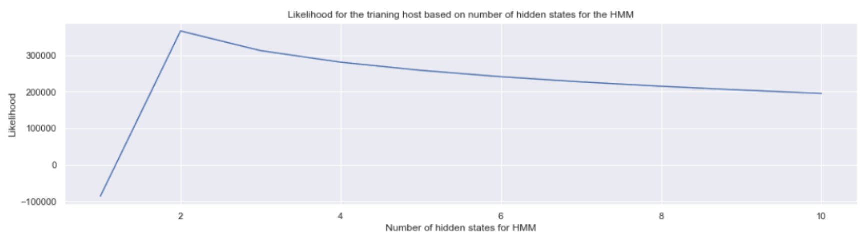


Figure 4: Likelihood associated to different numbers of hidden states for the Hidden Markov Model

In order to evaluate the other hosts, we needed to determine a detection threshold. Our reasoning is as follows: if a certain host is infected the same way the training host is, then the likelihood for such host should be similar to the one obtained for the training host. Therefore, we decided to introduce a similarity percentage (90%) with the profile of the infected host on which the HMM was trained. Therefore, the final detection threshold is 90% of the likelihood obtained for the training host.

For each of the hosts identified in the website of the scenario 10[2], we performed the sliding window to extract sequences of states, and then we computed the loglikelihood of those sequences being generated by the model of the infected host we trained on. If the obtained value was higher than the threshold (explained above), then we would mark it as infected.

The results, presented in the Appendix in Table 1, are really good: we obtained 100% accuracy in the detection of these hosts. With regards to the type of behaviour that we are able to detect with this model, by means of profiling we are able to detect those hosts whose behaviour is similar to the one we trained the HMM on. If we wanted to detect other behaviors, we would need additional HMMs trained on other botnets.

[1]https://hmmlearn.readthedocs.io/en/latest/
[2]https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-51/

5 Flow classification Task

For this task we used the capture 51 of the CTU-13 dataset corresponding to scenario 10. The data was preprocessed as we have already described before, by separating the IP and the port from the source and destination address and removing the background flows.

As an initial step, we performed a visual analysis, by plotting the different features for infected and legitimate hosts. It was interesting to observe that in the case of almost all features, the behaviour was different for the infected hosts when comparing with the normal hosts. All the different plots can be found in the accompanying notebook, we will only include here the plot corresponding to the 'protocol' feature, visualized in Figure 10 included in the Appendix. We can definitely observe the discrepancy between Botnet and legitimate data, as the dominant protocol type in the case of Botnet data is 'ICMP', whereas in the case of legitimate data it is represented by 'TCP'. After the analysis phase, the features that we decided to proceed with for the classification are the following: 'protocol', 'bytes', 'src_port', 'dst_port', 'flags' and 'duration'. The categorical variables, namely 'protocol' and 'flags', were encoded using one-hot encoding.

We proceeded with the classification task by splitting the data in train and test sets in such a way that the integrity of the hosts was preserved (i.e. the data from the same host was kept together either in train or test). We performed the splitting manually based on the documentation in such a way to achieve approximately 70% data in the training set and 30% data in the testing set. We observed that the data was not balanced in the training set, thus we employed undersampling on the training test in order to balance the two classes.

The classifier that we employed was Random Forest. We manually experimented with different number of trees in the random forest (i.e. the 'n_estimators' hyperparameter), ranging from 50 to 500. By analyzing the confusion matrices, we decided to keep this hyperparameter as 200 as it yielded the most stable result. Furthermore, we did not want to perform too much optimization, in order to avoid overfitting.

Regarding evaluation, we evaluated the performance of our classifier both at the packet, as well as at the host level. For both approaches we obtained satisfactory results. In the case of the **packet level**, as seen in the confusion matrix shown in Figure 11 in the Appendix, we achieved very high precision and recall rates of approximately 0.99, with a low, balanced number of false positives and false negatives. Regarding the **host level** evaluation, we used two approaches: majority voting (i.e. if more than half of the packets belonging to a host are classified as infected, then the host is classified as Botnet, otherwise as legitimate) and marking a host as infected if at least one of its corresponding flows is marked as infected. In the case of the majority voting we can observe in the confusion matrix included in Figure 12 in the Appendix that we achieve a perfect performance, correctly classifying all the Botnet and legitimate hosts in the test set. However, for the second host level evaluation approach, the classification resulted in more false positives, as all the legitimate hosts were classified as Botnet. This is expected, as hosts have a high number of flows associated to them and there is a high chance that at least one of them is detected as infected.

Finally, using our Random Forest classifier, we computed the importance of the features for the classification, shown in Table 2. We can observe that the source and destination port play an important role in distinguishing between Botnet and legitimate flows. Additionally, among the most important features are also the number of bytes of the flow and the 'ICMP' and 'TCP' protocols, which supports our earlier discussion regarding the obvious dominance of the 'ICMP' protocol in the case of Botnet and 'TCP' protocol in the case of legitimate hosts.

Regarding the comparison between the sequential model implemented before the classifier, we consider that a sequential model is more complex, as decisions have to be made both in the extraction of the states (for instance, when applying sliding window or when discretizing features) and in the modeling phase (e.g. choosing the models, such as HMM or state machines). Based on the results obtained in our experiments, we would choose a classifier for this task, as it is easy and fast to train and yields good performance.

However, sequential models, such as the HMM we built, enable possibilities beyond basic classification, such as profiling and fingerprinting. Additionally, the stronger point of the sequential models, is that a model can be built using for instance only data from an infected host, whereas in the case of a classifier we need more extensive and balanced training data.

References

[1] Gaetano Pellegrino, Qin Lin, Christian Hammerschmidt, and Sicco Verwer. Learning behavioral fingerprints from netflows using timed automata. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 308–316. IEEE, 2017.

Appendix

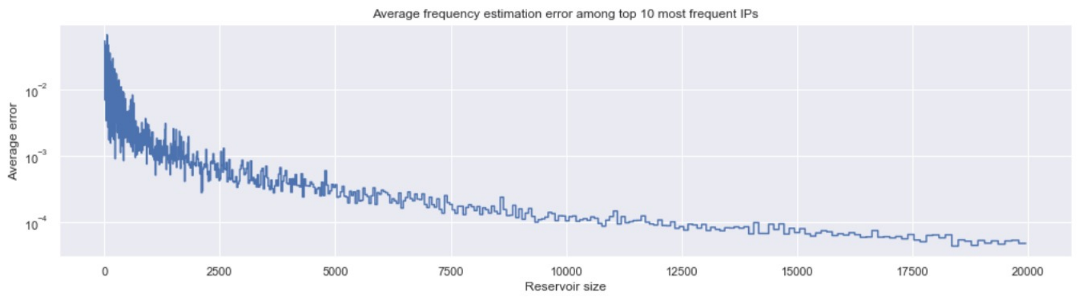


Figure 5: Values of mean frequency estimation error for the top 10 most frequent IPs for different reservoir sizes

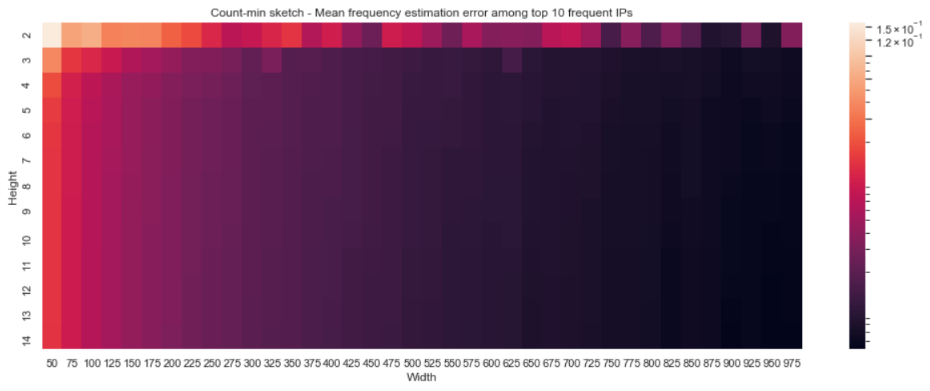


Figure 6: Values of mean frequency estimation error for the top 10 most frequent IPs for different widths and lengths of count min sketch

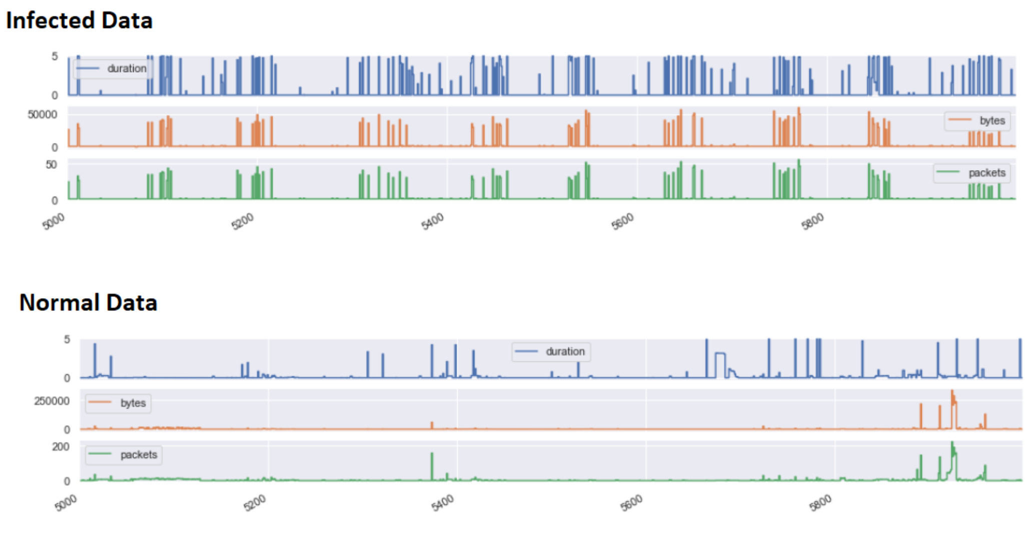


Figure 7: The 'duration', 'bytes' and 'packets' features plotted for infected and normal data

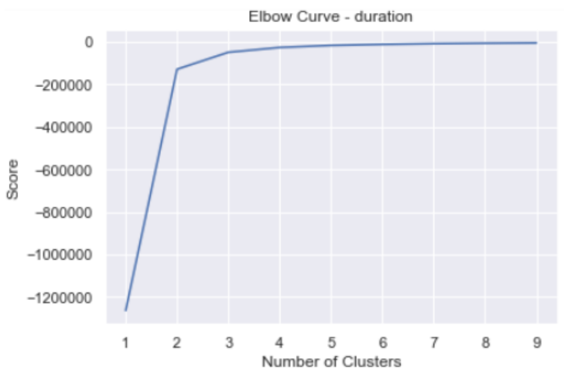


Figure 8: Scores achieved performing KMeans with different number of clusters for the 'duration' feature. Using the ELBOW method we observe that the optimal number of clusters is 2.

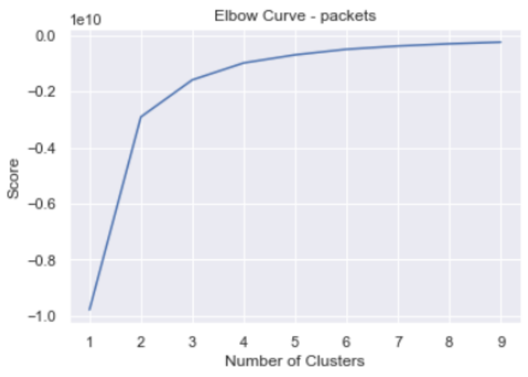


Figure 9: Scores achieved performing KMeans with different number of clusters for the 'packets' feature. Using the ELBOW method we observe that the optimal number of clusters is 4.

IP Address	True Label	LogLikelihood	Predicted Label
147.32.84.191	Infected	411563.21	**Infected**
147.32.84.192	Infected	410862.54	**Infected**
147.32.84.193	Infected	437472.31	**Infected**
147.32.84.204	Infected	430580.23	**Infected**
147.32.84.205	Infected	455115.4	**Infected**
147.32.84.206	Infected	432420.54	**Infected**
147.32.84.207	Infected	456786.14	**Infected**
147.32.84.208	Infected	399507.81	**Infected**
147.32.84.209	Infected	409377.53	**Infected**
147.32.84.170	Normal	-30126892.64	**Normal**
147.32.84.134	Normal	-2115421.96	**Normal**
147.32.84.164	Normal	-7115583.71	**Normal**
147.32.87.36	Normal	-165473161.54	**Normal**
147.32.80.9	Normal	210824.19	**Normal**

Table 1: Results obtained by means of the HMM trained on infected host '147.32.84.165'. The detection threshold was set to **329830** with the reasoning explained in the section. **Note**: normal host '147.32.87.11' is not reported as, filtering out background data, there were no flows available. **Note**: the last host in the table was almost detected as a false positive. However, the documentation says this host is not so reliable as it is a dns server.

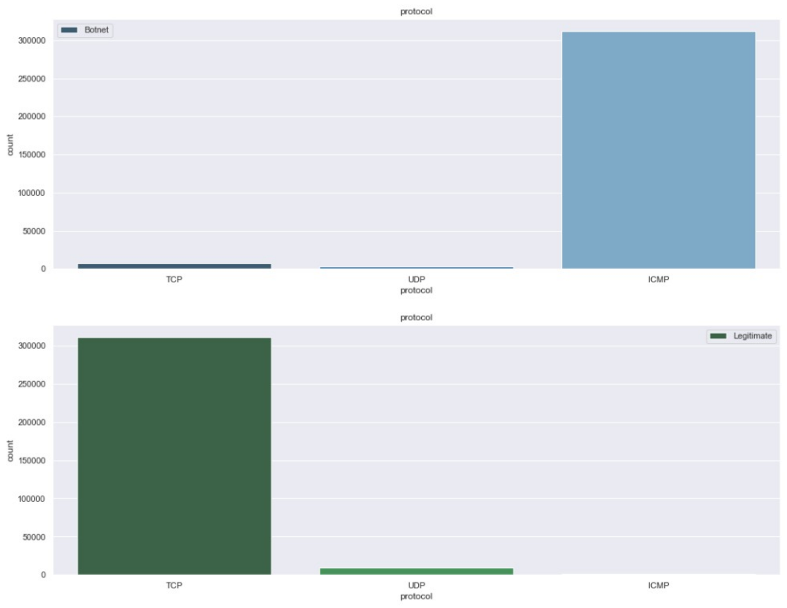
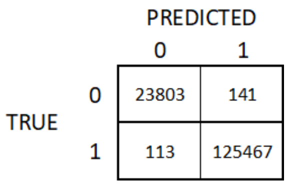
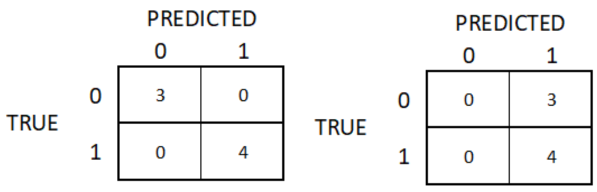


Figure 10: Visualization for the 'protocol' feature for the Botnet and Legitimate records



		PREDICTED 0	PREDICTED 1
TRUE	0	23803	141
	1	113	125467

Figure 11: Evaluation at **packet level**: confusion matrix for random forest. **Precision**: 0.9989, **Recall**: 0.9991



		PREDICTED 0	PREDICTED 1
TRUE	0	3	0
	1	0	4

		PREDICTED 0	PREDICTED 1
TRUE	0	0	3
	1	0	4

Figure 12: Evaluation at **host level** (7 hosts): confusion matrix for random forest. On the **left**, result using majority voting on the grouped flows per host. On the **right**, results when marking a host as infected if at least one of their flows is marked as infected.

Feature	**Importance Score**
src_port	0.22882
prot_ICMP	0.19074
dst_port	0.17450
bytes	0.14447
prot_TCP	0.09900
flag_A_	0.02140
flag_UNK	0.01976
flag_ECR	0.01958
prot_UDP	0.01950
flag_ECO	0.01676

Table 2: Feature importance score for random forest (top 10 important features).