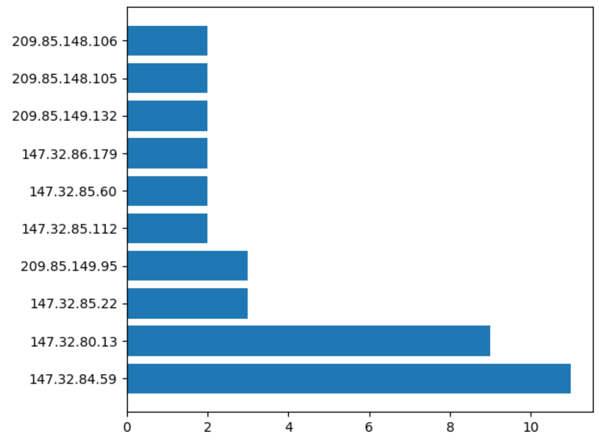
# Cyber Data Analytics

Lab 3 Assignment

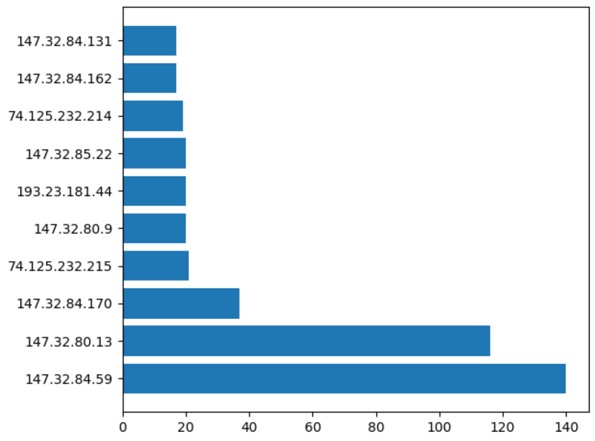
Kees Fani (4437179) and Jody Liu (4920392)

## 1 Sampling

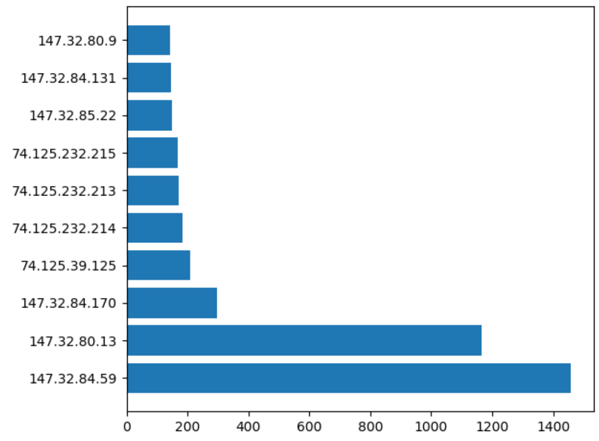
For the sampling task we used the CTU-Malware-Capture-Botnet-43 dataset. This dataset corresponds to scenario 2. We used reservoir sampling to find samples that are representative to the given data. In the following 4 figures, the 10 most frequent IP-addresses and their frequencies will be given. The last of these figures corresponds to the actual IP address frequency distributions in the data. Both the source and destination IPs were considered. The infected host '147.32.84.209' was filtered out.



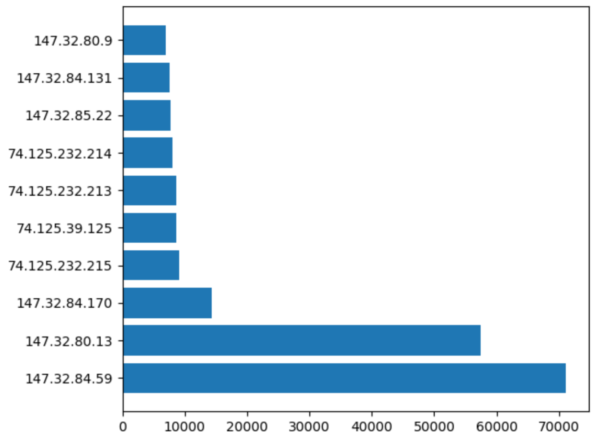
(a) The top 10 IP frequencies given a sample size of 100.



(b) The top 10 IP frequencies given a sample size of 1000.



(c) The top 10 IP frequencies given a sample size of 10000.

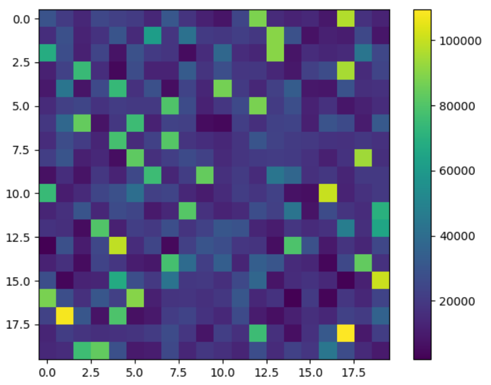


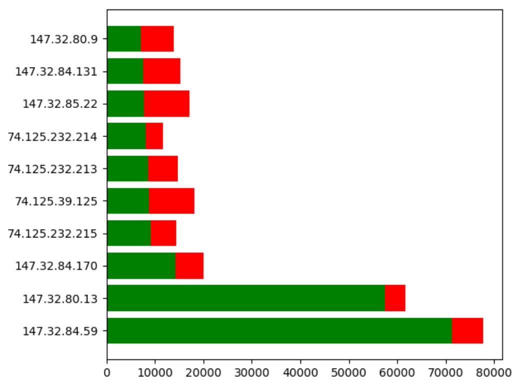
(d) The top 10 IP frequencies of the actual dataset.

We can clearly see a relationship between sample size and accuracy of the IP distributions. The higher the sample size, the higher the accuracy. This is because of the fact that any item in the dataset has a chance of m/n to be included in the sample. Here m refers to the sample size, and n refers to the number of items in the dataset. Thus increasing the sample size proportionally decreases the odds of disproportional item frequencies. Thus there is a clear trade-off between memory and accuracy.

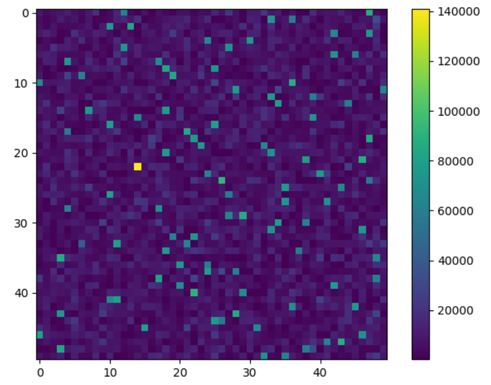
## 2 Sketching

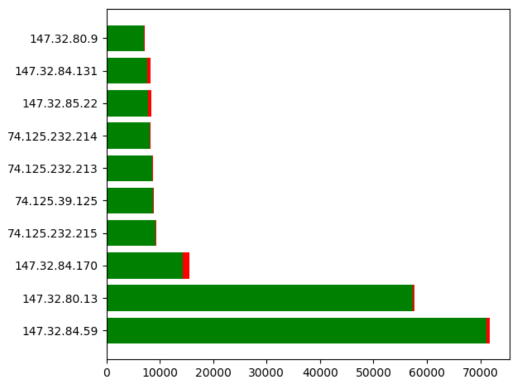
Another way to determine the frequency of IP addresses in this dataset, is by using what is called a COUNT-MIN Sketch. A COUNT-MIN sketch works by taking an amount $y$ of different hash functions. These hash functions have, with the help of basic modular arithmetic, an amount $x$ of outcomes. Every item $i$ fed into the COUNT-MIN sketch is hashed by every hash function $h$. For all hash functions $h$, the $matrix[h][h(i)]$ value is incremented. This $x \times y$ matrix is the sketch. To query this sketch with item $i$, compute the following: $\min_h matrix[h][h(i)]$. The results of this sketch and the sketches themselves are given in the following four figures.


(a) A $20 \times 20$ COUNT-MIN sketch, after being fed the IP addresses of the CTU-Malware-Capture-Botnet-43 dataset.



(b) The top 10 IP frequencies of the CTU-Malware-Capture-Botnet-43 dataset (green), appended by the excess values of the $20 \times 20$ COUNT-MIN sketch (red).


(c) A 50 x 50 COUNT-MIN sketch, after being fed the IP addresses of the CTU-Malware-Capture-Botnet-43 dataset.



(d) The top 10 IP frequencies of the CTU-Malware-Capture-Botnet-43 dataset (green), appended by the excess values of the $50 \times 50$ COUNT-MIN sketch (red).

Because COUNT-MIN sketching works by taking the minimum of the hash values, we can note two observations. First, some values in the matrix are way higher than the maximum frequency found in the dataset. These 'sketch outliers' are functionally useless, as their values will never impact the outcome of a query. Another observation is that the results of such sketch queries are never greater than the actual IP frequencies. This is because these queried matrix cells are always incremented at least once per occurrence in the dataset.
We can clearly see that increasing both the amount of hash functions and the amount of outcomes of these hash functions increases the accuracy of the sketch. Meaning that there is a trade-off between accuracy and memory, as the sketch clearly increases in size if either parameter is increased. However, there is also a trade-off between computing power and accuracy which is not present in the reservoir sampling method. Namely, for each added hash function, an extra hash computation must be performed for every query and addition of new items into the sketch. The reservoir sampling method has no such requirement. In these particular instances, computing the reservoir samples takes about 25 seconds for any valid configurations, however creating the $20 \times 20$ COUNT-MIN sketch took 52 seconds, and the $50 \times 50$ configuration took 84 seconds to construct.

## 3 Flow Data Discretization

We used a similar method as [1] to perform discretization on scenario 10. Before we discretized two features into one code, we first analysed the dataset. As seen in Figure 1(a)(b) the benign hosts seems to have larger bytes size than the infected ones with an outlier of around 2,500,000 and for the infected ones around 700,000. Computing the actual values of these boxplots gives us: (min - 60), (25% - 66), (50% - 297), (75% - 2,046), (max - 24,972,470) for benign and (min - 60), (25% - 1,066), (50% - 1,066), (75% - 1,066), (max - 788,743) for infected. This shows that most of infected netflows have constant bytes size (= 1,066) while the benign netflows have more variance and larger outliers, but generally lower sizes for at least 50% of the data. From this, we can therefore conclude that most of the infected bytes sizes are larger than the benign ones, with a constant of 1,066.
In Figure 1(c)(d) we see the frequency of each protocol type. Here we see that benign hosts mostly use TCP followed by UDP and the infected hosts mostly ICMP followed by TCP.

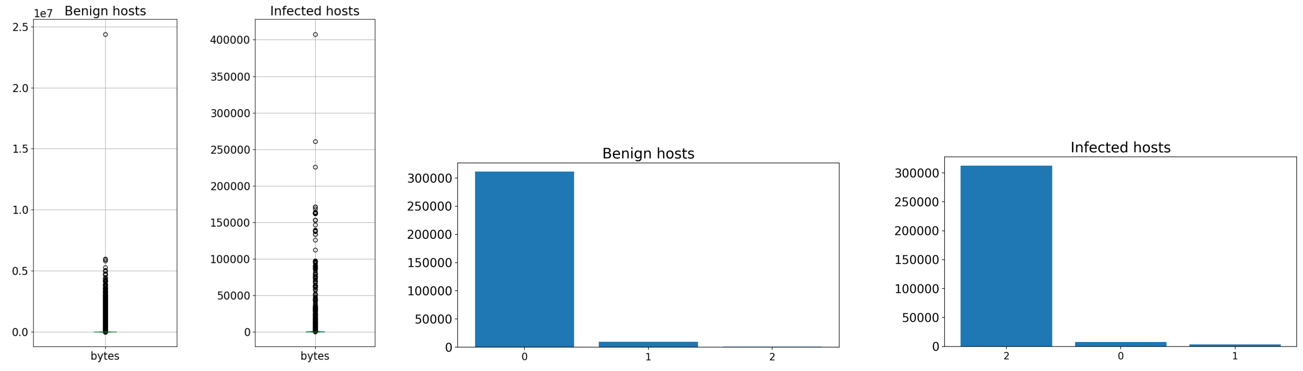
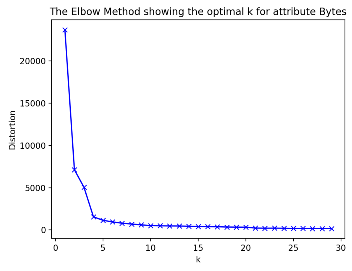


Figure 1: Visualizations of the two features: bytes and protocol type (0: TCP 1: UDP 2: ICMP)

With these two features we applied discretization for one infected host (= main host). Before we can do this, we first did a factorization for the 'protocol'. Since the 'bytes' contain too many values for factorization, we applied a k-means classification where we used the ELBOW method to determine the number of clusters [1]. In our case, this was 4. Based on the determined cluster size, we segmented each of the attribute values. E.g., for 'bytes' we determined a cluster size of 4 which means we segment them with $v = 0$ if the values are between $0\% - 25\%$ of the possible values, $v = 1$ if $26\% - 50\%$, etc.



| Code | Main host | Code | Infected | Code | Benign |
|---|---|---|---|---|---|
| 11 | 18,125 | 11 | 292,919 | 1 | 308,010 |
| 10 | 1,114 | 1 | 7,222 | 6 | 9,643 |
| 1 | 347 | 5 | 1584 | 0 | 2102 |
| 5 | 318 | 9 | 639 | 11 | 1014 |
| 14 | 75 | 8 | 612 | 2 | 679 |
| 13 | 58 | 6 | 344 | 3 | 300 |
| 12 | 35 | 7 | 27 | 4 | 169 |
| 6 | 20 | 12 | 2 | - | - |

Now that both features are numerically categorized, we applied the encoding technique.
In a similar fashion we discretized the benign hosts and the infected hosts with the main host excluded. This allowed us to compare all three to observe the behaviour of the discretized features (see above). From this, we observed a somewhat similar pattern for the infected main host and the other infected hosts, where code 11 appear for both most often. Also, the benign hosts have a maximum code of 11 but mostly values below 6 while the infected ones have mostly codes above 6. These higher code values can be explained from the slightly higher bytes sizes and the use of UDP - which has value 2 - as seen in Figure 1. Thus, we can conclude that codes higher than 6 most likely are infected hosts and below 6 benign.

## 4 Botnet Profiling

The profiling task is inspired by the provided paper [1] where scenario 10 of the available datasets has been used. Upon loading the dataset we noticed that Pandas had difficulty with loading the dataset correctly due to the tabs in the raw file. We removed these tabs and converted each attribute into numeric values. For 'flags' and 'protocol' we applied factorization and for 'duration, 'packets', 'bytes' we used the clustering-and-elbow-technique as described in Section 3. We then used the encoding technique to reduce the features into one feature, called 'code'.

### 4.1 Three data subsets

Similar to [1], we split the dataset into three sets: the configuration set, training set and validation set.
**Configuration set:** As [1], the configuration set is 30% of the total number of benign netflows in the dataset. This set was used to determine the cluster size for the attributes 'packets', 'bytes', and 'duration'. Moreover, the configuration set was also used to determine the threshold. This threshold is used to determine if a host in the validation set is infectious. [1] stated that the threshold is the error difference between the main profile (= training set) and the other profiles (= validation set); with $\mu + 2\sigma$ of the error. We thus built a Gaussian HMM-model (= HMM) on one benign host, and predicted this model on the other benign hosts in the set to determine the error difference. This resulted in a threshold of around 0.45. This means that hosts in the validation set with an error difference $> 0.45$ are infected.
**Training set:** The training set is an infectious host where the HMM will learn its behaviour of.
**Validation set:** The validation set is the left-over set and is used to assess whether the hosts are similar to the infectious host.

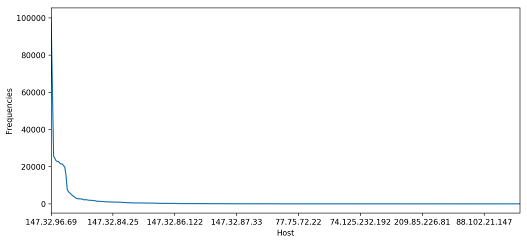


Figure 2: Frequency of the netflows per host. Most hosts have almost no netflows

### 4.2 Machine learning model

We used a Gaussian Hidden Markov model (= HMM) as the probabilistic sequential model. For this, we set a sliding window of 20 as in [1]. Moreover, we observed a large number of hosts containing only a small number of netflows (often with only 2). Because this is a behavioural profiling task, the model cannot derive a fair behavioural pattern in the host with only 2 netflows. Also, the runtime of the model became unnecessarily slow. Based on Figure 2 and the paper we therefore chose the 380 hosts with the most netflows. To evaluate which hosts in the validation set are infectious, we computed the confusion matrix.

$$\begin{bmatrix} TN & FP \\ FN & TP \end{bmatrix} \begin{bmatrix} 198 & 154 \\ 0 & 28 \end{bmatrix}$$

We see that the model is able to identify all the 10 infected hosts as mentioned in the provided dataset with zero False Negatives. Aside from this 10, it found other infected hosts where there seems to be a high usage of certain protocol types and a certain bytes size. However, it still has a large number of False Positives indicating that it misclassifies most benign hosts as infectious.

## 5 Flow Classification

### 5.1 Data preprocessing and feature selection

In order to evaluate this method with the method in Section 4 we used the configuration- and training set as the training set for this task and the validation set as the test set for this task. This however also resulted in a class imbalance in the training set, so after we split this new training set into a train-and validation set we applied the SMOTE technique on the training set.
Paper [2] evaluated a few algorithms and we therefore decided to take the most common features that were mentioned for each algorithm. Since this task focuses on each individual netflow, we removed 'datetime', 'source', 'port' and 'destination' because they would hold too much information leading to data leakage. Eventually we shuffled the data that contains the attributes: protocol, flags, bytes, duration, packets.

### 5.2 Machine learning model

For this classification task we used a K-nearest neighbour classifier with the default values as parameters. We computed for the validation set and test set the confusion matrices:

$$\begin{bmatrix} TN & FP \\ FN & TP \end{bmatrix} Validation \begin{bmatrix} 111526 & 1208 \\ 35 & 6935 \end{bmatrix} \quad Test_{netflow} \begin{bmatrix} 223061 & 2194 \\ 5854 & 297476 \end{bmatrix}$$

We see that the classification can be done quite well for the validation set. The test set has almost a similar number of False Positives, but has a larger number of False Negatives. So for the model it is more difficult to correctly classify an infected netflow. Because this classification task is on the packet level and not on host level, the values cannot immediately be compared with the one in Section 4. We thus aggregated the flows corresponding to the hosts and recomputed the confusion matrix. If there is a host with at least one infected netflow, we assign the host as infected:

$$\begin{bmatrix} TN & FP \\ FN & TP \end{bmatrix} Test_{host} \begin{bmatrix} 251 & 81 \\ 30 & 18 \end{bmatrix}$$

This results in a classifier that found 99 infected hosts with the majority misclassified. This is due to the host-level conversion. There are some cases where a host only contains one infected netflow. In that case a host is immediately assigned as an infected host. You could however raise the question whether this is reasonable and if the complete host with 100 netflows should be classified as infectious if it only contains one infected netflow. This can also be seen in the misclassification where the confusion matrices has a ratio FP:TP much higher for the host-level than for the packet-level. In that case, a packet-level is ideal because it will look at each netflow individually and is better at classifying them.
If we compare the host-level confusion matrix with the one in Section 4 we see that the HMM has zero FN, but a larger FP. In our case, we deem a low FP more important, but the results for both cases are not ideal. While the HMM can detect all the infected hosts, it is problematic that half of the benign hosts are misclassified as infected. This can lead to a system where the user would unnecessarily judge a host as infected. On the other hand, the classification task has a large number of FN meaning that it will let some infected hosts pass as benign. We would therefore choose the HMM because we deem it more important to find all the infected hosts correctly than benign hosts misclassifying as infected.

## References

[1] C. H. Gaetano Pellegrino, Qin Lin and S. Verwer, "Learning behavioral fingerprints from netflows using timed automata."

[2] e. a. Garcia, Sebastian, "An empirical comparison of botnet detection methods," in *Computers and Security*, 2014, pp. 100–123.