Stream data is a continous flow of data that builds up rapidly. To be able to process and quickly judge the individual flows, discretization can be applied. Discretization breaks the very high number of degrees of freedom for each feature into small groups/categories/bins such that is easier to understand and process the data. Another advantage of discretization is that not every model is able to work with continuous data. For each feature the values are split into a number of bins (N). Bins can be filled according to multiple methods: uniform, quantile and kmeans. Some data is more suitable for one method than the other, e.g. kmeans is good applicable if the data consists of several clusters and uniform (equal-width bins) if the data is equally distributed.
After processing the data the first thing to find out is which features are most suitable for discretization. Looking at the numbers we quickly see that the duration, sourcebytes, and protocols (see Figure 1) are quite different between the infected and non-infected flows. After further investigation we find that the protocol and duration feature looks most promising. For the infected hosts almost all flow uses the ICMP protocol, while the non-infected hosts mostly use UDP and TCP. The duration of a flow for infected hosts is almost twice as long as for non-infected hosts. There is not really a suitable plot method to visualize the difference for the duration feature.
To find out the optimal number of bins to use for descretization we can apply the Elbow method [1]. We select an infected and non-infected host to discretize the features. Before we can apply this method, the protocol feature must be encoded in numerical values. After applying the Elbow method we obtain that the protocol feature is optimal with 3 bins and the duration feature with 4 bins (see Figure 2,3). We can now descritize both features with the obtained bin sizes. Afterwards, we can concatenate both discrete values for both features to obtain a single discrete value (see Figure 4). We see that for the infected host almost all values are '00' and for the non-infected host '10' and '20' are most popular. In Figure 5 we have applied the same discretization on all hosts in the dataset. The results are almost the same, although the non-infected flows also include some other discrete values. We can thus clearly see the effect on discretization, which can easily and quite fast be implemented to be used for infection detection. For every incoming flow both features can be discretized and concatenated to be able to make a judgement on the infection status. However, this does not directly mean that if the discretized features results in a value of '00' the host is infected. Discretization should be combined with more features or more separate discretized features to be able to make a judgement with a higher success rate.
[1] Learning Behavioral Fingerprints From Netflows Using Timed Automata. Gaetano Pellegrino, Qin Lin, Christian Hammerschmidt and Sicco Verwer

In spacesaving.py the implementation of the SpaceSaving algorithm can be found. The SpaceSaving algorithm is used to approximate the distributions over 3-grams of the discretized symbols. As can be seen in task 1, the Protocol and Duration feature are discretized into a single value.
The top 10 3-grams are approximate for two datasets. The first dataset only contains non-infected hosts and the other data set contains only infected hosts. To test the influence of the number of counters used, the algorithm is run using 10, 20, and 30 number of counters.
Above the results are listed. For each result, the top 10 n-grams with their real count are shown. Next to that, the top 10 n-grams found by the SpaceSaving algorithm and their approximate count are listed. And the last row represents the accuracy ratio (approximate count / real count), computed if and only if, the n-gram is listed at the same position otherwise the counts of different n-grams are compared.
For the infected data set, the algorithm performs very accurate. 000 is the most occurring n-gram and has occurred ten times more than the second most occurring n-gram. As can be seen in the results, the top 10 n-grams are almost perfectly approximated. The number of counters used does have an impact on detecting the most frequent n-grams. When 10 counters are used, we are only able to perfectly match the top 6 most frequent n-grams. When a larger number of counters is used this number increases.
The results of the approximation of the n-grams in the non-infected data set are slightly worse compared to the infected dataset since the last n-grams are not ranked according to the real counting. The reason for this is that this data set is more varied. In this data set, the n-gram 777 is the most occurring n-gram. From the results concluded could be, that the SpaceSaving algorithm is a good method to approximate the most frequent elements.

The approximation errors could be explained using the mathematical theory as explained in the slides. As defined in the slides "Any item x whose true count > m/k is stored" where m is the sum of the counters and k the number of counters. This can easily be checked, using the following lines of code: Above it can be seen that every item that occurs more than m/k is included. Approximation errors could occur when, for example, the last item in the stream only occurs once. In the end, the last n-gram will be replace the value with the loweset minimun counter value. However, this n-gram only occurred one time while the value with the minimum counter value could be way higher. This is one example of how approximation errors could occur. However the approximation is upper bounded by m/k, therefore this approach is an interesting way of approximating distributions of data using limited memory.

COUNT-MIN sketches are very good applicable to continuous streams of data. They reflect an approximation of the real data in the streams in sub-linear space, in which the approximation differs because of the chosen variables of width and depth. As with discretization, it uses only a fraction of the distinct values. The width and depth (number of distinct hash functions) variables depend on the error probability (δ) and error factor (ε). The width is equal to the inverse of the error factor ($1/\varepsilon$) and the depth is equal the logarithm of the inverse of the error probability ($\log(1/\delta)$). The hash functions must be pair-wise independent to prevent hash collisions in the profiles. This independence is preserved using $g_i(x) = h_1(x) + i*h_2(x)$, described in [2]. The 3-gram profiles are built by concatenating the discretized values of the protocol feature of a row and its two predecessors.
We expect that the higher the width and depth, the less false positives should appear because the probability of a collision is lower. Simply choosing a very high width and depth contradicts the use of sketching since this method aims to lower the width and depth. We therefore must find a good trade-off in false positives and width and depth.
What we observe from the results is that for a width of 10 the number of false positives is quite big (see Figure 6). Increasing the width from 20 to 50 increases the accuracy significantly. For a width of at least 100 the number of false positives is almost zero. Increasing it even more does not make sense much since not much improvement is achieved compared to the increased memory usage. We would like to choose a width and depth that is memory efficient with a low number of false positives. The biggest relative accuracy increase (and not too many FP) is obtained with a width of 100 and a depth of 4, 'only' storing 400 values. Using the chosen width and depth, we obtain the top 10 with their frequencies. We see that "000" is clearly the most frequent 3-gram, with a fraction of collisions in about 100K occurences.
As said earlier, the COUNT-MIN sketch approximation is probablistic, which increases the accuracy by the use of a higher width and depth. The lower the width and depth, the more approximation errors arise. These approximation errors originate from collisions in the (3-gram) profiles. The more hash functions are used, the lower the probability that there exists hash collisions (overlap) with the same profile. However, this does not mean that the width and depth are independent from each other. If a lot of hash functions are used in combination with a low width, the probability of the same profiles as a result increases.
[2] Less Hashing, Same Performance: Building a Better Bloom Filter. Adam Kirsch, Michael Mitzenmacher.

Above the implementation of the min-wise LSH can be found. First, the minhash matrix is created where the rows represent the ip-pairs and the columns the n-grams. The cell value is set to 1 if the n-gram exists once in the stream data of the current ip-pair. To keep a good overview, this algorithm is only run for the non-infected hosts. This could also easily be done for the infected-hosts. The last printed line shows the runtime comparison. The total time needed to compute all pair-wise distances between all possible ip-pairs is divided by the runtime needed when only the candidate pairs found by the min-wise LSH algorithm are used, to see how much faster the min-wise LSH algorithm is. As can be seen, the algorithm is approximately 4.5 times faster. This is a huge performance increase, especially if you are dealing with much more data.
In total there are twenty hash functions used to imitate the permutation of the rows of the signature matrix. The parameters of the banding technique are set to, bands = 5; rows = 4; bins = 15. These parameters are tuned by looking at the results until a decent result was achieved.

In the list above the first 30 ip-pairs that were mapped to the same bucket are printed(for illustration only, the second bucket is shown). In the second column, the original similarity between the ip-pairs is listed, next to that the approximated similarity is shown. In the final column, the difference between the beforementioned columns is shown.
As can be seen, the approximated similarity is quite close to the original similarity for all cases. This showed that minhasing is able to accurately calculate the similarities between ip-pairs by using the signature matrix.
Remarkable is that these similarities are relatively low. A reason for this could be that the ip-pairs are not quite similar. To verify this, the algorithm could be run on the infected-hosts and see what the result is. Another reason could be the tuning of the number hash functions, band size, and amount of bins used in the algorithm. Because of the limited time, I was not able to test if the parameters used found the optimal buckets and similarities.

We first load and process the dataset of scenario 10. As in the tasks before we use the discretized Protocol feature from the first task. We group the unique ip-pairs of the dataset, to be able to count the 3-gram occurences of this discretized feature to create a profile. We set the bin size to 4 (2 bits) since choosing more (next number of bins is 2^3=8) result in unused bins. We also choose to project (apply a set of random hyperplanes) the 3-gram occurence counts three times to get a more reliable result than only using one projection. For one projection the probability of a pair being hashed to a wrong bin is higher, and the combined result of multiple projections decreases this probability significantly. In the Table 5a we see the combined bins of the three projections in the last column. We use this result to check the frequencies for the infected, normal and unknown hosts. We see that for the infected hosts about 64% results in a combined_bins that only occurs in about 16% of the cases for the normal hosts. We also see the opposite in which the 82% of the combined_bins for the normal host occurs in only 3% of the infected hosts. This indicates that, based on the combined_bin profile of the distinct projections we can make a good prediction on whether a host is infected or normal. For the unknown hosts we see that about 91% will result in being indentified as infected and the other 9% as normal.
Applying LSH has an advantage in run-time, in which the result of binning offers a significant speed-up. If we do a pair-wise (Euclidean) distance computation between all pairs (on the 3-gram profiles) for this dataset, we approximately need 1.9 seconds. If we only compute the distance between the pairs that are assigned to the same bin, we only need less than 1.2 seconds. This is an increase of more than 50%, which is important if dealing with a continuous stream of data. These numbers are however based on the calculation of all pair-wise combinations and all bins, which is not the case in a stream. In that case we only need to calculate the distance of that item to all other items or the bin the item is assigned to.

We preprocess the data for scenario 9 to 12 and discretize the protocol feature the same way we did as the previous tasks. Scenario 9 will be the 'training' scenario, in which we will choose a suitable threshold for the distance calculation in the other scenarios. This threshold separates the new predicted infected hosts from normal hosts. First an infected source/host is selected from scenario 9, in this case with the most netflows in the dataset to train as best as possible. We then calculate all 3-grams in the flows from this host and count the occurences, resulting in a profile for this host. Instead of using a timed sliding-window, the unique hosts are grouped and profiles are created from counts of 3-grams. The choice for 3-grams is based on the fact that the possible combinations of 2-grams are quite low (only 4 combinations) and there are not enough flows originating from the unique hosts for using 4-grams.
For each unique host we calculate the 3-grams profile the same way as we did for the selected host, and additionaly we calculate the cosine distance in relation to the selected host. At the end of this iteration the distance between the initially selected host and all unique hosts is determined. We must choose a threshold that separates the infected and normal hosts as correctyly as possible and additionaly optimizes the number of newly identified infected hosts. Part of the hosts in the dataset are labeled as infected or normal, while there is a part that is not labeled yet and thus to be identified. A grid search is applied for a threshold of 0 to 0.25 (cosine distance). The results show that with a

threshold of 0.18 there are 12 (out of 19 unidentified) new infections identified at cost of a couple of true positives and false positives, dropping the accuracy to 62.5%. The false positives indicate hosts that are labeled as infected while they are not. It is thus important to keep this number as low as possible in comparison to the number of newly identified infected hosts.

With this chosen threshold we can do the same for the other three scenarios. We find that the results with this threshold do not deviate much for the other scenarios. We identify zero, zero and six newly identified infections to respectively scenario 9, 10 and 11, at cost of one, one and five false positives. A low number of new infections does not directly indicate that the profiling performed bad, since we don't know how many infected hosts there actually are among the unknown hosts.
We can thus conclude that by profiling using n-grams some new infected hosts can be detected. However, the more the threshold is on 'edge', the more this may result in an increase in false positives. This is a trade-off someone should be willing to make (or not).

For the fingerprinting task we again use the n-gram profiles for the hosts, based on the 3-gram occurence counts. In the botnet profiling task the profiles were used to estimate the distance between a selected host and other hosts, in combination with a threshold that labels a host infected or legitimate. In this task we do not use the distances to distinguish the hosts, but instead rely on the fact that some 3-grams do not occur for benign hosts and do occur for infected hosts.
We first exclude the hosts which label are unknown/undetermined. Then we split the set in a train and test set. For the train set we count all occuring 3-grams for the distinct hosts and sum them together (for infected and benign hosts separate). Afterwards, we can easily see which 3-grams do not occur for benign hosts and do occur in infected hosts. We then create a fingerprint from those 3-grams that satisfy that condition. These 3-grams thus decide whether a host is labeled infected or not. For the test set the same process is repeated, but now we count the number of false positives (and other metrics). The unknown hosts are now being used to find possible new infections, based on the fingerprints determined earlier. If for these hosts not all 3-gram occurence counts are zero for the 3-grams in the fringerprint, we label the host as infected and otherwise as legitimate.
If we look at the results we see that it differs quite a lot between the different datasets. However, we see that it results in detection of new infections, at cost of not too many false positives. We also see that for scenario 10 this method does not work properly since all possible 3-grams appear among the benign hosts. Because if the benign hosts act very differently, the fingerprinting method is almost unusable. Possible modifications may be to allow a couple of occurences of 3-grams to account for this problem.
If we compare the results to the results of the botnet profiling task, we see that the results of profiling is slightly better. Its overall accuracy is about 10% higher, the number of detected new infections is better in one scenario, but the number of false positives is also a lot higher (although we must note that the number of hosts included in the fingerprinting test set is about 3 times lower). In the end the results are not extremely different if we consider the same size of test set, and both are thus applicable for the same purpose. As mentioned earlier, the basic difference between these methods is that profiling measures the distance between profiles while fingerprinting builds upon the fact that some 3-grams do not occur in a type of host.