MalPaCA: Malware behaviour analysis using unsupervised machine learning

Comparative analysis of various clustering algorithms on determining the best performance in terms of network behaviour discovery

Hugo de Heer

This paper is written as part of the academic course CSE-3000 (Research Project)

Supervisor: Azqa Nadeem
Responsible professor: Sicco Verwer
Delft University of Technology
The Netherlands
16 June, 2021

Abstract

MalPaCA makes use of unsupervised machine learning to provide malware capability assessment by clustering the temporal behaviour of malware network packet traces. A comparative analysis was performed on various clustering algorithms to determine the best clustering algorithm in terms of network behaviour discovery. The clustering algorithms included in the analysis were HDBSCAN, OPTICS, Agglomerative Hierarchical Clustering and K-medoids. Metrics that capture cluster separation, cohesion, purity and completeness were used to determine the performance of the clustering algorithms. Agglomerative Hierarchical Clustering had the lowest total error of 0.950 in the comparative analysis compared to the baseline HDBScan with an error of 1.381.

Contents

1 Introduction

1.1 Background of the research

In this era of modernization, digital technology plays a major role in all facets of life. As the use of the digital space has grown, so have the prospects of internet crime by creating malware. Nowadays, malware is one of the leading threats to security. Anti Virus (AV) communities provide a key role in threat management by detecting malware and assigning labels to them. However, an extensive study has shown that a huge heterogeneity in their categorization has been observed [1]. Since most AV companies work in a competitive fashion instead of a collaborative one, AV companies have their own way of labelling malware. The main problem is that family malware labels do not provide any insight into the capability of malware samples. To tackle this problem, MalPaCA [2] was developed. It makes use of unsupervised machine learning to provide malware capability assessment by clustering the temporal behaviour of malware network packet traces. In this way, MalPaCA aims to construct a behavioral profile for each malware sample that is more descriptive than its family label.

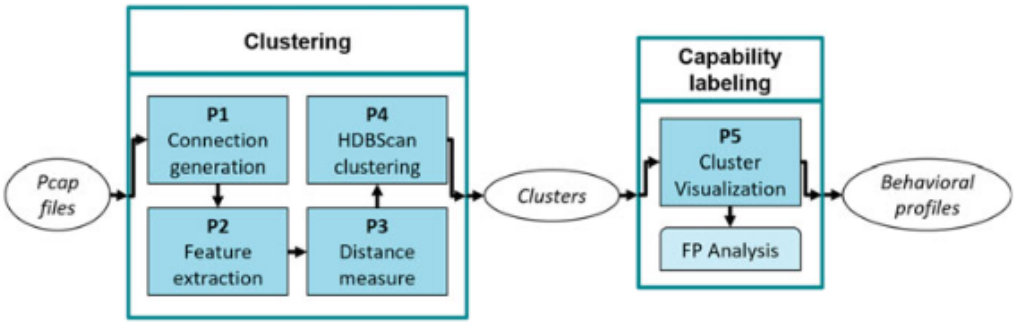

Figure 1: Pipeline of MalPaCA.

1.2 Working of MalPaCA

MalPaCA takes PCAP files with network traffic as input to the system. It splits those network traces into unidirectional connections and the following 4 features are extracted for every unidirectional connection: Packet size, time interval between packets, source port and destination port. These features aim to capture the temporal behaviour of the connection and so they are sequential features. Then similarities between these connections are quantified by calculating the distances between each other. Dynamic Time Warping [3] is used for calculating the distances between the numeric sequences (packet size and time interval), Ngram analysis [4] is used to calculate distances between categorical distances (source port and destination port). This all results in a distance matrix where all similarity scores are listed between each connection. The connections are then clustered using the HDBScan clustering algorithm [5] that takes as input a distance matrix. The result of the clustering is visualized by making use of temporal heatmaps showing the connections' feature values inside a cluster. Figure 1 summarizes the working of MalPaCA.

1.3 Research question

This report presents a comparative analysis of various clustering algorithms to be able to compare them to each other in terms of clustering network behaviour discovery. Hence, the main research question that we aim to answer in the research is:

RQ: Which clustering algorithm has the best performance in terms of network behaviour discovery?
With the best performance in terms of network behaviour discovery, we aim to capture unique behaviours present in network traces and allocate them to separate clusters. So ideally, one cluster represents one behaviour. Answering this question can give more insight into the different clustering algorithm's ability to capture network behaviour. Furthermore, it can improve MalPaCA by providing better clustering results in the form of more pure and complete clusters that distinguish between behaviours which results in better malware capability assessment. To be able to answer this question, we define the following subquestions:

RQa: Which clustering algorithms are chosen for the comparative analysis?
To be able to determine the best clustering algorithm, a careful selection of possible candidates should be made. This is done by a literature study on various clustering algorithms that have promising results. More detailed information about this in section 'Clustering Algorithms'.

RQb: What metrics capture the cluster quality?
All the clustering algorithms should be tested against some metrics to analyse their performance in terms of network behaviour discovery, answering the above question will define those metrics to be able to perform the comparative analysis. This will be explained in the section 'Methodology'

RQc: How to tune the parameters of the clustering algorithms to optimize performance?
A significant part of the comparative analysis consists of tuning the clustering algorithms' parameters to optimize the score on the defined metrics by RQb. The tuning selection of each algorithm will be explained in the section 'Experimental Setup'.

RQd: What are the differences and similarities between all the clustering algorithms in terms of network behaviour clustering results?
Some clustering algorithms take a precomputed distance matrix as input that defines the distances between connections. One could also consider the distance matrix (n, n) as n connections with n features, where feature i of a connection represents the distance to the connection on row i in the matrix. Comparing those two ways of clustering will be analysed. To properly answer which clustering algorithm has the best performance, differences and similarities between all algorithms should be analysed and discussed. Interesting results will be analysed using by using the temporal heatmaps. This will further be explained in the section 'Discussion'.

1.4 Outline of the paper

The section 'Methodology' will explain the steps that were taken to set up the research. It will discuss the dataset and the metrics that have been used. The section 'Clustering Algorithms' will explain which algorithms were chosen for the comparative analysis and why they were chosen. The section also aims to explain how the algorithms function. The section 'Experimental Setup and Results' provides the evaluation setup and how the tuning of the clustering algorithms is performed. The results of all the clustering algorithms are also shown here. The section 'Discussion' aims to compare all the results and reflect on the explainability of the results. And finally, the 'Conclusions and Future Work' section answers the research questions and lists possible improvements that could be made.

2 Methodology

This section will explain the steps that were taken to answer the research question. It will discuss the dataset and the metrics that have been used. Also, MalPaCA's refactoring and extension to realize this research are stated here.

2.1 Dataset

A labelled dataset with malicious and benign IoT network traffic has been used to properly determine the performance of the clustering algorithms. Aposemat IoT-23 [6] is a new dataset of network traffic from Internet of Things (IoT) devices. This dataset consists of 23 captures of different IoT traffic. These captures are split up into 20 scenarios from infected IoT devices and 3 scenarios of real IoT devices network traffic. Next to the PCAP files that contained the network captures, labels were provided to describe the relation between flows related to malicious or possible malicious activities. These labels were given in the form of log files that assigned labels to packets sent over the network. In this way, cluster behaviour can be analysed by giving unidirectional connections a label. Figure 2 contains the distribution of malicious labels exhibited by the IoT traffic.

2.2 Extension and refactoring of MalPaCA

To be able to run MalPaCA on the IoT-23 dataset, a speedup of MalPaCA was required, since it would take several hours to execute on a fairly small amount of data (500 connections). This was mainly because of the high complexity of the Dynamic Time Warping (DTW) to calculate distances between the numerical features and the cosine distance calculation in the categorical features. To tackle this problem, Numba [7] was used to provide a significant speed up to the DTW and cosine distance calculation. Numba is an open-source JIT compiler that translates a subset of Python and NumPy code into fast machine code.

Since the IoT-23 was very large in general (over 20GB of PCAP files), a subselection of the whole dataset had to be made. After using Numba as a speed-up library, MalPaCA was able to accept around 2500 connections as input to return a feasible execution time of around 10 minutes for the experiments. This was mainly because of the quadratic complexity of calculating the distance matrix of (n, n) connections. Being that MalPaCA only analyses the first 20 packets of each connection, all connections that were shorter than 20 packets were left out. Each PCAP file was read and only the first 20 packets of a connection were selected and combined with the corresponding malicious labels in the log files. This resulted

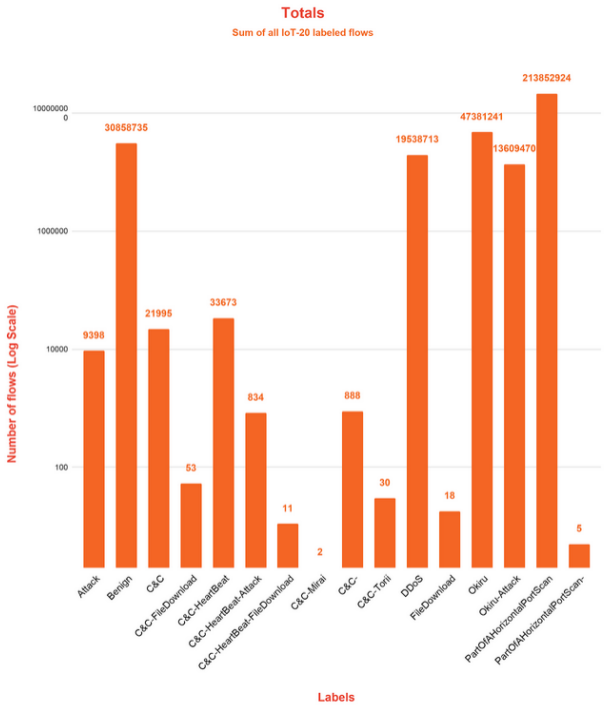

Figure 2: Labels distribution of Aposemat IOT-23 labeled flows in log scale.

in a dictionary for each PCAP file where the key was a connection (defined by a source Ip address and a destination Ip address) and the value was a list of packet information that contained the 4 sequential features and the malicious label provided by the dataset. These dictionaries were then serialized using the Python pickle module to provide a straightforward way to save the labelled dataset in a compact format.

2.3 Metrics

To evaluate and compare the clustering algorithms on their performance, we can define metrics that measure the cluster quality. Metrics can be categorized into two groups according to whether ground truth is available [8]. If there is a ground truth, it can be used by extrinsic metrics that compare clustering results against the ground truth. Intrinsic metrics evaluate clusters based upon their separation and cohesion. In this case a ground truth is available, namely the malicious behaviours specified by the dataset in the form of labels.

Cluster homogeneity and cluster completeness are two essential criteria when aiming to define the cluster quality [8]. Cluster homogeneity requires that the more pure the clusters in a clustering are, the better the clustering is. So ideally, all connections in a cluster should have the same label. Cluster completeness is the counterpart of cluster homogeneity and states that if two connections have the same label, they should be clustered together. The metrics defined below aim to satisfy both criteria.

All metrics return an error score lying between 0 - 1 where 1 indicates a high error and 0 a low error.

2.3.1 Silhouette

The Silhouette Coefficient is an intrinsic metric that measures how similar objects are to their own cluster compared to other clusters. In this way, it captures cluster cohesion and cluster separation. The Silhouette Coefficient is defined for every data point (in our case a connection) and it is composed of two scores:

- a: The mean distance between a data point and all other points in the same cluster.
- b: The mean distance between a data point and all other points in the next nearest cluster.

The Silhouette Coefficient for a single data point i is then given as:

$$s(i) = \frac{b(i) - a(i)}{max(a(i), b(i))}$$

This metric returns a score lying between -1 and 1, where a high value indicates that a sample is well matched to its own cluster. The average over all samples has been taken and that score been converted to an error metric.

The Silhouette Error for all data points p is then given as:

$$sErr(p) = \frac{1 - (\frac{1}{|p|} \sum_{i \in p} s(i))}{2}$$

2.3.2 Cluster purity

The cluster purity metric is an extrinsic metric that aims for full benign or full malicious clusters. When all clusters are 50% malicious, clusters are impure and it will give a high error of 1. When all clusters are pure, so 0% malicious or 100% malicious, it gives a low error of 0.

We define the maliciousness mal of a cluster where m is the set of all malicious connections:

$$mal(c) = \frac{|c \cap m|}{|c|}$$

We define the cluster purity error for a cluster c as where mal is the percentage / 100 :

$$cpErr(c) = \begin{cases} mal(c) \cdot 2, & mal(c) \leq 0.5 \\ (1 - mal(c)) \cdot 2, & \text{otherwise} \end{cases}$$

The total error is then retrieved by taking the average $cpErr$ over all clusters.

2.3.3 Cluster malicious purity

The cluster malicious purity metric is an extrinsic metric that aims for clusters containing only one specific behaviour. If every cluster contains every possible malicious behaviour, it will give an error of 1. If every malicious cluster only contains 1 behaviour, it will return an error of 0. cb is defined as the set of all malicious behaviours present by the connections in a cluster. b is defined as all malicious behaviours present in the dataset.

We define the cluster malicious purity for a cluster c as:

$$cmpErr(c) = \begin{cases} \frac{|cb|}{|b|} & |cb| \geq 2 \\ 0 & \text{otherwise} \end{cases}$$

The total error is the average of $cmpErr$ over all clusters that contain malicious behaviour.

2.3.4 Noise error

Some clustering algorithms allocate points to noise clusters if they detect that they do not belong to any cluster. These noise clusters are left out in the calculations of the other metrics. To minimize the number of connections placed in the noise cluster C_n a noise error is introduced that calculates the ratio of connections allocated to the noise cluster against the total amount of connections. The noise error on all data points p is defined as:

$$nErr(p) = \frac{|C_n|}{|p|}$$

2.3.5 Cluster completeness

Cluster completeness is an extrinsic metric that is the counterpart of cluster purity. It aims that if two connections have the same label, they should be clustered together. To calculate this, we consider all unique behaviours present in a cluster and count how many times each behaviour is present in all clusters. This results in a dictionary where the key is a label and the value is the number of clusters in which this particular label is present. Ideally, every label is only present once. Let v be the list of values from the dictionary and b be the set of all malicious behaviours present in the dataset. Then we define the cluster completeness error over all data points p as:

$$ccErr(p) = \frac{\sum_{x \in v} n(x)}{|b| \cdot 10}$$

The multiplication by 10 was performed to return an error score that mostly lies between 0 - 1.

$n(x)$ is defined so that this metric only gives an error if there is more than 1 cluster that contains a certain behaviour.

$$n(x) = \begin{cases} x & x \geq 2 \\ 0 & \text{otherwise} \end{cases}$$

2.3.6 Visualization clustering error

The visualization clustering error makes use of human visualization of the temporal heatmaps to determine if connections were clustered appropriately. We define a visualization clustering error ($VisErr$) as a connection that is placed in a cluster where more than half of its features differ from the remaining connections in that cluster. Two features are different if more than 50% of their sequence shows a different colour on the heatmap. Figure 3 shows an example of a $VisErr$. In this case features a, b and c from the connection highlighted in red were different from the remaining connections in that cluster. We can calculate the error rate of a cluster c by: $\frac{VisErrs}{|c|}$. Then the average of the error rates for each cluster is used as a cluster quality metric for comparing interesting results.

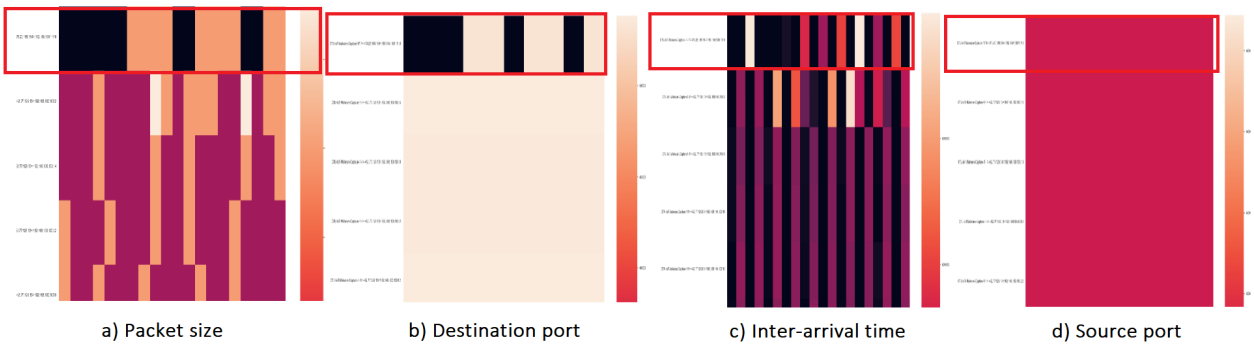

Figure 3: A visualisation clustering error: this connection is not clustered properly.

3 Clustering algorithms

This section will which algorithms were chosen for the comparative analysis. It also aims to explain how they function and why they were chosen.

3.1 HDBSCAN

HDBSCAN [5] is the clustering algorithm that is currently used in MalPaCA. It extends the popular DBSCAN [9] clustering algorithm by converting it into a hierarchical clustering algorithm, and then using a technique to extract a flat clustering based in the stability of clusters.
It has two major parameters that can be selected.
1. $min\_cluster\_size$ = smallest amount of points grouped together to form a cluster.
2. $min\_samples$ = provides a measure of how conservative the clustering will be. A higher $min\_samples$ indicates that more points will end up in the noise cluster.

3.2 OPTICS

OPTICS: Ordering Points To Identify the Clustering Structure [10] is another density-based clustering algorithm. It is similar to DBSCAN, except that it addresses one of DBSCAN's major weaknesses: the problem of detecting meaningful clustering in data of varying density. To do so, all points are ordered such that spatially closest points become neighbours in the ordering. OPTICS abstracts from DBSCAN by removing the parameter ϵ, which describes the maximum distance to consider other points, by giving it the maximum value
OPTICS requires one parameter to be tuned.
2. $MinPts$ = number of points required to form a cluster.

3.3 Agglomerative Hierarchical Clustering

There exists a familial structure among malware behaviours. Therefore it makes sense to use hierarchical clustering to model the relationships between them [11]. Agglomerative hierarchical clustering [12] performs a hierarchical clustering using a bottom-up approach: In this type of clustering, each data point is defined as a cluster. Pairs of clusters are merged as the algorithm moves up in the hierarchy.

It requires two parameters to be selected:
1. $linkage$ = which linkage criterion to use. The linkage criterion determines which distance to use between sets of observation.
2. $n_clusters$ = the number of clusters to find.

3.4 K-medoids

K-medoids [13] is a clustering algorithm that makes use of partitioning. In contrast to the k-means algorithm, k-medoids chooses actual data points as centres (medoids or exemplars) and thereby allows for greater interpretability of the cluster centres than in k-means, where the centre of a cluster is not necessarily one of the input data points (it is the average between the points in the cluster). This makes k-medoids also more robust to noise and outliers. It requires one parameter to be tuned:
1. k = The amount of clusters.

4 Experimental setup and Results

This section will go over the experimental setup that has been done to perform the comparative analysis and the corresponding results will be.

4.1 Experimental setup

A validation set was introduced to be able to tune the hyperparameters of the clustering algorithms. Due to the low frequency of some malicious connections with labels such as Okiru and C&C-Torii it was decided to split the full dataset into approximately two equal parts, a validation set and test set. In this way, those rare malicious connections were also included in the validation set. The validation set consists of a total of 1779 labelled connections, and the test set consist of a total of 1836 connections. Figure 4 shows a 2D projection of the validation and test set using t-SNE [14].

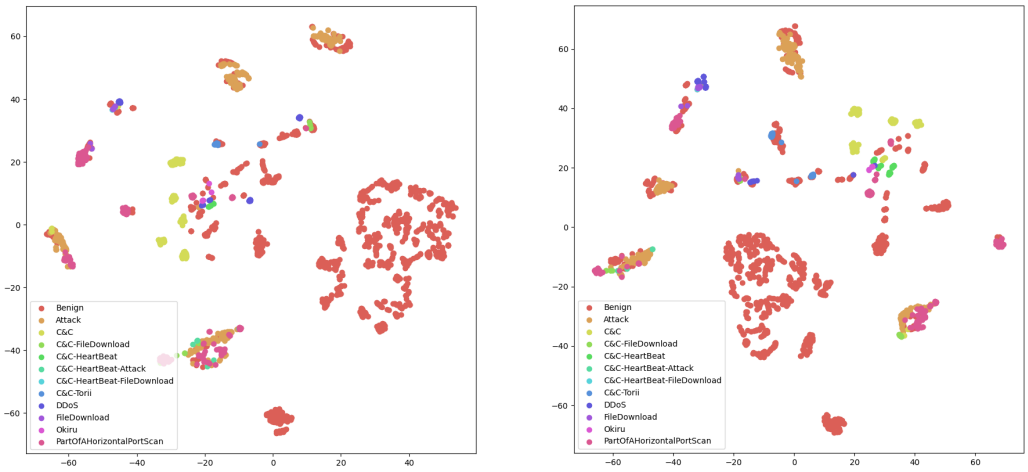

Figure 4: 2D t-SNE projection of the validation set (left) and test set (right).

All clustering algorithms are able to take a precomputed distance matrix as input that defines the distances between connections. A clustering algorithm running on the precomputed distance matrix is referred to as configuration a.

One could also consider the distance matrix (n, n) as n connections with n features, where feature i of a connection represents the distance to the connection on row i in the matrix. This configuration is indicated with the letter b. Due to the high dimensionality of configuration b, PCA [15] was applied to reduce the dimensionality of the distance matrix to 100 dimensions. This provides a balance between computation speed and retaining meaningful properties of the original matrix. All algorithms are tested on both configurations a and b. To be able to tune the hyperparameters of the clustering algorithms, a grid search of the parameters was performed on the validation set. To find the optimal parameters, the total error was minimized. This was defined by the sum of all metrics:

$$totErr = sErr + cpErr + cmpErr + nErr + ccErr$$

After tuning the hyperparameters, the clustering algorithms with the now optimized parameters were run on the test set where results were analysed.

4.2 Results of parameter optimization

HDBScan
For both configurations a and b, a grid search with $min\_cluster\_size$ ranging between 2-25 and $min\_samples$ ranging between 2 and 25 resulted in:
HDBScan a : $min\_cluster\_size = 10$ and $min\_samples = 2$.
HDBScan b : $min\_cluster\_size = 4$ and $min\_samples = 10$.

OPTICS
For both configurations a and b, a grid search with min_Pts ranging between 2-35 which resulted in:
OPTICS a : $min\_Pts = 24$
OPTICS b : $min\_Pts = 34$

Agglomerative Hierarchical Clustering
For configuration a, a grid search with $n\_clusters$ ranging between 2-100 and linkage criteria varying between *complete* and *average* was performed. Configuration b added the *ward* linkage criteria to a, since *ward* required euclidean as a distance metric and so this linkage criteria was not compatible with a precomputed distance matrix as used in a. The grid searches resulted in:
AHC a: $n\_clusters = 35$ and $linkage =$ average.
AHC b: $n\_clusters = 13$ and $linkage =$ average.

K-medoids
For For both configurations a and b, a grid search with $n\_clusters$ ranging between 2-100 resulted in:
Kmed a: $n\_clusters = 10$
Kmed b: $n\_clusters = 54$

4.3 Results of experiment

Cluster configurations	clusters	$sErr$	$cpErr$	$cmpErr$	nEr	$ccErr$	$totErr$
HDBScan a	58	0.311	0.094	0.081	0.125	0.770	**1.381**
HDBScan b	33	0.249	0.179	0.110	0.051	0.483	**1.072**
OPTICS a	20	0.424	0.161	0.171	0.418	0.378	**1.553**
OPTICS b	13	0.273	0.340	0.254	0.198	0.330	**1.395**
AHC a	35	0.221	0.172	0.080	0.000	0.475	**0.950**
AHC b	13	0.328	0.316	0.429	0.000	0.308	**1.381**
Kmed a	10	0.516	0.435	0.629	0.000	0.458	**2.038**
Kmed b	54	0.274	0.169	0.075	0.000	0.717	**1.245**

Table 1: Metric scores of all configurations.

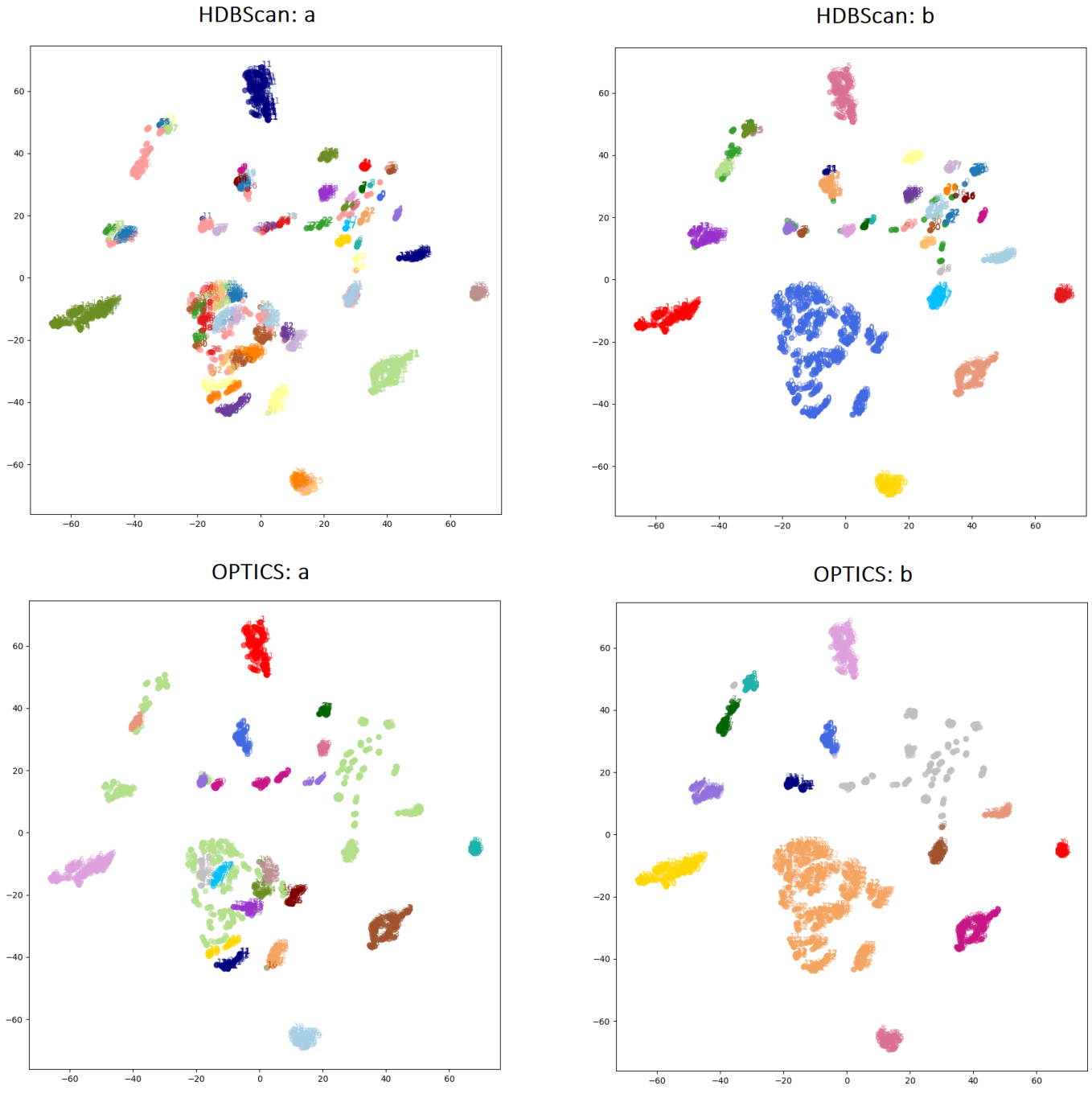

Figure 5: Clustering results of HDBScan and OPTICS for configurations a and b.

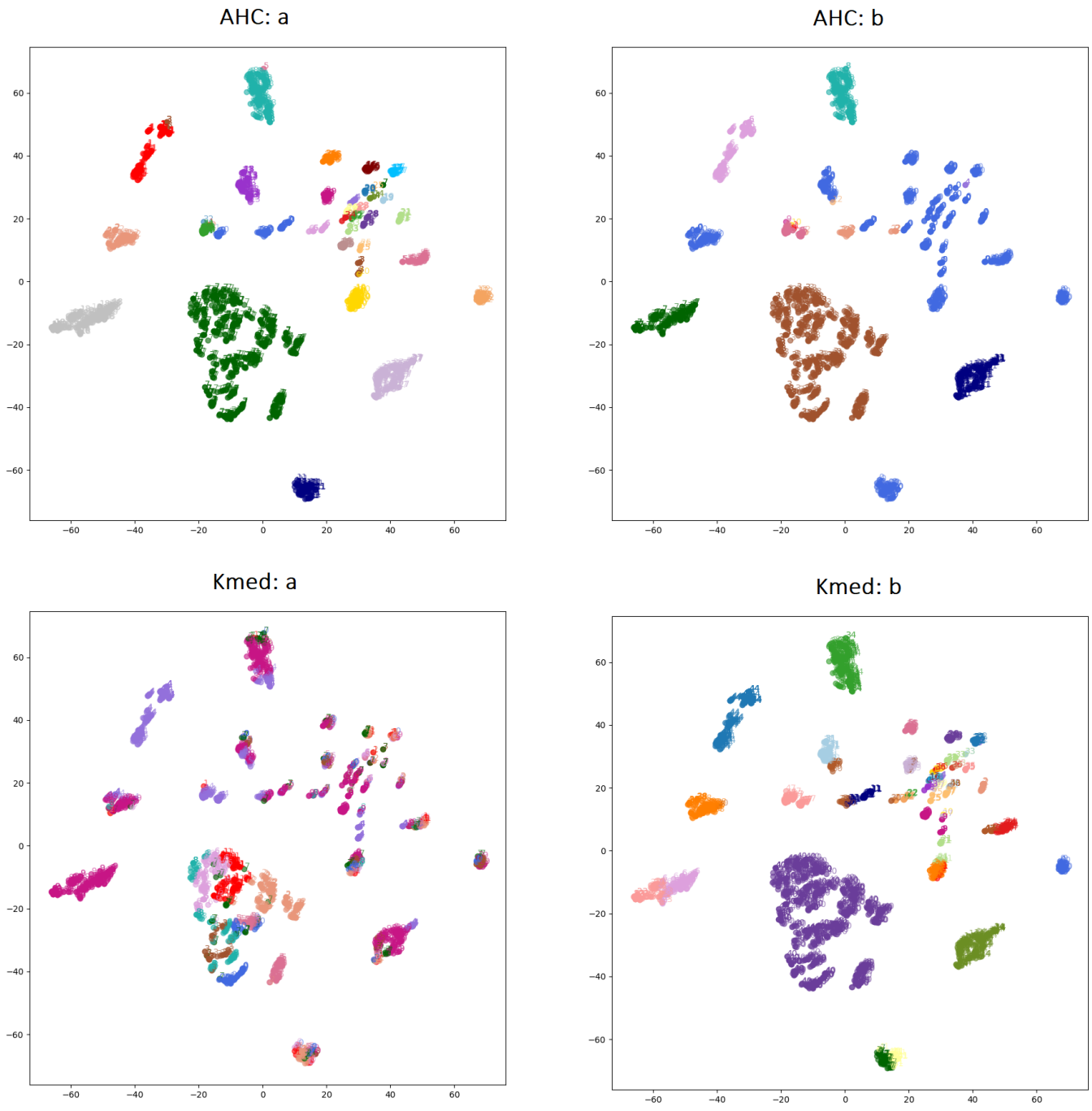

Figure 6: Clustering results of Agglomerative Hierarchical Clustering and K-medoids for configurations a and b.

5 Responsible Research

Improving MalPaCA is the purpose of this research and no ethical aspects were relevant so they are not further addressed.

A goal set within this research was to provide reproducibility of the experiments. This was facilitated by thoroughly explaining the experimental setup. The metrics that were used were illustrated using clear formulas and the dataset preparation was also clarified in detail. Furthermore, the code that was used to carry out this research is also publicly available on Github: `https://github.com/hjdeheer/malpaca`.

The total experiment was executed on a local machine in under 30 minutes, so one would not need to acquire access to a powerful server to be able to run the experiments.

6 Discussion

This section contains the discussion of the results shown in section 4.3.

6.1 Comparing AHC against HDBScan

It is interesting to notice that AHC a has a significantly lower error of 0.950 compared to HDBScan a with an error of 1.381 according to Table 1. This was mainly due to the higher cluster completeness error of HDBScan, since multiple connections with the same labels were placed into different clusters. For example, the large green cluster seen in the plot of figure 6 of AHC a was split up into multiple smaller clusters using HDBScan a as seen in figure 5. This large cluster is pure and only contains benign behaviour, so it was correctly clustered here by AHC a. After further heatmap inspection of HDBScan a, clusters 36 - 52 were the clusters that had very similar benign behaviours and so they should be grouped into a single cluster just like AHC a accomplished to do.

After using human visualization of the temporal heatmaps to calculate metric 2.3.6, HDBScan a resulted in an average error rate of 7.8% and AHC a resulted in an average error rate of 7.5%. Both configurations also have similar $cmpErr$ metric results. This similarity can be explained by the fact that the visualisation error metric is closely related to the cluster malicious purity metric since they both try to capture behaviour purity in a cluster.

Even though AHC a has no noise cluster to filter out undetermined connections, it still performs better in terms of cluster separation and cluster cohesion because of the lower $sErr$.

6.2 Comparing configuration a against b

Another intriguing observation is that configuration b performs better than a in 3 out of 4 times according to the $totErr$ in Table 1. This improvement is mainly because of noise reduction and a lower silhouette error. Considering the distance matrix as a list of connections with features seems to improve the clustering in terms of cluster separation and cluster cohesion.

6.3 Miscellaneous clustering results

Clustering results of OPTICS had a high percentage of noise. This is mainly due to the parameter $minPts$[1] that OPTICS requires to be set. In the currently used dataset, there exists a high variety of data density. Small groups of points that are lower in size than the $minPts$ will not be able to be grouped together and thus will be discarded as noise. K-medoids performed very poor on configuration a, but it did a better job in configuration b. An interesting observation on k-medoids configuration b clustering result was that it contained a much higher cluster size variance than the other clustering algorithms.

7 Conclusion and future work

Succinctly put, this report describes a comparative analysis of various clustering algorithms to determine which one has the best performance in terms of network behaviour discovery. Agglomerative Hierarchical Clustering (AHC) was decided to be the best due to the lowest total error of 0.950 in the comparative analysis where all algorithms were tested against metrics that capture cluster separation, cohesion, purity and completeness. The currently used algorithm, HDBScan, had difficulties clustering connections in varying data density. AHC is more capable than HDBScan in clustering data which has a high variance in density by not requiring a parameter that mentions the minimum amount of connections grouped together to form a cluster. A problem in the current version of MalPaCA is that HDBScan may discard connections as noise that represent rare attacking capabilities, since they lie in lower density regions. Agglomerative Hierarchical Clustering clusters data points without having a dedicated noise cluster unlike the currently used HDBScan algorithm whilst still achieving a higher cluster separation and cohesion.

Future work and improvements
In the current experiment, all connections that are not malicious have the label 'Benign' assigned to them. These benign connections can still contain multiple behaviours and this can lead to inaccuracies in the calculation of the metric scores. Namely, there exist some clusters that contain connections with mixed labels but have the same behaviour. A cluster that consists out of 50% malicious FileDownload connections and 50% benign connections results in a high error score of the cluster purity metric. These benign connections can still have file download behaviour and so they are appropriately clustered together with the malicious connections. This can further be improved by a labelling that is more specific for the behaviour of the connection. A possible way of approaching this is by performing a deep packet inspection using Nfstream [16] that assigns more specific behaviours to connections that can be utilized as labels.

MalPaCA should be further tested with different kinds of maliciously labelled datasets to prevent overfitting on the current dataset and to further test the different algorithms in their performance.

An interesting observation that arose from the research is the similarity between the visualization clustering error score and the malicious purity error score. Using the temporal heatmaps for determining clustering errors might be a useful metric to determine the purity of clusters in an unlabelled fashion, further investigation of this is needed.

[1] Number of points required to form a cluster

References

[1] A. Sharma, E. Gandotra, D. Bansal, and D. Gupta. Malware capability assessment using fuzzy logic. *Cybernetics and Systems*, 50(4):323–338, 2019. Cited By :4.

[2] Azqa Nadeem, Christian Hammerschmidt, Carlos H. Ganan, and Sicco Verwer. *Beyond Labeling: Using Clustering to Build Network Behavioral Profiles of Malware Families*, page 381â409. 2021.

[3] Donald J. Bemdt and James Clifford. Using dynamic time warping to find patterns in time series, 1994.

[4] William B. Cavnar and John M. Trenkle. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, Las Vegas, US, 1994.

[5] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. *Density-Based Clustering Based on Hierarchical Density Estimates*, pages 160–172. Transactions on Computational Science XXXVIII, 2013.

[6] Maria Jose Erquiaga Sebastian Garcia, Agustin Parmisano. Iot-23: A labeled dataset with malicious and benign iot network traffic (version 1.0.0). 2020.

[7] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, New York, NY, USA, 2015. Association for Computing Machinery.

[8] Jiawei Han, Micheline Kamber, and Jian Pei. *Cluster Analysis*, page 443â495. 2012.

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226â231. AAAI Press, 1996.

[10] Mihael Ankerst, Markus M. Breunig, Hans peter Kriegel, and Jorg Sander. Optics: Ordering points to identify the clustering structure. pages 49–60. ACM Press, 1999.

[11] Guillermo Suarez-Tangil, Juan E. Tapiador, Pedro Peris-Lopez, and Jorge Blasco. Dendroid: A text mining approach to analyzing and classifying code structures in android malware families. *Expert Systems with Applications*, 41(4):1104â1117, 2014.

[12] Marie Lisandra Zepeda-Mendoza and Osbaldo Resendis-Antonio. *Hierarchical Agglomerative Clustering*, page 886â887. 2013.

[13] Xin Jin and Jiawei Han. *K-Medoids Clustering*, page 697â700. 2017.

[14] L.J.P. van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[15] Hae-Sang Park and Chi-Hyuck Jun. A simple and fast algorithm for k-medoids clustering. *Expert Systems with Applications*, 36(2):3336â3341, 2009.

[16] Zied Aouini. aouinizied/nfstream: v5.2.0, July 2020.