ISEG
Lisbon School
of Economics
& Management
Universidade de Lisboa

MASTER IN

Data Analytics for Business

MASTER'S FINAL WORK

Dissertation

Supervised Clustering With SHAP Values

Rodrigo Queirós Conceição

March - 2023

Lisbon School of Economics & Management
Universidade de Lisboa

MASTER IN
DATA ANALYTICS FOR BUSINESS

MASTER'S FINAL WORK
DISSERTATION

SUPERVISED CLUSTERING WITH SHAP VALUES

RODRIGO QUEIRÓS CONCEIÇÃO

ADVISOR:

JOÃO AFONSO BASTOS

MARCH - 2023

Abstract

In the last years, data has grown at a fast rate. Not only growing in size, data is also becoming far more complex then what it used to be. As companies are shifting to data-driven environments, this complexity dificults the analysis and extraction of value from the data. As a result traditional methods are becoming obsolete as their performance is decreasing and machine learning and deep learning models are becoming more complex so the desirable accuracy scores can be achieved.

This work proposes an approach that is capable of recognizing complex relationships and identifies groups that are not visible at first glance while providing a full interpretability of the methods used. It combines a black-box model with SHAP values to generate clusters from the explanations that were previously unknown. The clusters obtained are a combination of multiple local explanations that SHAP values offer and are easily interpretable since the feature values correspond to the feature importance assigned by the model.

To implement this approach, a dataset containing the properties of benign and malware samples, designed for malware detection tasks, was used. It is shown that by combining SHAP values with XGBoost it is possible to generate new clusters, that were previously hidden and unobtainable with traditional approaches. This clusters are highly interpretable as they derive from SHAP values and have the support of a supervised environment.

KEYWORDS: SHAP values; Black-Box models; Interpretability; Supervised Clustering, XGBoost

Table of contents

List of figures

List of tables

Acknowledgements

I would like to thank my parents for always believing in me and for their indispensable support through my academic path.

I would also like to thank my professor and advisor João Bastos for his steady willingness to help and guide me through this project.

Final thanks to my friends from the masters' for making this challenge more enjoyable and for always sharing their opinions and help.

1 Introduction

The cybersecurity industry is growing at a fast pace and as companies grow and open themselves to the IoT and cloud environments, they also create a window for cyberattacks, which are getting more frequent and sophisticated. One particular concern is the rising of malware attacks that spread and causes harm and losses to new companies every day. As a response to these attacks, cybersecurity strategies pass by techniques and tools such as antivirus, firewalls, detection systems and one method that has been rising, machine learning. Machine learning allows the companies to detect and eliminate threats, providing a more accurate and real-time analysis, resulting in a powerful and essential tool in the cybersecurity industry. In this fast-moving environment it is essential to keep the models up to date to efficiently fight the online threats as they are always changing and exploring new gaps. This can prove to be quite challenging and some models end up failing or becoming too complex to use. For instance, clustering is a very well-known unsupervised machine learning technique that has been used for a long time, through different algorithms, it allocates data points that are somehow similar to each other into distinct groups. However, grouping malware data into meaningful clusters can sometimes be quite challenging, from selecting the most suitable algorithm to the ongoing increase of the data's complexity itself, the clusters found may not produce the best results. On the other hand, when dealing with detection problems with supervised learning approaches, models can become very complex and, therefore, hard to interpret. They also do not account for the differences among the samples of the same classes, neglecting the existence of subgroups among those classes. Finding these groups can reveal to be even more tricky as the classification models do not consider them directly and labelling the samples correctly, prior to the application of a model, is very costly to achieve.

This thesis explores a different approach, Supervised Clustering with SHAP values (Aidan Cooper et al., 2021), it consists of guiding the cluster algorithms (unsupervised) with the help of classification models (supervised) to overcome the problems previously identified. To be able to do this it uses an explainable artificial intelligence (XAI) tool, Shapley Additive exPlanations (SHAP), as a pre-processing step. SHAP values (Lundberg and Lee, 2017) are an adaptation of the Shapley values (Shapley, 1953) to the machine learning framework, instead of having games and players, it is focused on models and features. It explains the output of a machine learning model by calculating the importance of the features locally and globally, contributing to the interpretability and transparency of black-box models. SHAP values can be thought as a second way for when the characteristics of the data itself are not enough, it can complement the clustering and classification models analysis by giving a different insight of the data, especially on the local level. The conjunction of this approaches generates new clusters that can be easily interpretable and allow transparent analysis. Instead of finding clusters based on the raw characteristics of the data itself, this method consists of finding clusters based on how a classification model attributes data points to a class, more precisely, it creates clusters around data points where the features share the same importance in a certain classification model.

The objective of this study is to compare the clusters produced by this approach against the traditional methods, using a malware dataset, and show how this methodology not only performs better at producing clusters but also has a strong interpretability to support the analysis and use of classification models

The development of the work is divided into the following sections; the next section consists of a literature review regarding the problems identified and techniques addressed. The third section presents the methodology of the approach taken, explaining all the processes used and how the results are going to be accessed. Section 4 explains the data that was used for the work, followed by section 5, where the results obtained will be covered and analysed. The last session will consist of the conclusions and discussion of the project.

2 Literature Review

2.1 Machine learning in Cybersecurity

The idea of implementing machine learning to the cybersecurity industry goes back to the 1980s where intrusion detection systems (IDS) (D. E. Denning, 1987) based on anomaly detection are implemented. DARPA researchers also created benchmarks datasets to train machine learning methods (Lippmann et al., 1999). However, these applications were still very primitive, the biggest problems faced had to do with the low accuracy on the detection rates, resulting in a large rate of false alarms and failing to detect new attacks. The training data was hard to obtain and its quality was not the best. With the lack of resources such as computational power, available data and efficient techniques, this field only started to see some progress again with the introduction of big data.

Big Data leveraged the use of Artificial Intelligence and machine learning for cybersecurity. With the companies becoming data-driven and the processes automatized, the risk of cyber attacks increased with millions of malware attacks every day, but the implementation of more sophisticated and accurate models was also made possible. Fraley and Cannady(2017) explain how Big Data is affecting the business with threats every hour and how the analysts cannot deal with all the problems in feasible time. They proceed to explain how machine learning can leverage the cybersecurity industry from the possible tasks and datasets to the model development and evaluation. Since then, researchers took focus on this topic, with special attention in the last years, and have been improving and discovering effective techniques. From the implementation of simple supervised methods like Support Vector Machine (SVM) and decision trees to complex neural networks, Yang et al. (2018) summarize the methods used through the years as well as the public datasets that were made available to train the models.

Regarding the current state and challenges of machine learning, three main problems are identified by Gibert et al. (2020). The first problem, concept drift, takes in consideration the fast pace of this industry and how malware evolves over time. The models need to have in consideration that the historical data is different from the future data and the relations keep changing. The second problem is about the adversarial learning and how reverse machine learning can be used to fool the detection models by alternating the feature space in the malwares favour, making it seem like a benign sample. The last problem relies on the interpretability of the models. With the use of black box models being more common and essential in the detection of malware, analysts can have a hard time understanding the model's decision. This compromises the black-box models utility because despite proving to be efficient in detecting, analysts cannot properly explain the model decision. This may lead to analysts preferring simpler models, they might have lower accuracy rates but it allows the analysts to have full control on the model decisions.

2.2 Clustering in cybersecurity

Cluster algorithms are usually implemented in the process of intrusion detection systems, and its application are mainly focused on two areas. The first area has a greater search and is related with the implementation of clusters as a pre-processing step, the data is divided according to the cluster algorithm and then the models are applied separately to each cluster. Khorolska et al. (2022) explain how the evolution and increase complexity of AI technology has lead cluster algorithms to help in the decision making. Furthermore, Rathore et al. (2021) show how combining the cluster algorithms with other machine learning and deep learning models improves the classification tasks. The other use case has to do with the identification of clusters that can represent the different malwares and create distinct groups. Generally, the works on this topic show that finding meaningful clusters is often a challenge as the malware data is too complex and the results end up not being good enough to be used compared to other techniques. Renato and Carlos (2021) show the problems of finding meaningful clusters in malware data and explains that the patterns of this type of data makes it hard for distance-based cluster algorithms to divide the data properly. Other problem is the fact that the data normalisation breaks the relevancy of the features, which brings a negative effect as features have different importance. To solve this problem, they implement a method that calculates the degree of relevance of the features, which in some way is similar to the SHAP values, and show that the clusters perform better. In Basole and Stamp, (2021) while trying to find a relation between malware categories and their families they notice that while some clusters capture distinct malware families, others could not capture information due to the complex relationship. Nevertheless, they found out that some families within the same category are more similar to each other while others are completely different.

2.3 SHAP Values in Cybersecurity

Interpretability and XAI are two very sensitive topics that have gained importance over the last years across a wide range of industries. The use of black-box models is becoming necessary for machine learning and deep learning models to be efficient, however, it comes with a cost of human interpretation. Cybersecurity industry is no exception and, as Charmet et al. (2022) mention, XAI brings multiple advantages that are indispensable to analysts in the cybersecurity industry, it supports the complex model's decisions and can be used for different applications. One new XAI tool that has raised some attention is the SHAP values. SHAP values are being used to improve intrusion detection systems and other tasks that involve machine learning and deep learning models by combining local and global explanations to interpret the different complex models used.

Researchers are using new frameworks for IDS that include SHAP values to explain the predictions of different classifiers and how they differ (Wang et al., 2020) and (Alenezi and Ludwig, 2021). With SHAP values, analysts can have a better understanding of what features are having more impact on the decision, both at the global model view and sample view, guiding them on the alerts and the model overall behaviour for future improvements and modifications.

This work focuses on a framework that tries to take the most of the feature's importance values of SHAP's local importance, which is not taken in consideration on other works. By exploring the potential of finding unknown groups of samples with different behaviours with cluster analysis, it provides a global view of that cluster that is unique in the model. This can help analysts identifying subgroups of samples across the same malware categories whose behaviour is distinct.

3 Methodology

The supervised clustering with SHAP values approach requires pre-processing procedures so that the clustering algorithms can be applied efficiently. In this section, it is carefully explained the multiple steps of the approach, the pre-processing techniques and how the results are measured, all detailed in the order which they were implemented as it can be seen in the Figure 3.1. This approach is similar to the traditional clustering approaches with the exception that it includes a new pre-processing step using the SHAP values to generate a new dataset. It also has a different cluster analysis and evaluation as the true label values are known.

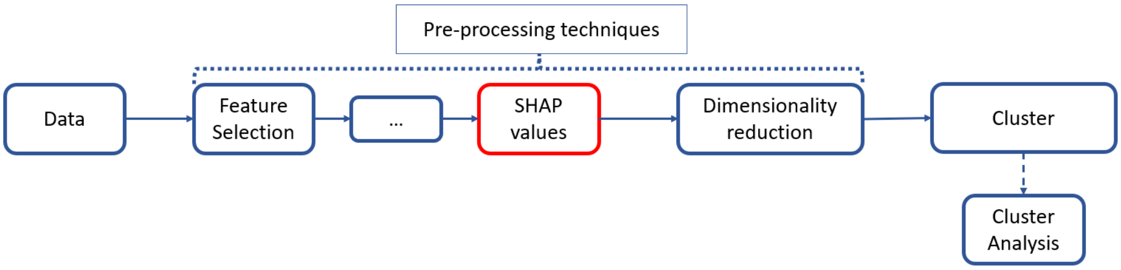


Figure 3.1: Scheme of the methodology's processes

3.1 *Pre-Processing techniques*

There are multiple types of pre-processing techniques that can be considered before applying a machine learning algorithm, however, their implementation often depends on the characteristics of the data and the current problem. In this section, the main pre-processing steps that are indispensable to this work are carefully explained, from their advantages to the reasons that they were chosen.

3.1.1 *Feature Selection*

The quantity of features varies according to the domains and models analysed, often datasets come with a reasonable large number of features, including features that are considered noise. As these features end up being irrelevant to the machine learning models, feature selection focus on only retaining the relevant features to be used by the models.

Among the feature selection techniques, the Pearson correlation coefficient was chosen. It measures the strength of the linear relationship between the features assigning a correlation value to each pair of features. A threshold value is selected, eliminating all the features with an absolute correlation above the threshold settled.

$$R_{x_i y_k} = \frac{\sum_{z=1}^{n}(x_{iz} - \overline{x_\imath}).(y_{kz} - \overline{y_k})}{\sqrt{\sum_{z=1}^{n}(x_{iz} - \overline{x_\imath})^2} \cdot \sqrt{\sum_{z=1}^{n}(y_{kz} - \overline{y_k})^2}}, \tag{3.1}$$

where α is the number of features of the correspondent dataset, n is the number of samples of the dataset, $x_i\,/y_k$ are the feature i and the feature k respectively, x_{iz} correspond to the sample z of the feature x_i and the same reasoning is applied for y_{kz}. $\bar{x}_i/\overline{y_k}$ corresponds to the sample mean for the respective feature. $R_{x_i y_k}$ is the correlation between each feature pair $x_i\,/y_k$ and can take a value between -1 and 1, where the larger the absolute value is, the stronger is the relation between the two features.

The Pearson correlation coefficient was selected since it is an efficient technique to eliminate features that do not give any new contribution to the models and it aligns with the SHAP values methodology, that will be explained further bellow in this section, as they cannot deal perfectly with correlated features. Python has a built-in function that makes the Pearson correlation formula easy to apply.

3.1.2 SHAP Values

For each feature, SHAP values measure the contribution to the model output by doing a weighted summation average of all possible features subsets and calculating the marginal contribution of that feature to the model. The SHAP values is a local method, which means that for every row/instance in the dataset, it calculates the SHAP values for the feature i , given a model f and a vector x of features to be explained;

$$\emptyset_i(f, x) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\,(N - |S| - 1)!}{N!}(f_x(S\ \cup \{i\}) - f_x(S)), \tag{3.2}$$

where S is a subset of the features used in the model and N the total set of the features in x, having $S \subseteq N\setminus\{i\}$ as all the possible subsets excluding the feature i, $f_x(S\ \cup \{i\})$ and $f_x(S)$ corresponds to the predictions of the model f_x, given a set of features S, with and without the feature i, respectively, marginalized over the feature that is not included in that set S.

SHAP values can also serve as a global method, calculating the feature importance for each feature in the model by doing an absolute average of the Shapley values:

$$I_j = \frac{1}{n}\sum\nolimits_{z=1}^{n} \left|\emptyset_j^{(z)}\right|, \tag{3.3}$$

where j represents each feature and z each observation.

SHAP values are model agnostic, supporting any machine learning model, however they have some problems. The first problem is the most relevant, the fact that it requires a big computational complexity, $2^n - 1$ steps (exponential growth), where n is the number of features used to calculate the Shapley values, this will consume a lot of time as the rows and features of the datasets increase. To address this problem, explainers were implemented to be more time efficient, in particular, the TreeExplainer (Ludenberg et al., 2020).

The TreeExplainer method is a tree-based model exclusively that allows the calculation of Shapley values in polynomial time, it does not require a background dataset to sample from and calculates the exact Shapley values. It does so by only computing the values for the relevant nodes of the given instance of a tree. The second problem is related to the correlation between features, as the features are correlated and pass the same information to the model's output, when calculating SHAP values, the difficulty of assigning the correct feature importance to each feature is raised which might result on different solutions since the same feature importance is split between the correlated features. This can lead to possible miss interpretation of the SHAP values as the true feature importance is not explicit and different SHAP values can be obtained. Implementing a pre-processing step that deals with correlated features attenuates the problem.

SHAP values act as a pre-processing step in the way that a new dataset is created where the original data points are replaced with the corresponding SHAP values. The new dataset maintains the same shape/structure of the original one (number of samples and features) and the data points do not represent the original characteristics of the data, instead they represent the contributions of the features of each sample given a model. Also, it is important to take in consideration that the SHAP values act based on the model's predicted value which can be different from the true label if the sample was misclassified.

3.1.3 Classification Model

SHAP values require a base model to infer on. Given SHAP's characteristics and the problem that this work addresses, any classification model can be used, however, due to the use of TreeExplainer, it shrinks the classification models to tree-based only.

Taking in consideration the classification tree models available, the eXtreme Gradient Boosting algorithm (XGBoost) was chosen as the input model. Since its introduction (Chen and Guestrin, 2016), XGBoost has gained popularity against other classification models as it provides greater efficiency and accuracy, proving to be one of the strongest supervised machine learning models. XGBoost is a tree-based ensemble algorithm that uses a combination of gradient descending with boosting, gradient boosting, by combining multiple small trees, also called weak learners.

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F, \tag{3.4}$$

the prediction of each tree is summed up and complemented by each other, where K is the number of trees, f_k corresponds to a tree with its own structure and weights and F corresponds to the set of possible classification trees. Using a gradient descent, a loss function is calculated in each tree and the model trains on the residuals, giving more importance to misclassified observations and, consequently, the next weak learners act on where the existing trees are failing. The objective is to minimize the loss function to a point that it cannot decrease anymore, meaning that no more trees are created and the algorithm stops. The objective function can be written as,

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t), \tag{3.5}$$

where the first term is a loss function that depends on the trees created up to the iteration t, y_i is the real label value from the dataset and $\hat{y}_i^{(t-1)}$ is the value predicted from the previous trees plus the prediction of the current tree f_t. The second term is a regularization parameter that prevents the current tree from overfitting by penalizing the complexity. Combining all these methods makes XGBoost a powerful tool, however, like any boosting algorithm it can overfit the data quickly, resulting in adequate models.

In this methodology, the goal is to analyse a dataset by fitting a model and deducing the SHAP values from the model. In the process of fitting the appropriate weights and parameters for the data, the model needs to be trained. To do so, the data is randomly shuffled in two parts, 75% of the data goes for training and the other 25% to test the model. To ensure that the models do not overfit, XGBoost has a wide range of hyperparameters that regularize the model and make it more conservative while maintaining its potential. Since the learning process for these parameters is not automatic, the Grid search methodology is used to find the best values for the parameters. It combines a set of possible values and creates multiple models with different sets. Through cross validation, the values of the hyperparameters for the model that performed best are given as the optimal solution. For this work, the following parameters were tunned*:* *n_estimators*, *min_child_weight*, *max_depth, learning_rate, gamma, early_stopping_rounds* and *colsample_bytree.* Table 3.1 gives a brief description of how the hyperparameters affect the model.

Hyperparameter	Definition
n_estimators	Number of trees(estimators) built by the model. Has the value grows, the complexity of the model increases.
min_child_weight	Minimum sum of samples weight needed in a child to split. Higher values reduce the overfitting by limiting the size of the trees.
max_depth	Determines the maximum depth that the trees can go to. The more complex the trees are, more is the risk of overfit.
learning_rate	Sets the weight at which the model updates in each interaction. Large value leads to a faster convergence but less accurate model.
gamma	Determines the minimum loss reduction to split the tree. Increasing the value of gamma helps controlling the overfit.
early_stopping_rounds	Sets a number of rounds that the model is going to stop if no improvement is detected, preventing excessive training time.
Colsample_bytree	Determines the fraction of columns sampled for each tree. Increasing the values improves the training process and reducing the overfit

Table 3.1: XGBoost hyperparameters

3.1.4 Dimensionality Reduction

With big amounts of features to deal with, some machine learning algorithms struggle in processing all the information properly, making it harder to find a solution in a feasible amount of time (curse of dimensionality). On the other hand, it is hard for humans to interpret graphs above 3 dimensions. To produce meaningful visualisations of data and for machine learning algorithms to be efficient, it is necessary to reduce the dimension of the features to a lower dimensional space that can be projected.

This technique plays an important role when it comes to clustering algorithms. First, it allows the algorithm to overcome the curse of dimensionality. Second, it allows the visualization of the data in two- or three-dimension graphs that humans can extract information off.

Essentially, the Dimensionality reduction algorithms fall in two categories; matrix factorization or neighbour graphs. The first category tends to preserve more the global structure of the data while the second category prioritizes the preservation of the local structure of the data. Therefore, these algorithms play a trade-off of trying to keep the maximum structure of the data while reducing its dimension.

Uniform Manifold Approximation and projection (UMAP) was the algorithm chosen for this approach. This algorithm belongs to the neighbour graphs category and can separate clusters on higher dimensions, reducing it to low dimensions while maintaining the local structure of the data and preserving part of the global structure. Leland and John (2018), the authors of UMAP, carefully explain all the mathematical assumptions and foundations behind it but the main idea is that it constructs a high dimensional graph of the data with wedges and weights that represent the likelihood of points being connected and then tries to replicate the same structure but in a lower dimension level.

This approach holds some advantages against other widely used dimension reduction algorithms, like Principal Component Analysis (PCA) and t-distributed stochastic neighbour embedding (t-SNE). PCA sometimes is not able to reduce the data to a low dimension at the cost of losing most of the data structure and it can only preserve the global structure of the data. UMAP also tends to outperform t-distributed stochastic neighbour embedding (t-SNE) when it comes to scalability, achieving results much faster, and on preserving the global structure of the data, which is very important when it comes to inter-cluster analysis.

Like XGBoost, UMAP has two hyperparameters that have a significant impact on the result*, n_neighbors* and *min_dist*. These parameters represent the trade-off that exists between the global and local view, so the way to optimize these parameters depends on how we want to visualize the data and based on trial error. Table 3.2 describes the hyperparameters.

Hyperparameter	Definition
n_neighbors	Sets the number of neighbours used in the construction of the neighbourhood graph, high values result on preserving the global view and higher computational costs.
min_dist	Sets the minimum distance between points in the low dimension defined, increasing the value leads to more separation at the cost of loss of the local view.

Table 3.2: UMAP hyperparameters

3.2 Cluster techniques

After applying all the pre-processing procedures, a cluster algorithm needs to be selected. As it was mentioned before, cluster techniques can be used to serve multiple purposes in machine learning. Along with this, there are several different clustering algorithms that one can use, each producing their own clusters and results. Therefore, in this section, the cluster algorithms that were tested are presented. The objective is to choose the algorithm that can better fit the problem identified.

3.2.1 K-Means

The k-means is a simple and widely used iterative algorithm. It groups the data, based on its similarity, into a predefined number of clusters. These clusters initialize randomly but are updated on each iteration, they start being built around their centroids, minimizing the distance of the data points to that center until the clusters stabilize. The distance measure used will depend on the type of the data used. The next table represents the pseudocode of this iterative process.

```
Input:
  Data
  K number of clusters
Process:
  Randomly initialize k centroids
  Assign each data point to the closest centroid
  Update the centroid to the mean of data points to the cluster
Output:
  Final centroids and the assignment of each data point to the cluster
```

Table 3.3: K-Means pseudocode

Due to its simplicity, this algorithm is very efficient and most of the times can produce meaningful clusters. On the other hand, it requires some prior knowledge of the data or the resort of techniques (Ex: Elbow Method) to find the optimal amount of clusters. K-means is also sensitive to noisy data, which can affect the clusters creation.

3.2.2 DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN is another cluster algorithm that groups the data based on its density. High-density regions of data will be clustered together while low-density regions will be marked as noise. It does not need to have a predefined number of clusters to act like K-means. DBSCAN will define the clusters based on two parameters: *Eps* and *MinPts*. *Eps* consists of the maximum distance between two data points so the data point can be considered on the same cluster and *MinPts* is a minimum number of data points that a cluster needs to have. DBSCAN is also a relatively efficient algorithm that overcomes two of the k-means problems. It does not require the user to specify a number of clusters and is able to identify noise with ease. It can also capture arbitrary shapes easily but it is very sensitive to the values of the parameters Eps and MinPts, becoming a challenge to select the correct values. The following table displays the process:

```
Input:
  Data
  Eps
  MintPts
Process:
  If data point is not visited, mark it as visited
    If the point is a core point, initialize a new cluster
      add all the points within the Eps distance to the cluster
        repeat the process for all points in the cluster until no more points can be added
    If the point is a border point, add it to the nearest cluster
Output:
  The assignment of each data point to the cluster
```

Table 3.4: DBSCAN pseudocode

3.2.3 Gaussian mixture model/ Expectation-Maximization

Gaussian Mixture Models (GMM) assume that the data is generated by different gaussian distributions. Each cluster will have its own gaussian distribution with a mean and covariance parameter. To find the optimal parameters, it will resort to an iterative process with the Expectation Maximization (EM) algorithm until the convergence is met. This algorithm works by applying two steps iteratively, first, it estimates the probability of a data point belonging to a cluster through the empirical probability density function and, on the second step, it updates the parameter of the gaussian distribution for the clusters according to the probabilities observed on the first step. GMM can handle complex data with missing values at the cost of computer complexity but, like K-means, it requires the user to specify a number of clusters a priori.

3.3 Choosing the cluster algorithm

When it comes to identifying which algorithm to choose, it is clear that there is not a straightforward answer. As it was seen in this section, each algorithm has its own strengths and weakness and its fitness will depend on the problem faced. Overall, there is not a best cluster algorithm to choose, it will be necessary to consider the characteristics of the data, the problem that is faced and to have the awareness of the capability of the different algorithms. Only by trying the different algorithms and having in consideration the different assumptions it will be able to select a proper cluster algorithm to work with.

3.4 Cluster Analysis

This cluster analysis differs from the traditional unsupervised methods since the true labels are known. Instead of evaluating the clusters with methods that measure distances and similarity (Within-Cluster sum of squares or Silhouette Score), the clusters are assessed based on two steps. First, the classes off the samples that belongs to the clusters are taken in consideration, identifying the different classes contained in the cluster. Having different classes in the same cluster does not necessarily mean that the cluster is not viable. In SHAP values perspective, it informs that, despite the samples belonging to different classes, they present the same behaviour for the input model, being, therefore, grouped together. Secondly, it is used one of the SHAP values properties to get the global view of the variables for each cluster. By calculating the mean of the SHAP values variables for each cluster, it can be seen how the different variables are affecting different clusters. Ideally the clusters identified will have different SHAP values leading to samples with a different order of feature importance values.

This clusters will give extra information that the global view and local view of SHAP values cannot give and will boost the decision making of analysts. From this point on, the analysts can interpret the results with different perspectives, according to their objectives and focus. The clusters can be filtered according to the different classes, being able to focus on only one class or multiple classes at the same time. They can also focus on the samples that were misclassified, viewing the clusters that contain them and to which samples the model is associating them.

4 Data

4.1 CIC-MalMem-2022

To support this work, it was used a publicly available dataset, provided by the Canadian Institute for Cybersecurity (CIC), the CIC-MalMem-2022 dataset. This dataset was developed by Carrier et al. (2022) and serves as a benchmark model that tries to represent the real-world situation incorporating malware and benign samples captured from a computer environment and then transformed into variables to construct the dataset.

This dataset was created with the intuit of testing detection models through memory analysis consisting of 58.596 samples evenly balanced between benign and malware instances as it can be seen in Figure 4.2.

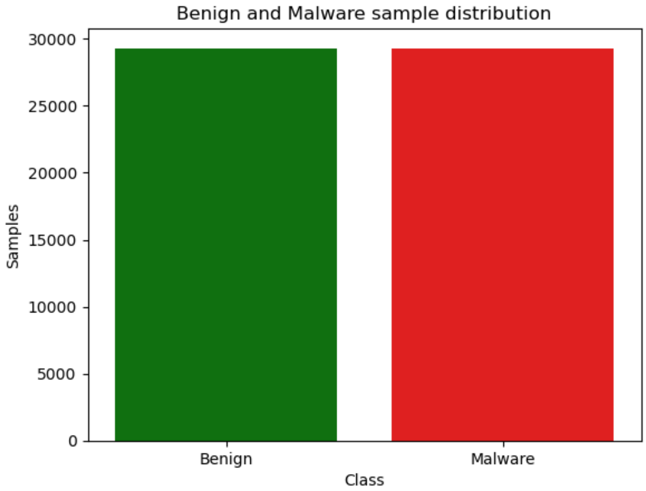


Figure 4.2: Samples distribution among the Benign and malware classes

Inside the Malware class, the dataset contains three different malware categories, Ransomware, Spyware and Trojan and each category contains five different families, maintaining the balance between each category and family, like Figure 4.3 shows.

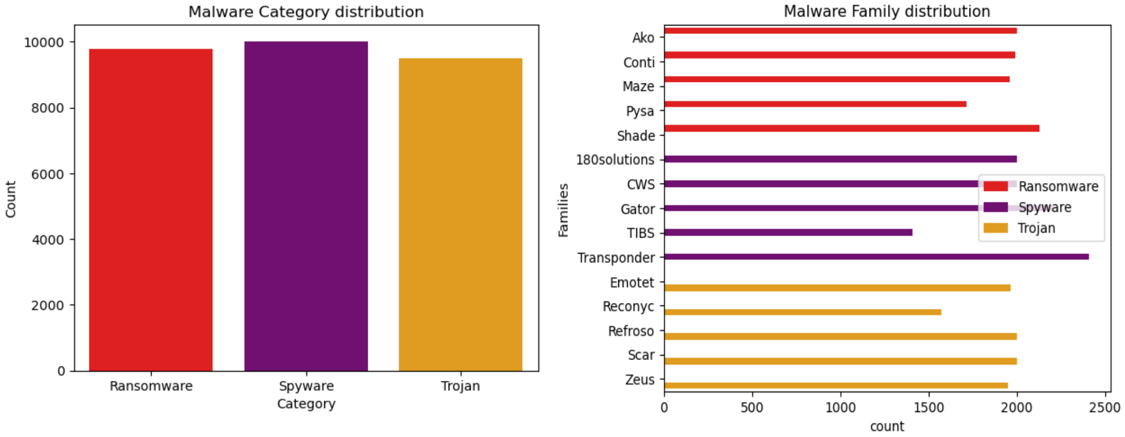


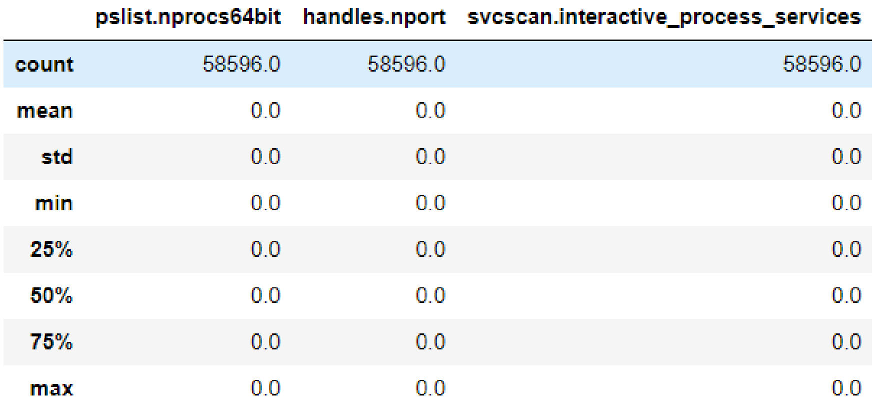
Figure 4.3: Malware samples distribution among the different categories and families

4.1.2 Variables

The dataset contains fifty-eight features extracted from memory dump files using an open source tool. Three of them identify the class, category and family of the correspondent sample, while the other features are split between features that focus on general characteristics of memory and others that target specifically hidden malware. The features can be divided into five categories, *Malfind* which focus on malware associated with trojan malware behaviour, *Ldrmodule* looks for injected code, mostly related with spyware, *Process View* category looks into the process lists of the system to find malicious processes, *API hook* looks to the API (Application Programming Interface) calls and searches for modifications and the last category, *Handle*, monitors the Handles of the computer system.

4.1.3 Transformations

This section covers all the transformations done to the data to prepare it for the implementation of XGBoost, SHAP values and Cluster algorithms. After looking at the descriptive statistics of the data, three variables were eliminated as they only contained zeros and would not pass information to the models as it can be seen in Figure 4.4.



	pslist.nprocs64bit	handles.nport	svcscan.interactive_process_services
count	58596.0	58596.0	58596.0
mean	0.0	0.0	0.0
std	0.0	0.0	0.0
min	0.0	0.0	0.0
25%	0.0	0.0	0.0
50%	0.0	0.0	0.0
75%	0.0	0.0	0.0
max	0.0	0.0	0.0

Figure 4.4: Descriptive statistics of the features eliminates

The next step was to look at the correlation between the variables, in this dataset, as we can see in Figure 4.5, correspondent with the dark green and dark red colours, there are quite a few features that show high correlation in between them.

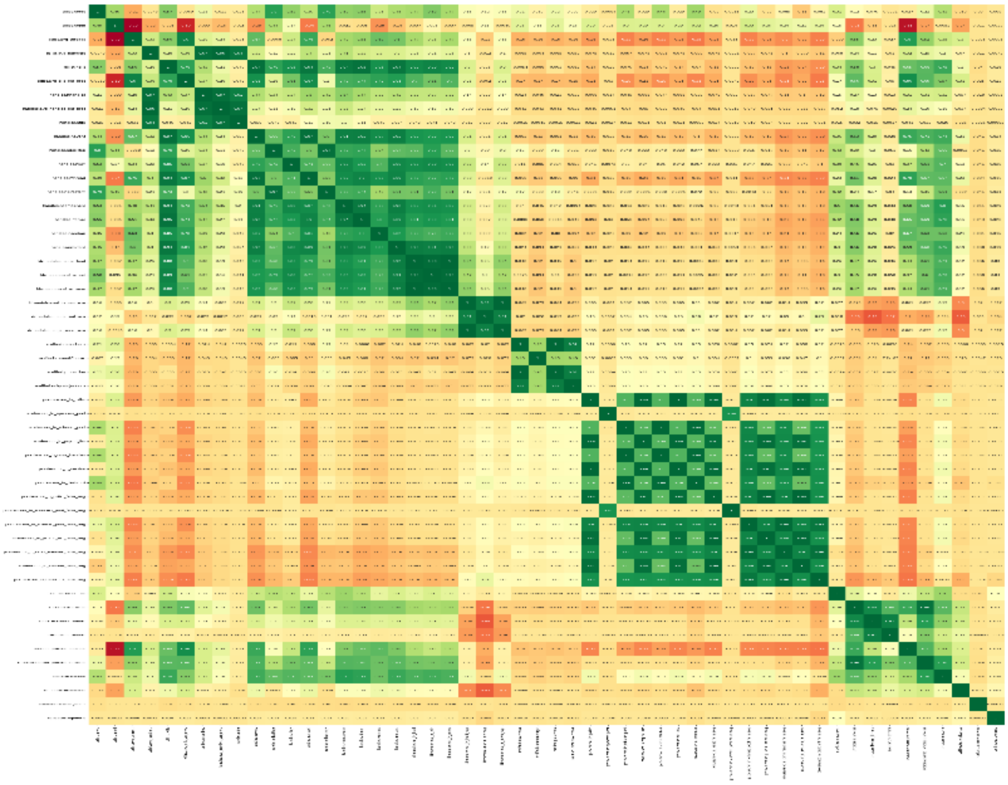


Figure 4.5: Correlation matrix between the features

Since the SHAP values are very sensitive to the correlation between variables, a threshold of 0.7 was selected. This will ensure that the SHAP values will keep their consistency and produce trustworthy values.

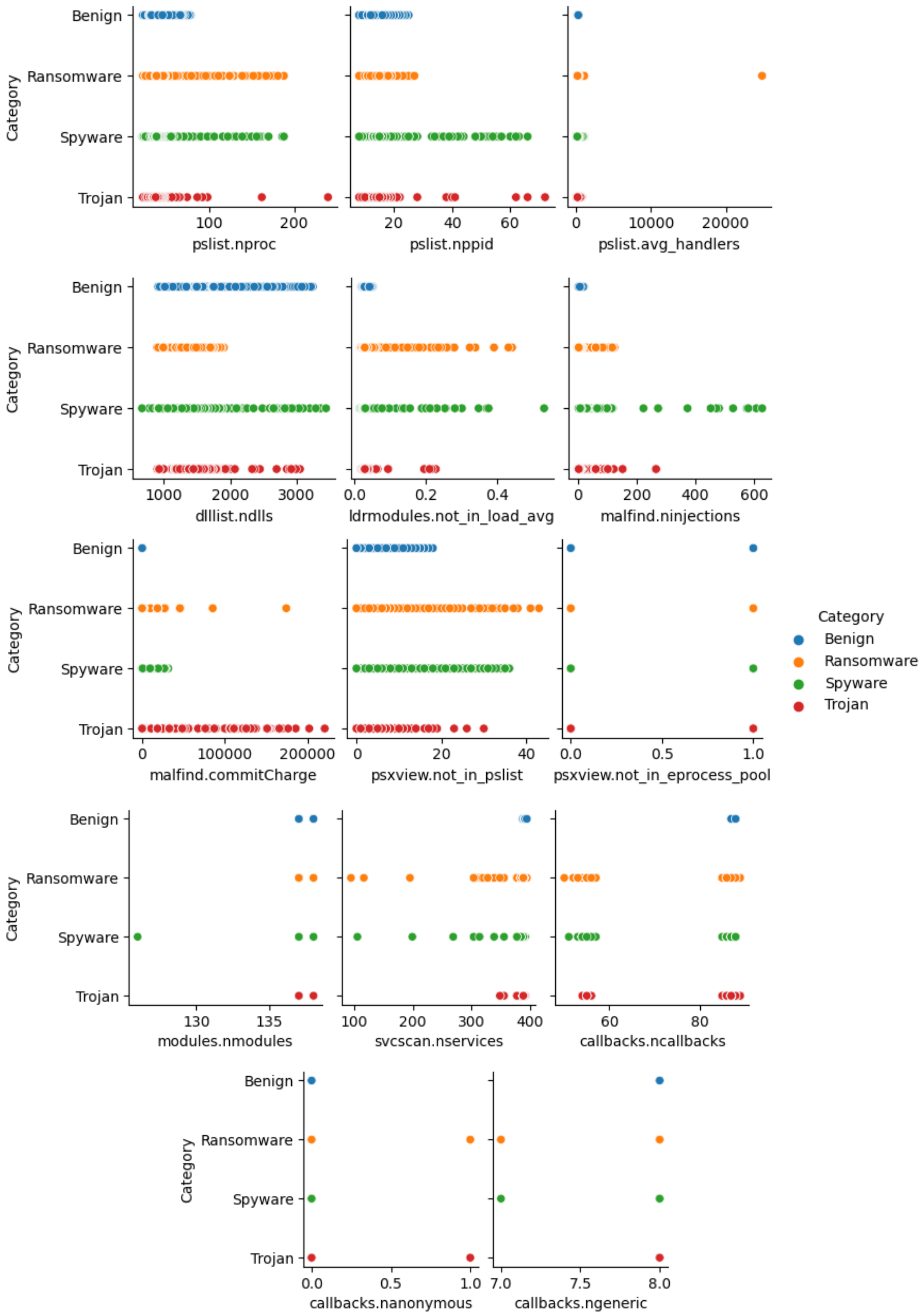


Figure 4.6: Samples values for the final feature set

As a result, out of the fifty-two variables, thirty-eight were removed. The final dataset consists of fourteen independent features and three target labels, Figure 4.6 shows the behaviour of the different variables for each category. For the purpose of this work, the target label considered by the models will be the category label while the other labels will serve to complement the analysis.

5 Results

5.1 SHAP Values- Classification model

The first step to create the SHAP values dataframe is to define a model, since XGBoost was the chosen model, we need to ensure that it performs well and does not overfit the data. For this purpose, the hyperparameters were optimized according to the grid search technique by obtaining the lowest log-loss score out-of-sample. Table 5.5 shows the values proposed and selected for the hyperparameters.

Hyperparameter	**Value Proposed**	**Value Selected**
n_estimators	[100,200,500]	500
min_child_weight	[1,3,5,7]	1
max_depth	[10,15,20,30,50]	10
learning_rate	[0.05, 0.1, 0.15, 0.2, 0.25, 0.3]	0.15
gamma	[0.0, 0.1, 0.2, 0.3, 0.4]	0.4
early_stopping_rounds	[25,50,100]	100
Colsample_bytree	[0.3, 0.4, 0.5, 0.6, 0.7]	0.7

Table 5.5: XGBoost hyperparameters optimized with grid search and the selected values

The model obtained an accuracy of 94,11% with a validation score of 0.17801, as we can see in the confusion matrix below it distinguished the benign from the malware samples easily, only misclassifying two benign samples. However, when it comes to identify the different malware categories, the model struggles to find the difference between some samples, being this the main cause for the decreasing of the model accuracy. The following confusion matrix displays the output of the model.

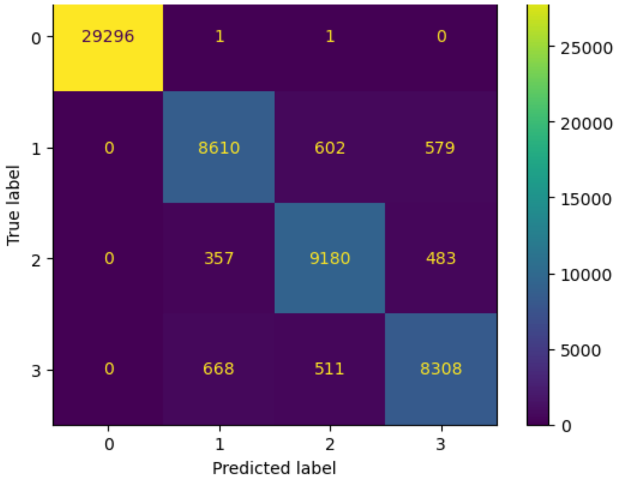


Figure 5.7: Confusion Matrix for the XGBoost model implemented

5.1.1 Computing SHAP values

Once the model is prepared, the SHAP values are deducted. Given the use of the of the TreeExplainer and the reduction of the feature space, the complexity and computational power of this task is reduced to a computing time of 1 minute and 49 seconds. To have a general view of the values obtained, Figure 5.8 shows the mean absolute SHAP values for each feature.

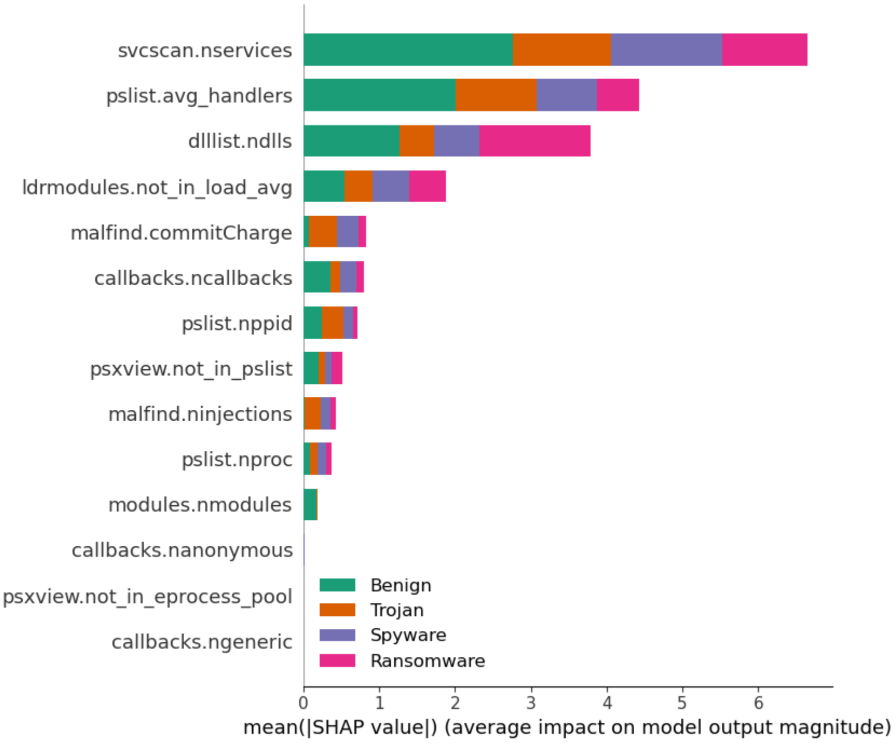


Figure 5.8: SHAP global feature importance of the XGBoost model for the diferent categories

At a global view level, the features that have bigger impact to the model are the *svscan.nservices*, *pslist.avg_handler* and *dillist_ndills*, on the other hand, *callbacks.nanonymous*, *psxview._not_in_eprocess_pool* and *callbacks.negeneric* are the features with less relevant impact. To complement this analysis, we also need to look at the local view, beeswarm plots allow us to see how each sample, represented by a dot, is allocated across the features.

Looking at Figure 5.9 and 5.10, it is possible to identify that each sample has its unique feature importance and behaviour. Furthermore, we can see that for each category, the order of the most important features is different. It is also important to note that besides the *callbacks.ngeneric* feature, which brings no importance to the model as the SHAP values are 0 for all the categories, the features with the lowest mean importance, are relevant for particular samples meaning that they provide important information to the model decision.

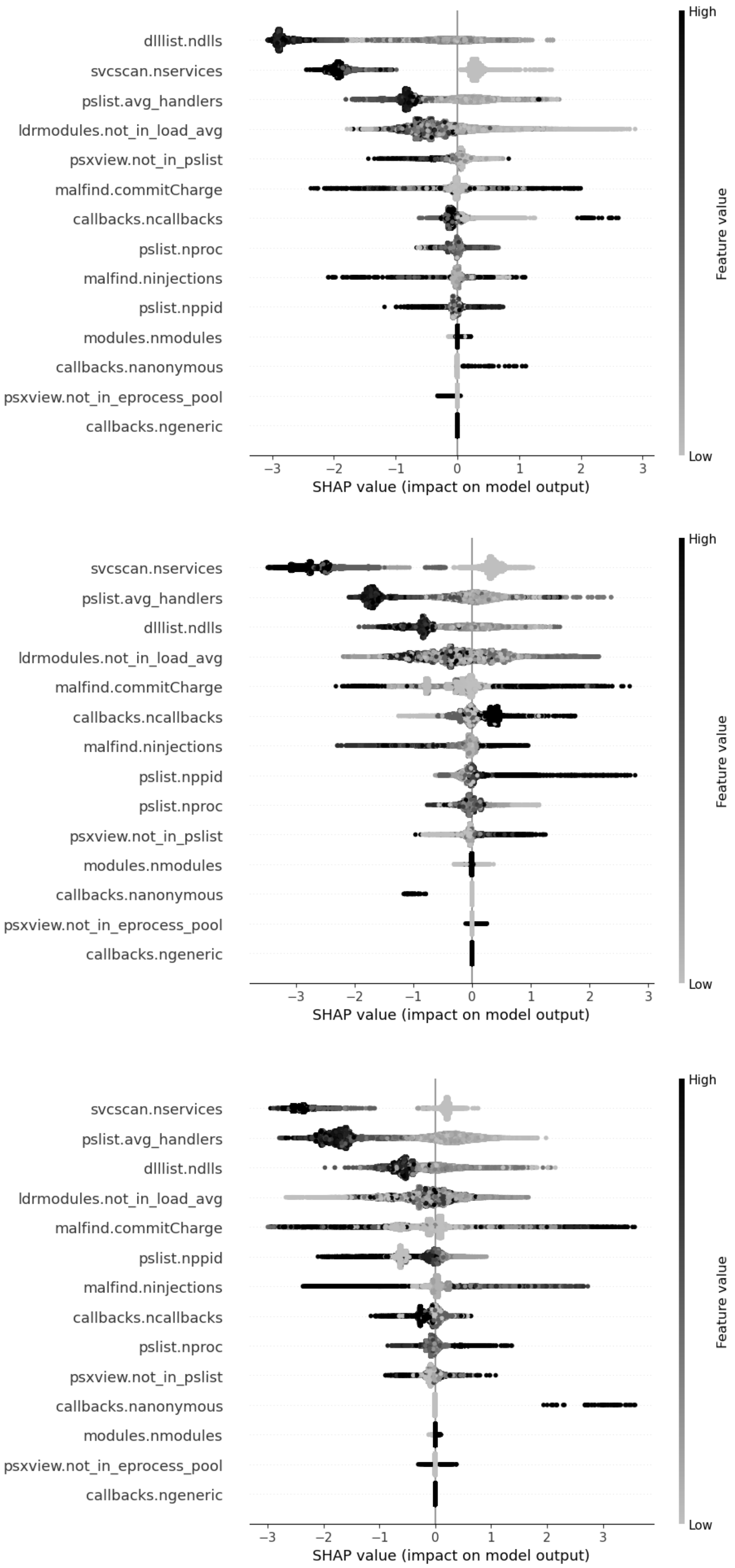


Figure 5.9: Beeswarm plots for the 3 malware categories, Ransomware, Spyware and Trojan from top to bottom

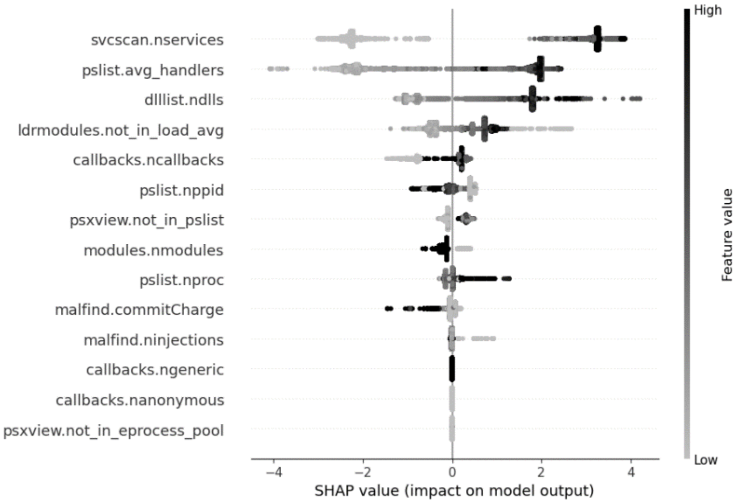


Figure 5.10 : Beeswarm plots for the Benign samples

Although the beeswarm plots give us information about the local importance for the samples, having many samples makes it hard to analyse them. To better understand the values obtained, cluster algorithms can be put up to use to retrieve the maximum information from the local importance.

5.1.2 Creating a new dataframe

To continue the SHAP values analysis, we need to transform the SHAP values in a new dataset while maintaining the original structure. The SHAP and Pandas libraries makes it easy to manipulate the results obtained and adding them to the dataframe. An advantage of this new dataframe is the fact that all variables come in the same measure, feature importance, so we do not need another step to standardize the data.

5.1.3 Reducing to two dimensions

To get a better interpretation of the values obtained for the cluster algorithms and visualisation purposes, we use UMAP to reduce the feature space into 2 variables. In comparison we can look at the differences of the original dataframe and the SHAP values dataframe. The hyperparameters selected were 10 for *n_neighbors* and 0.1 for *min_dist*.

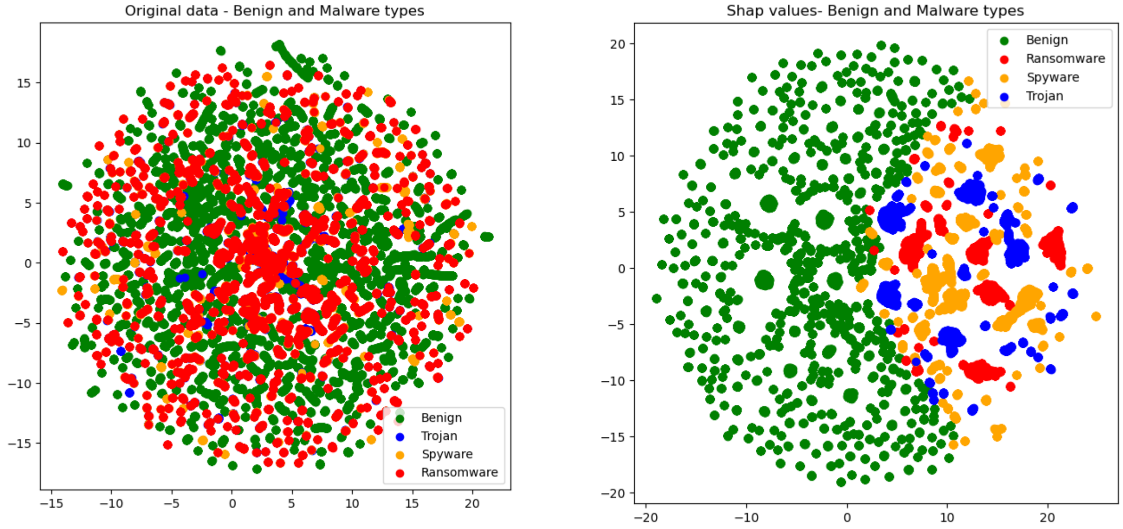


Figure 5.11: UMAP 2D visualisation for both the original data and the SHAP values data

Looking at both graphs, the differences are clear. In Figure 5.11, left graph, it is hard to extract useful information as the malware category samples tend to overlap and there is no clear separation between benign and malware samples. On the other hand, the feature space on Figure 5.11, right graph, is arranged according to the interpretation and classification of the model (SHAP values). There is a clear boundary between benign and malware samples.

Other important aspect to note is that the malware samples got divided into subgroups, noting a distinct behaviour between the different categories and among the categories themselves. Furthermore, we can turn our focus into one particular category or class.

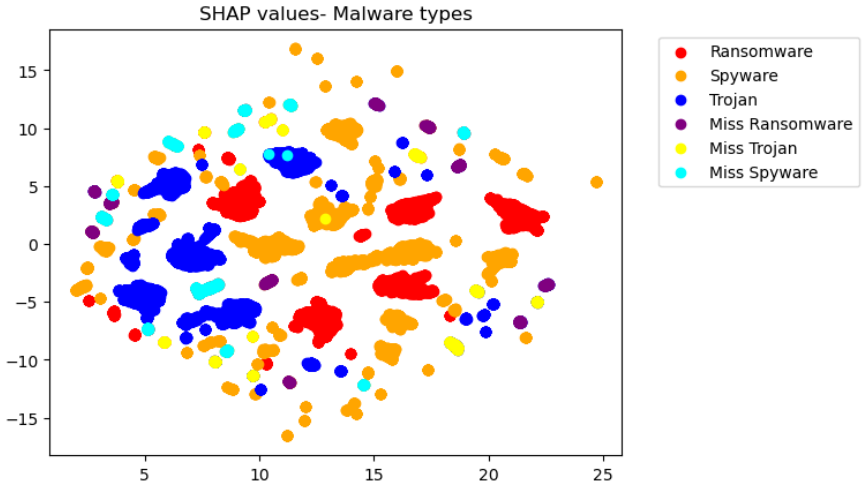


Figure 5.12: UMAP 2D visualization for the malware SHAP values

Analysing the malware samples has a greater importance in Cybersecurity and since benign samples do not present significant groups, we can highlight the malware samples and have a better view of the behaviour of the samples.

Figure 5.12 provides an amplified view of the malware samples. It is possible to see the different groups that emerged with more ease. Both Ransomware and Trojan categories present tight groups while Spyware groups tend to be more louse and disperse. It is also possible to see that the misclassification samples create small groups around the corrected classified groups. Although having missclafied samples might seem a bad indicator at first glance, if the model used is trustworthy enough, this reveals very important information to analysts as they can not only understand the features that lead to that choice but also understand to which samples the model is associating them.

This view allows us to compare the position of the different malware categories but we can also focus the analysis to each malware category and their respective families, taking the maximum advantage of the known labels, Figures 5.13, 5.14 and 5.15 shows exactly that.

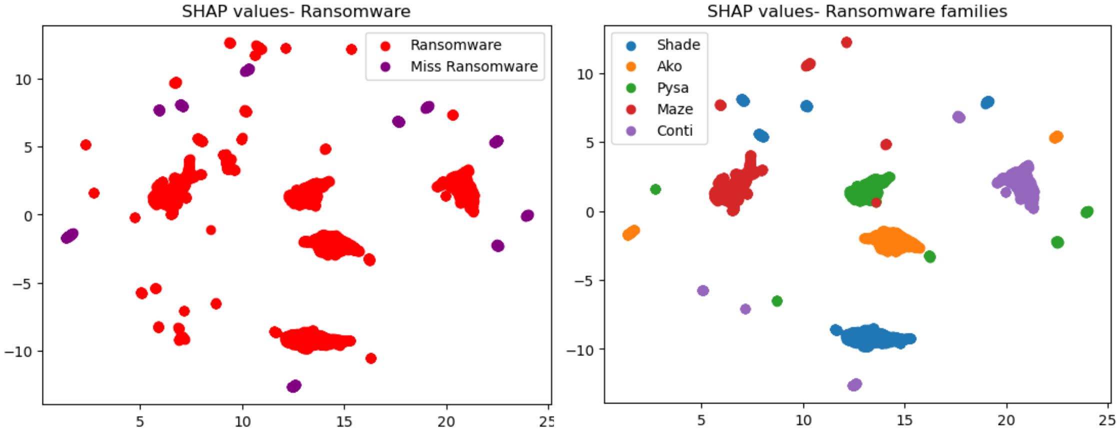


Figure 5.13: Ransomware category and its family's visualisation

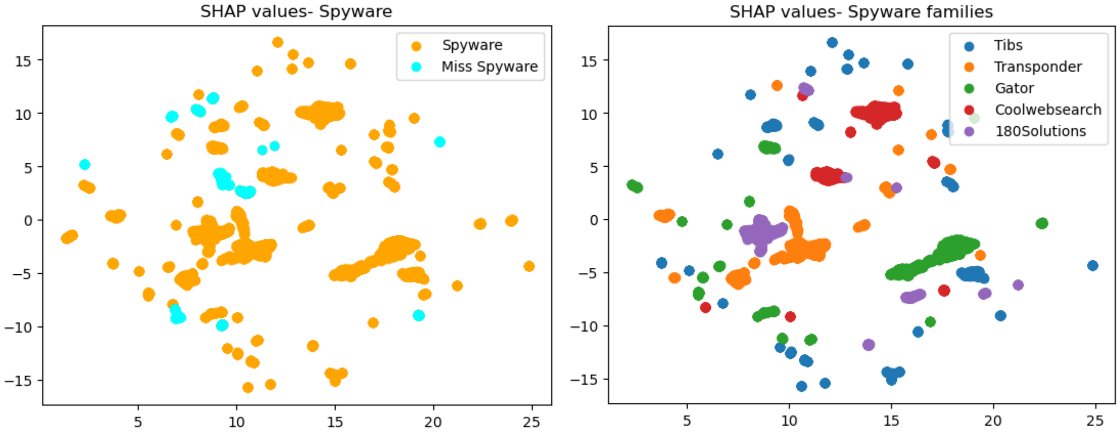


Figure 5.14: Spyware category and its family's visualisation

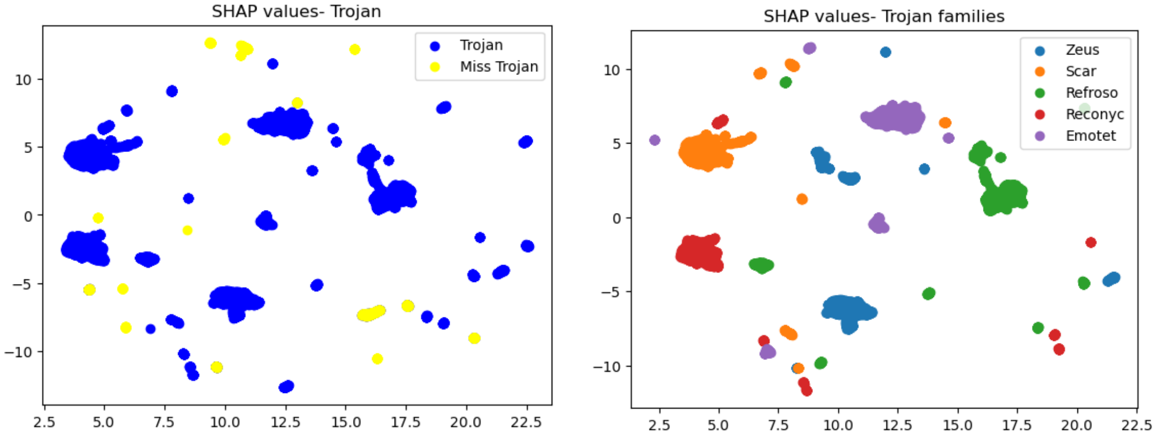


Figure 5.15: Spyware category and its family's visualisation

We can have an exclusive view for each category and its families. It is possible to observe that despite the model not receiving any information about the types of families within each category, it still groups parts of them together. When looking at each category we also need to have in consideration that some samples are misclassified and the model is seeing them as a different category and as a result they end up becoming looser from the visual groups.

Having confirmed that there are distinct groups with their own properties we can proceed with finding the cluster algorithm that better capture the groups visible on the 2D graphs.

5.2 Applying Cluster algorithms

The previous visualisation graphs give us an idea of how the clusters should be formed. Given the information acquired and the cluster algorithms that we have available it is predictable that K-Means will not have a good performance compared to DBSCAN and EM clustering. The reasoning behind this has to do with the fact that K-Means assumes equal spherical clusters and in the 2D visualisations, multiple groups with different shapes and sizes can be drawn. Besides it is likely that the algorithm would get stuck in local optimum as it is sensitive to the cluster's initialization.

DBSCAN and EM clustering identify arbitrary shapes with more ease, however both have their downsides. While DBSCAN is highly sensitive to the hyperparameters, small changes in its values will have great impact on the cluster's creation, EM clustering requires more computer power and suffers the same problems as K-Means as it can be negatively affected by the initialization of the gaussian parameters. Only by applying both cluster algorithms and comparing the output we will be able to determine which one to choose. EM clustering requires the specification of the number of gaussian distributions (clusters), for this purpose the Bayesian information criterion (BIC) was selected to find the optimal value.

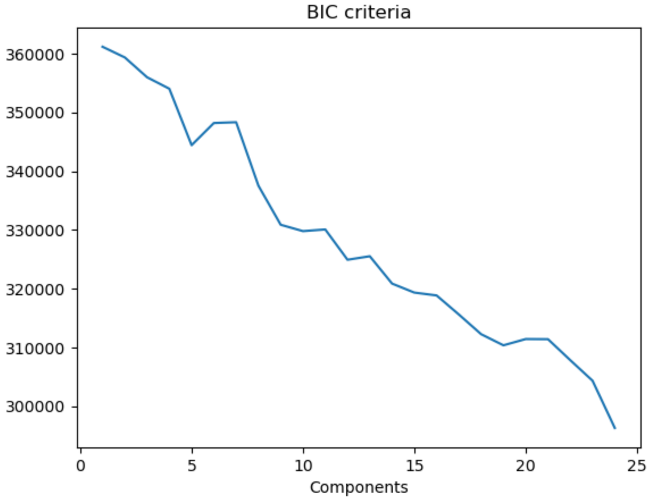


Figure 5.16: BIC criteria to select the number of EM clusters

According to BIC, Figure 5.16, the number of clusters that optimize the EM clustering algorithm is 25 clusters. Lower BIC values could be achieved with the increase of the clusters however it would lead to overfit and extra computational power, so the number of clusters chosen to apply was 20.

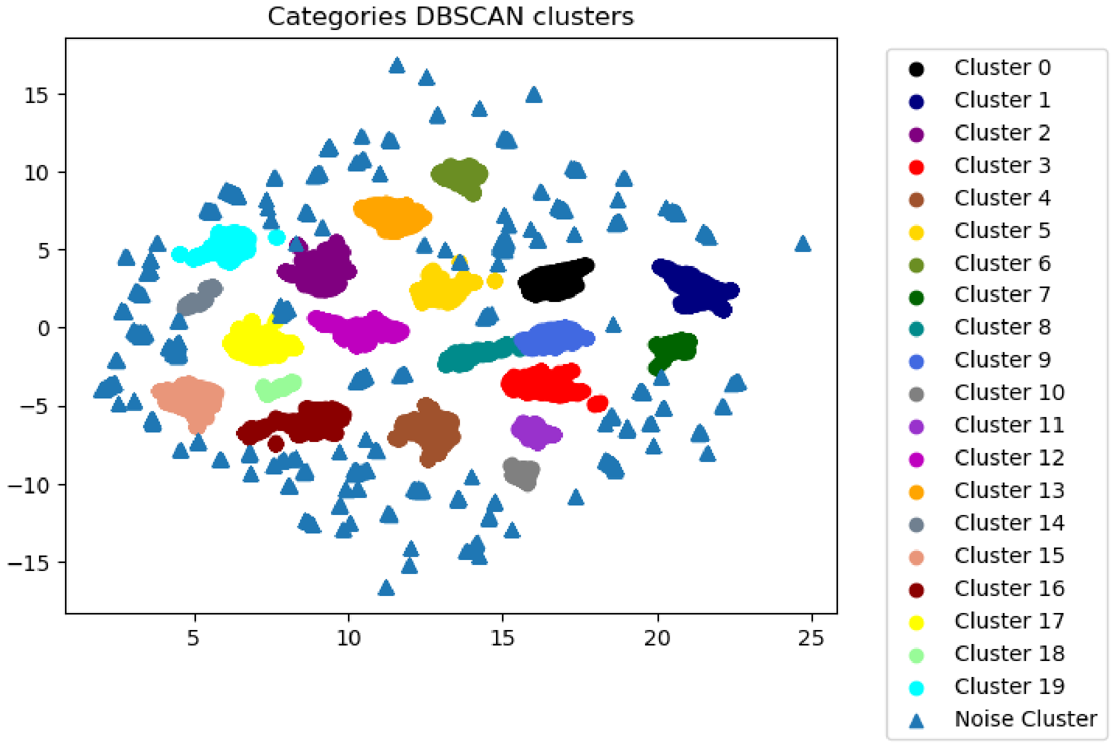


Figure 5.17: DBSCAN clusters

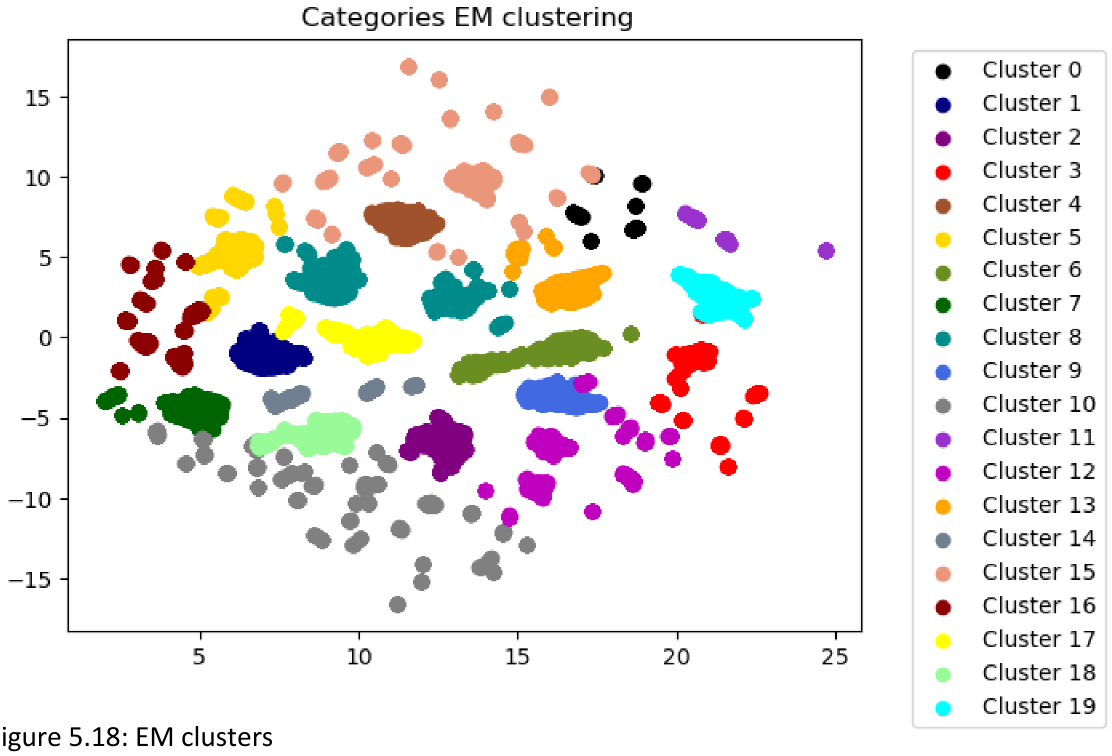


Figure 5.18: EM clusters

DBSCAN algorithm, with the parameters 1 for *eps* and 300 for *min_sample*, identified 21 clusters, as we can see in Figure 5.17, the clusters obtained fit perfectly the groups identified previously. It also tags samples that do not belong to a specific group into a noise cluster, which can be useful for the analysis.

Regarding the EM clustering, the biggest challenge is to select an appropriate number of clusters, with the help of BIC we can have an idea of the number to choose but it is only by looking at the 2D visualisation that we can clarify the number to use. As we see in the Figure 5.18, the EM cluster produced results similar of DBSCAN.

Comparing the clusters algorithms, both capture distinct groups, constructing them in a similar way to the ones visible in the 2D graph. While DBSCAN can identify a clear noise cluster, EM is only capable of associating those samples to the closest cluster which can be useful for interpretation but can also interfere with the cluster quality. The analysts are the ones that will have to make the assumption of analysing the cluster or considering it a noise cluster, with the respective care. Other problem has to do with the fact that while DBSCAN fits the number of clusters automatically, EM clustering requires the prior knowledge of the users to input a specific number of clusters which can reveal to be tricky for analysts to find an optimum. It is also visible that DBSCAN clusters are more compressed to the samples that are close and EM clusters tend to absorb samples that distance from the main group.

The visualization of the clusters suggest that EM is better to analyse anomalies and misclassified samples while DBSCAN can better differentiate the behaviour among the malware samples.

EM cluster 6 absorb all the samples belonging to the spyware groups while DBSCAN divides it into two different clusters, 8 and 9. EM clusters 14 and 15 end up integrating samples that are dispersed across the visualization and that association might disturb the analysis of the feature importance. EM clusters 3 and 12 as associate other samples that are relatively close to the main group, which can facilitate on understanding the behaviour of the outlier samples.

It is hard to choose which cluster is better or not, both can be applied to tackle different problems, so ultimately, the choice of the algorithm cluster will depend on the analyst's goal. For the purpose of this work we are going to use DBSCAN to highlight the different clusters across the categories.

5.3 Cluster Analysis

As it was mentioned previously, knowing the true labels and predicted labels of the data and the model gives the analysts a unique and diverse way of analysing the clusters. For this case we can start by analysing the clusters size and different categories that they include to have an idea of how the samples are divided through the clusters. Then we can focus on the results of Figure 5.17 and analyse the mean SHAP values of each cluster to find the differences of each cluster for the malware class. This view can be drilled down to focus on the clusters for a particular category or respective families of each category.

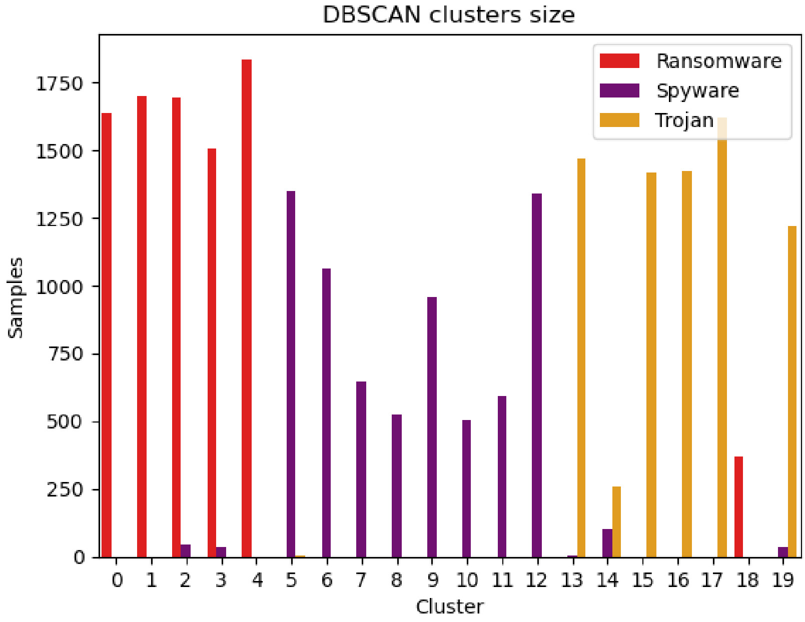


Figure 5.19: Cluster composition according to the different malware categories

Figure 5.19 allows us to see the composition of the clusters according to the predicted categories. Clusters range in size from 360 samples to 1834 samples. Although presenting a dominant category, clusters 2, 3 and 19 have two categories, which has to do with the fact that the category in minority is misclassified. Cluster 14 is the smallest cluster and contains 2 different categories that are not misclassified. Clusters from 0 to 4 and 18 mainly represent Ransomware samples, 5 to 12 spyware samples and 13 to 19, Trojan samples.

The most important step of the cluster analysis it to understand the differences between the clusters and if, in fact, they present different SHAP values and therefore, different feature importance for the model.

To do this analysis we create a second mean global view for each cluster. This will allow us to identify the behaviour of the samples of each cluster and identify the possible differences. The next set of figures shows the different mean SHAP values for each cluster with the objective of comparing them and analysing their differences.

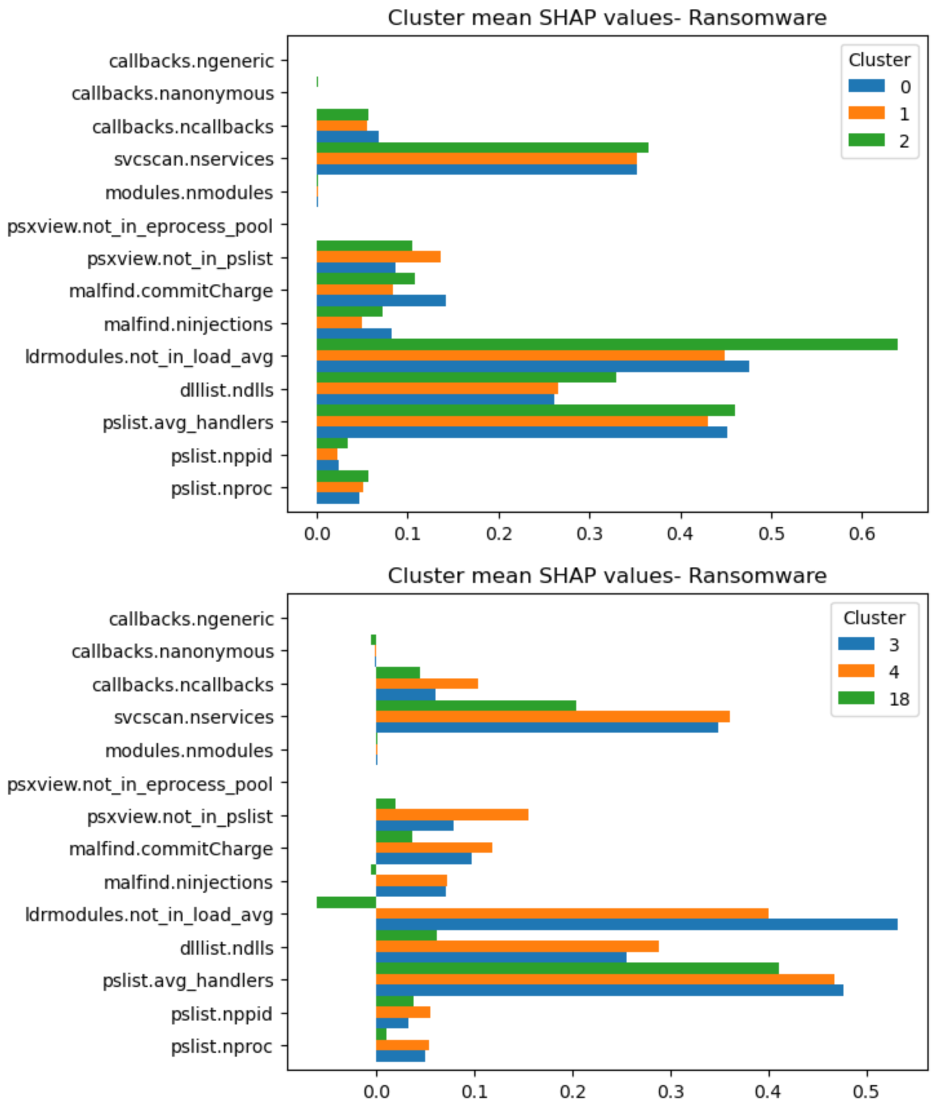


Figure 5.20: Mean SHAP value of each future for the different Ransomware clusters

Focusing first on the clusters that contain the Ransomware samples, it is possible to see that the order of the SHAP values slightly changes across the first four clusters. *Ldrmodules.not_in_load_avg* was the feature with most impact for the clusters apart from cluster 4 and 18. It was also the feature with the most differences between each cluster. *Pslist.avg_handlers, dlllist.ndlls* and *svcscan.nservices*, were the following features that more contributed to the model decision making. Cluster 18 stands out as a contrast to the other ransomware clusters, it has a negative value for *ldrmodules.not_in_load_avg* which is the complete opposite of the other clusters and it also has lower values for all the features that stand out on the other clusters. The remaining ransomware clusters accumulate small differences that can also be relevant to the analysis, for instance, cluster 4 and 1 have the biggest values for *psxview.not_in_pslist,* cluster 0 has bigger values for *malfind.commitCharge* and cluster 3 has the biggest values for *pslist.avg_handlers*.

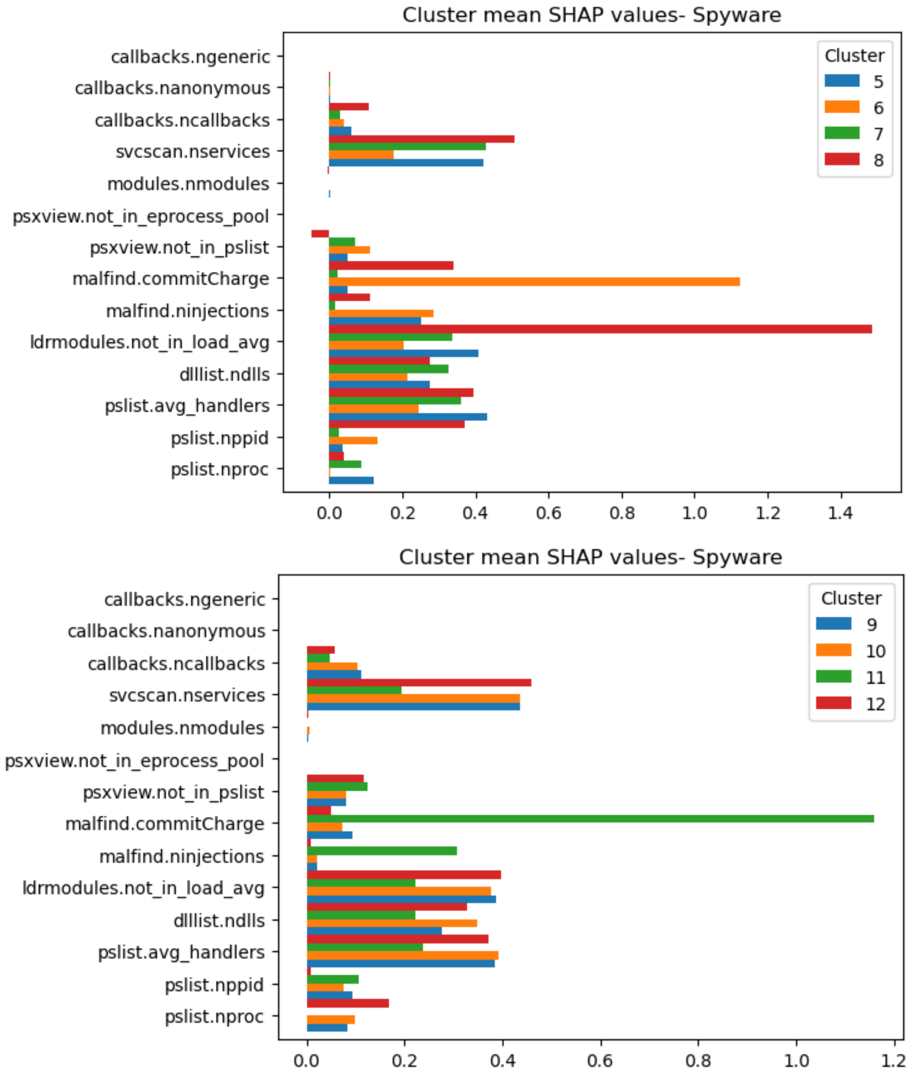


Figure 5.21: Mean SHAP value of each future for the different Spyware clusters

Moving to the Spyware clusters, 5 to 12, we can see that unlike the Ransomware clusters, the differences in these ones are more noticeable. Cluster 8 has the feature ldrmodules.not*_in_load_avg* with the highest feature importance value across all clusters achieving a value over 1.4 while the other clusters do not go beyond 0.5. Cluster 11 and 6 have *malfind.commitCharge* has the highest feature importance, distancing themselves of the other clusters by a large margin. The other spyware clusters present similar feature importance values. They compensate the difference between the spyware clusters previously mentioned by accumulating small differences on the remaining features. *Svscan.nservices* and *pslist.avg_handlers* represent the higher feature importance for this group.

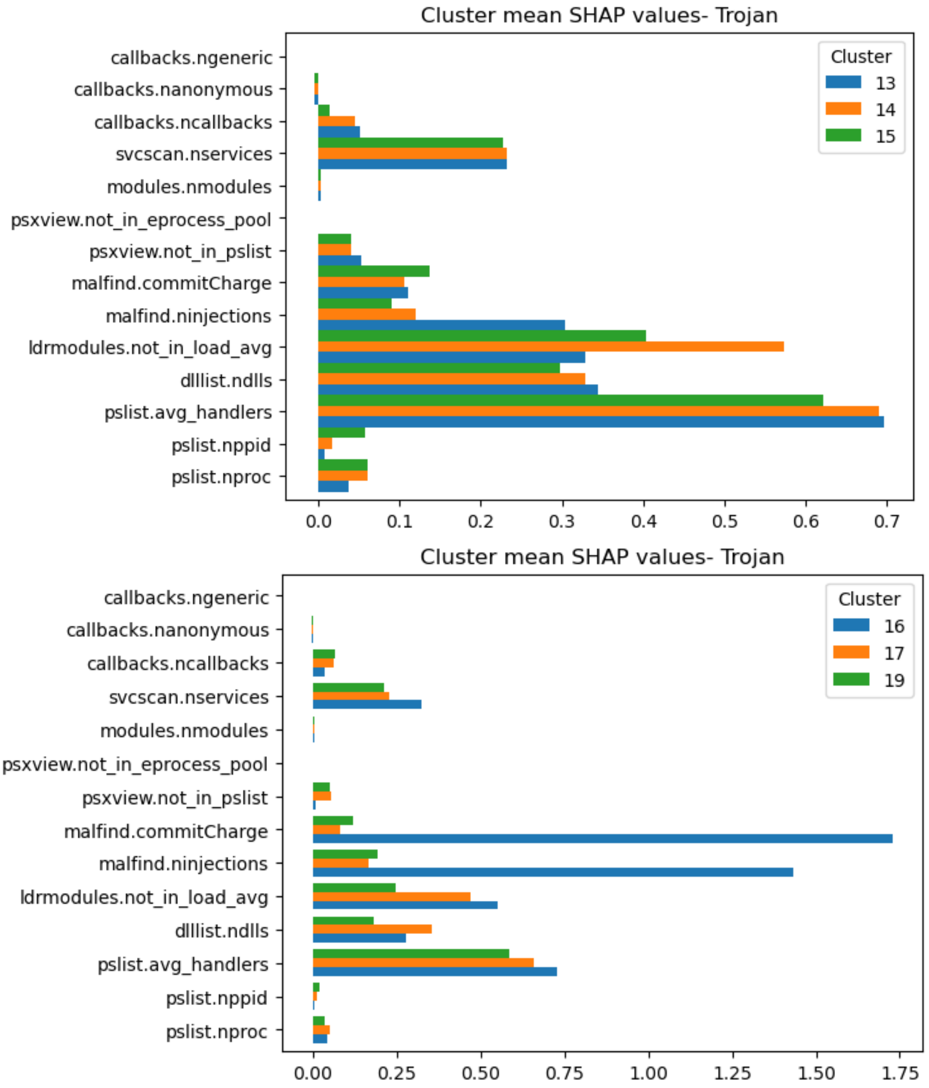


Figure 5.22: Mean SHAP value of each future for the different Trojan clusters

The last set of clusters, from 13 to 19, are composed by samples from the Trojan category with the exception of cluster 18. Cluster 14, which is composed of trojan and spyware samples, stands out for having the highest feature importance for *ldrmodules.not_in_load_avg*. Cluster 16 has a higher value for both *malfind.ninjections* and *malfind.commitCharge* compared to the other clusters. Cluster 13 and 15 are pretty much identical, the only exception has to do with the feature *ldrmodules.not_in_load_avg* that stands out the most for cluster 15 and *pslist.avg_handlers* that stands more for the cluster 13. Clusters 17 and 19 almost have a stair shape, they focus on the same features but each has considerable differences that distincts them.

Recalling the figures 5.9 and 5.10 we can see that this cluster analysis is extremely useful to complement both global and local importance that the SHAP values offer. As it was seen, each cluster has its unique properties that differ from the global view of the model. It was possible to observe the different categories and their unique feature importance as well as it was possible to identify the differences inside each category. This can help analysts understanding how specific samples from the same category differ from each other and how the model is interpreting those differences.

Furthermore, the supervised environment gives room to maneuver according to the target of the analysts, being able to change the analysis focus into a particular set of samples and observing their clusters or creating new clusters just for that set. For instance, the analysis can be focused on the misclassified samples and how they position themselves against the correctly classified samples.

6 Conclusion

This work takes advantage of the local properties of SHAP values to search for different patterns in samples that belong to the same class. Often researchers only use the local properties to explore isolated cases and most of the times act based on the global feature importance. As it was observed it could not be possible to find reliable clusters within the original data as samples from the different categories and families were overlaping, however, the clusters found using SHAP values are reliable and have different feature importance levels which can complement and bring further interpretation to the model decision as well as helping analysts with the decision-making process, especially on understanding patterns and misclassifications of the model. This methodology can help on facing some current problems on the cybersecurity industry related with the interpretability and adversarial learning of the models. It can boost the interpretation of black-box models making them more desirable to use as the analysis becomes more reliable and complete. This interpretability will also contribute to the adversarial learning problem since it gives extra vision to the analysts as they are able to have more control on the possible modified samples, allowing to know to which samples they are grouped with.

Although SHAP values and supervised clustering can prove to be very helpful, they present some relevant issues. They are highly reliable on the model deployed, so it is crucial to assure that the model is well defined and it is maintained once in a time so the SHAP values produced are trustworthy to analyse. For this work, the computation cost was optimized by reducing the feature space and applying the TreeExplainer, however this is not always possible to keep under control and the SHAP values might take a lot of resources to be computed, especially if we are not dealing with tree models and have complex feature spaces, compromising the analysis in feasible time.

Regarding further work opportunities, although this work focuses on a narrow scope, it is possible to implement this idea on different problems of other industries, where samples from the same classes might present different behaviours such as credit scoring and medicine field. With the raising of Deep Learning models, which are more accurate but also harder to implement and interpret, and the fact that SHAP Values can be used on these types of models such as neural networks. It can be interesting to apply this methodology on Deep learning models to understand if it can contribute to the model interpretability, taking in consideration the extra computation power that it would require compared to other simpler models. Finally, it would also be interesting to apply this methodology in time series data and study how the model decision and interpretation was changing through the time.

References

Alenezi R, Ludwig SA (2021) Explainability of cybersecurity threats data using SHAP. In: 2021 IEEE symposium series on computational intelligence (SSCI). IEEE, pp 01–10

Basole, S., Stamp, Mark, Cluster Analysis of Malware Family Relationships (2021) arXiv:2103.05761

Charmet, F., Tanuwidjaja, H.C., Ayoubi, S. et al. Explainable artificial intelligence for cybersecurity: a literature survey. Ann. Telecommun. 77, 789–812 (2022).

Cooper, A., Doyle, O., Bourke, A. (2021). Supervised Clustering for Subgroup Discovery: An Application to COVID-19 Symptomatology. In:, et al. Machine Learning and Principles and Practice of Knowledge Discovery in Databases. ECML PKDD 2021. Communications in Computer and Information Science, vol 1525. Springer, Cham.

D. E. Denning, "An Intrusion-Detection Model," IEEE Transactions on Software Engineering, vol. 13, no. 2, pp. 222–232, 1987.

Gibert, D., Mateu, C., Planes, J., "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges", *(2020)* Journal of Network and Computer Applications*,* 153, art. no. 102526

J. B. Fraley and J. Cannady, "The promise of machine learning in cybersecurity," SoutheastCon 2017, Concord, NC, USA, 2017, pp. 1-6, doi: 10.1109/SECON.2017.7925283.

Khorolska, K., Lazorenko, V., Bebeshko, B., Desiatko, A., Kharchenko, O., Yaremych, V. (2022). Usage of Clustering in Decision Support System. In: Raj, J.S., Palanisamy, R., Perikos, I., Shi, Y. (eds) Intelligent Sustainable Systems. Lecture Notes in Networks and Systems, vol 213. Springer, Singapore.

Ludenberg, S. and Lee, S. (2017), 'A Unified Approach to Interpreting Model Predictions', *Advances in Neural Information Processing Systems,* Volume 2017-December, Pages 4766 – 4775

Lundberg S., Erion G, Chen H, DeGrave A, Prutkin JM, Nair B, Katz R, Himmelfarb J, Bansal N, Lee SI. From Local Explanations to Global Understanding with Explainable AI for Trees. Nat Mach Intell. 2020 Jan;2(1):56-67.

M. Wang, K. Zheng, Y. Yang and X. Wang, "An Explainable Machine Learning Framework for Intrusion Detection Systems," in IEEE Access, vol. 8, pp. 73127-73141, 2020

McInnes, L, Healy, J, UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction, ArXiv e-prints 1802.03426, 2018

R. Lippmann, R. K. Cunningham, D. J. Fried, I. Graf, K. R. Kendall, S. E. Webster, and M. A. Zissman, "Results of the 1998 DARPA Off-line Intrusion Detection Evaluation," in Proc. Recent Advances in Intrusion Detection, 1999.

Rathore, H., Sahay, S.K., Thukral, S., Sewak, M. (2021). Detection of Malicious Android Applications: Classical Machine Learning vs. Deep Neural Network Integrated with Clustering. In Gao, H., J. Durán Barroso, R., Shanchen, P., Li, R. (eds) Broadband Communications, Networks and Systems. BROADNETS 2020. Lecture notes of the Institute for Computer Sciences, Social and Informatics and Telecommunications Engineering, vol 355. Springer, Cham

Renato Cordeiro de Amorim, Carlos David Lopez Ruiz, Identifying meaningful clusters in malware data, Expert Systems with Applications, Volume 177, 2021, 114971, ISSN 0957-4174

Shapley, L. S., "17. A Value for n-Person Games". Contributions to the Theory of Games (AM-28), Volume II, edited by Harold William Kuhn and Albert William Tucker, Princeton: Princeton University Press, 1953, pp. 307-318.

Tiani Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD ’16). Association for Computing Machinery, New York, NY, USA, 785-794.

Tristan Carrier, Princy Victor, Ali Tekeoglu, Arash Habibi Lashkari,” Detecting Obfuscated Malware using Memory Feature Engineering”, The 8th International Conference on Information Systems Security and Privacy (ICISSP), 2022

Y. Xin *et al*., "Machine Learning and Deep Learning Methods for Cybersecurity," in *IEEE Access*, vol. 6, pp. 35365-35381, 2018, doi: 10.1109/ACCESS.2018.2836950.