Paper accepted in Expert Systems with Applications (Elsevier), to appear in 2021. http://doi.org/10.1016/j.eswa.2021.114971

Identifying meaningful clusters in malware data

Renato Cordeiro de Amorim[a,*], Carlos David Lopez Ruiz[a]

[a] *School of Computer Science and Electronic Engineering, University of Essex, Wivenhoe Park CO4 3SQ, UK.*

Abstract

Finding meaningful clusters in drive-by-download malware data is a particularly difficult task. Malware data tends to contain overlapping clusters with wide variations of cardinality. This happens because there can be considerable similarity between malware samples (some are even said to belong to the same family), and these tend to appear in bursts. Clustering algorithms are usually applied to normalised data sets. However, the process of normalisation aims at setting features with different range values to have a similar contribution to the clustering. It does not favour more meaningful features over those that are less meaningful, an effect one should perhaps expect of the data pre-processing stage. In this paper we introduce a method to deal precisely with the problem above. This is an iterative data pre-processing method capable of aiding to increase the separation between clusters. It does so by calculating the within-cluster degree of relevance of each feature, and then it uses these as a data rescaling factor. By repeating this until convergence our malware data was separated in clear clusters, leading to a higher average Silhouette width.

Keywords: feature rescaling, drive-by-download malware, clustering.

1. Introduction

The term malware is used to describe malicious software, which has been designed with the specific purpose of exploiting vulnerabilities in computer systems. In general, any given malware aims at compromising such systems. Malware can be further divided into non-exclusive categories such as trojan, virus, adware, worms, and others. Malware development may have had an innocent start, but there are now multiple examples suggesting this is a multi-million business sometimes associated with organised crime (Fox, 2017b,a; Tidy, 2019; BBC, 2020a). Malware has also been used in acts of sabotage, and may even have political motivations (BBC, 2020b).

Here, we are interested in clustering malware data. That is, identifying k homogeneous groups (i.e. clusters, for a longer discussion of what a cluster is, see Hennig (2015) and references therein) of malware in a given data set — without the need for labelled samples for an algorithm to learn from. Once one is able to say that a new malware sample should be assigned to a cluster containing homogeneous malware samples it becomes easier to create defence mechanisms. Clustering algorithms (for complete reviews, see Mirkin (2012); Aggarwal and Reddy (2014) and references therein) can be usually divided in two groups: partitional and hierarchical. The latter includes algorithms able to produce a clustering as well as information regarding the relationships that exist between clusters (this information can be represented with the help of a dendrogram). Partitional clustering algorithms identify k disjoint clusters in a given data set, so that each object (a malware sample, in our case) in the data set is assigned to a single cluster. Although hierarchical algorithms produce information one would usually associate with families (i.e. a tree), it comes at a computational cost. In 2018 alone, 246,002,762 new malware variants were found (Symantec, 2019). Hence, we see partitional clustering as a more realistic approach for the real-world.

There are various examples of clustering algorithms applied to malware data in the literature (for instance, Faridi et al. (2018); Asquith (2016); Cordeiro de Amorim and Komisarczuk (2012) and references therein). However, these apply classical normalisation to the data sets (for details, see Section 3). This type of normalisation (e.g. z-score, range normalisation, unit length, among others) aims at putting all features used to describe an object at the same level. It does not favour more meaningful features over those that are

*Corresponding author

Email addresses: r.amorim@essex.ac.uk (Renato Cordeiro de Amorim), cl18417@essex.ac.uk (Carlos David Lopez Ruiz)

Paper accepted in Expert Systems with Applications (Elsevier), to appear in 2021. http://doi.org/10.1016/j.eswa.2021.114971

less meaningful. There is considerable similarity between malware samples (at times some are even said to belong to the same family). Hence, it is reasonable to expect that there will be a considerable amount of overlap between clusters. Also, malware samples have a tendency to be released in bursts with a skewed distribution (Song et al., 2016). This scenario makes clustering particularly difficult.

In this paper we introduce a novel method to deal with the problem described above. Our method is capable of increasing the separation between clusters during the data pre-processing stage. It does so by calculating the within-cluster degree of relevance of each feature in a given data set, and using this as a rescaling factor. By iterating this process our method increases the quality of malware clusterings, as measured by the average Silhouette index (Rousseeuw, 1987). We apply our method to drive-by-download malware data. This referrers to malware delivered to client systems that browse resources on the web, usually via http. This is a very timely issue given that in 2019 one in ten unique resource locators (urls) were found to be malicious (Symantec, 2019).

The remainder of this paper is organised as follows. Section 2 discusses the clustering algorithms that are directly relevant to our research, as well as a method to measure how good a clustering is. Section 3 briefly explains the classical normalisation algorithms used in the pre-processing stage. Section 4 explains our method, together with its mathematical motivation. Sections 5 and 6 explain our process of data gathering, our methodology, and the results we have obtained. Finally, Section 7 presents our conclusions, and indications of future work.

2. Related work

The literature on automatic malware analysis is rather vast. Traditional machine learning methods when applied to malware analysis often require a considerable effort in terms of feature engineering, as well as considerable computational resources (Xiao et al., 2020). The former relates to the dependency on what essentially are hand-designed features based on expert knowledge. This difficulty, in particular, has lead to research effort being directed to the production of algorithms capable of automatically combine different types of features (see for instance Gibert et al. (2020), and references there in). Recently, there seems to be a tendency of modern methods in treating a malware sample as an image (for a recent survey, see Komatwar and Kokare (2020)). Initially this was done by converting malware samples into grayscale images but now there are some examples of methods capable of using colour images e.g. (Vasan et al., 2020). Representing malware samples as images may seem like an unexpected direction but it does open the doors to the use of computer vision methods such as Deep convolutional neural networks. Hence, malware classification based on malware images and deep learning has become a strong direction of research (Yuan et al., 2020).

However, the above describes classification algorithms. Such methods require an extensive and correct set of labelled samples to learn patterns from. If this requirement is not met, these algorithms are unable to produce a classification. The problem we deal with in this paper is considerably harder. We do not assume the existence of a large and correct data set in which each malware is labelled by family name. Given the existence of different malware taxonomies produced by different AntiVirus companies, we see our approach to be of particular interest to the community.

Given the above, most of the work we discuss in this section relates to clustering algorithms one could use to cluster malware samples. Partitional clustering algorithms aim at identifying a set $S = \{S_1, S_2, ..., S_k\}$ so that each $S_l \in S$ contains homogeneous objects, and $\forall(S_l, S_j \in S)$, $l \neq j \iff S_l \cap S_j = \emptyset$. K-means (MacQueen et al., 1967) is arguably the most popular such algorithm (Fränti and Sieranoja, 2019; Jain, 2010; Steinley, 2006). Given a data set $\mathcal{X}$ containing n objects, each described over d features, k-means minimises the within-cluster distance

$$W(S, Z) = \sum_{l=1}^{k} \sum_{x_i \in S_l} \sum_{v=1}^{d} (x_{iv} - z_{lv})^2, \qquad (1)$$

where $z_l \in Z$ is the centroid of cluster $S_l \in S$, that is, the d-dimensional point with the lowest sum of distances to all objects in S_l. We defined z_l as a point to make it clear that it may or may not belong to $\mathcal{X}$. The k-means algorithm iteratively minimises (1) with three simple steps:

1. Select k objects from $\mathcal{X}$ uniformly at random, and use their values to initialise $z_1, z_2, ..., z_k$.
2. Assign each $x_i \in \mathcal{X}$ to the cluster S_l represented by the centroid z_l that is the nearest to x_i.
3. Update each $z_l \in Z$ to the centre of S_l.

The k-means criterion (1) applies the Euclidean squared distance. Hence, the centre of a cluster S_l is the component-wise mean of its objects, that is $z_{lv} = |S_l|^{-1} \sum_{x_i \in S_l} x_{iv}$ for $v = 1, 2, ..., d$.

As popular as k-means may be, it does have known weaknesses. The most relevant in this paper are: (i) the

Paper accepted in Expert Systems with Applications (Elsevier), to appear in 2021. http://doi.org/10.1016/j.eswa.2021.114971

final clustering depends heavily on the initial set of centroids, which are usually found at random. Suboptimal initial centroids are likely to lead the algorithm to a local minima solution; (ii) k-means requires the number of clusters, k, to be known beforehand; (iii) all features are treated as if they were equally relevant, which is rather unlikely in real-world data sets.

There has been a considerable research effort to address the weaknesses above. For instance, k-means++ (Arthur and Vassilvitskii, 2007) selects one object $x_i \in \mathcal{X}$ uniformly at random, and then copies its values to the first centroid z_1. All other initial centroids are selected following a weighted probability that is proportional to the distances between objects and their nearest centroid.

1. Set $l = 1$. Select an object from $\mathcal{X}$ uniformly at random and copy its values to z_l.
2. Increment l by one. Select an object $x_j \in \mathcal{X}$ at random, with probability $\frac{D(x_j)^2}{\sum_{x_i \in \mathcal{X}} D(x_i)^2}$ and copy its values to z_l, setting $Z = Z \cup \{z_l\}$.
3. Repeat the steps above until $l = k$.
4. Run k-means using each $z_l \in Z$ as an initial centroid.

In the algorithm above $D(x_i)$ is the distance between $x_i \in \mathcal{X}$ and its nearest centroid $z_l \in Z$. Experiments show that k-means++ has a faster convergence to a lower criterion output (1) than the traditional k-means algorithm (Arthur and Vassilvitskii, 2007). This algorithm has enjoyed considerable popularity, and it is now the default k-means option in MATLAB (MATLAB, 2019) and scikit-learn (Pedregosa et al., 2011).

The intelligent k-means algorithm (*ik*-means) (Mirkin, 2012) addresses both weaknesses (i) and (ii) by identifying good initial centroids for k-means as well as the number of clusters in $\mathcal{X}$. This algorithm iteratively identifies anomalous clusters, afterwards the centroids of these anomalous clusters are then used as initial centroids for k-means. In this, k is set to the number of anomalous clusters in $\mathcal{X}$.

1. Set c to the centre of $\mathcal{X}$, and $Z' = \emptyset$. Identify $x_t \in \mathcal{X}$, the object that is the furthest from c.
2. Apply k-means to the data set using x_t and c as initial centroids, but do not allow c to move in the cluster update step. This will lead to clusters S_t and S_c with centroids z_t and c, respectively.
3. If $|S_t| \geq \theta$, set $Z' = Z' \cup \{z_t\}$. In any case, remove each $x_i \in S_t$ from $\mathcal{X}$.
4. If $|\mathcal{X}| > 0$ go to Step 1. Otherwise run k-means by setting $k = |Z'|$, and using each $z_l \in Z'$ as an initial centroid.

In the above θ is a user-defined parameter that helps to avoid small clusters in S, should this be of interest to the user. If the value of k is known one can sort the elements of Z' by the cardinality of their initial clusters (i.e. their value of $|S_t|$ in Step four), and keep only the k elements of Z' with the highest cardinality (this would happen between Steps three and four). Another approach would be to select the k elements of Z' in the order they were found. This way they would be the k most anomalous initial centroids.

We can see that *ik*-means identifies each centroid $z_t \in Z'$, and related cluster S_t by iteratively minimising

$$P(S, Z) = \sum_{x_i \in S_t} \sum_{v=1}^{d} (x_{iv} - z_{tv})^2 + \sum_{x_j \in S_c} \sum_{v=1}^{d} (x_{jv} - c_v)^2, \quad (2)$$

where c is the component-wise mean of $\mathcal{X}$.

In order to address all three weaknesses (i), (ii) and (iii), we introduced the intelligent Minkowski weighted k-means (*imwk*-means) (De Amorim and Mirkin, 2012). This extends *ik*-means by following the intuitive idea that a given feature v may have different degrees of relevance at each cluster $S_l \in S$. We model this behaviour by introducing w_{lv}, the weight of feature v at cluster S_l. The higher w_{lv} is, the higher the contribution of v at cluster S_l is to the clustering. First, we define the weighted Minkowski distance between x_i and z_l as

$$d(x_i, z_l) = \sum_{v=1}^{d} w_{lv}^p |x_{iv} - z_{lv}|^p. \quad (3)$$

The above is in fact the pth power of the Minkowski distance, which is analogous to the use of the Euclidean squared distance in k-means. This approach saves the computational effort of calculating pth roots, and does not change the clusterings produced by the algorithm. The *imwk*-means algorithm minimises

$$W(S, Z, W) = \sum_{l=1}^{k} \sum_{x_i \in S_l} \sum_{v=1}^{d} w_{lv}^p |x_{iv} - z_{lv}|^p \quad (4)$$

subject to

$$\begin{cases} S_l \cap S_j = \emptyset \text{ for } l, j = 1, 2, ..., k \text{ and } l \neq j; \\ w_{lv} \geq 0 \text{ for } l = 1, 2, ..., k \text{ and } v = 1, 2, ..., d; \\ \sum_{v=1}^{d} w_{lv} = 1 \text{ for } l = 1, 2, ..., k; \\ p \geq 1. \end{cases} \quad (5)$$

Leading to

$$w_{lv} = \left(\sum_{u=1}^{d} \left[\frac{D_{lv}}{D_{lu}} \right]^{1/(p-1)} \right)^{-1}, \quad (6)$$

Paper accepted in Expert Systems with Applications (Elsevier), to appear in 2021. http://doi.org/10.1016/j.eswa.2021.114971

where D_{lv} is the dispersion of feature v at cluster S_l, given by $\sum_{x_i \in S_l} |x_{iv} - z_{lv}|^p$. We usually add a small constant to each dispersion (0.001, say) to avoid a division by zero in (6) when v perfectly discriminates S_l (i.e. for all $x_i \in S_l$, x_{iv} has the same value). The *imwk*-means algorithm can be described as follows.

1. Set c to be the Minkowski centre of $\mathcal{X}$, $\mathcal{X}'$ to be a copy of $\mathcal{X}$, and each $w_{lv} = d^{-1}$.
2. Find the object $x_t \in \mathcal{X}'$ that is the farthest from c using (3), and copy its values to z_t.
3. Assign each $x_i \in \mathcal{X}'$ to either S_t or S_c, depending on which centroid is the nearest to x_i (z_t or c) as per (3). If this step does not change either S_t or S_c, go to Step 6.
4. Update z_t to the Minkowski centre of its cluster S_t. Update each w_{lv} as per (6). Go back to Step 3.
5. If $|S_t| \geq \theta$, add z_t to Z and w to W. In any case, remove all objects $x_i \in S_t$ from $\mathcal{X}'$. If $|\mathcal{X}'| > 0$ go to Step 2.
6. Assign each $x_i \in \mathcal{X}$ to the cluster S_l whose centroid z_l is the nearest to x_i as per (3). If this step produces no change to to any $S_l \in S$, stop.
7. Update each $z_l \in Z$ to the Minkowski centre of its cluster S_l. Update each w_{lv} as per (6). Go back to Step 6.

The Minkowsi centre of a feature v at cluster S_l with an exponent p is the value μ leading to the lowest $\gamma(p) = \sum_{x_i \in S_l} |x_{iv} - \mu|^p$. Notice $\gamma(p)$ is a convex function. Hence, one can approximate its minimum by setting $\mu = |S_l|^{-1} \sum_{x_i \in S_l} x_{iv}$, and then keep moving μ by a small number (0.0001, say) to the side that minimises $\gamma(p)$.

If one knows how many clusters a data set has, we can re-state step 5 as "Add z_t to Z and w to W. Remove all objects $x_i \in S_t$ from $\mathcal{X}'$. If $|\mathcal{X}'| > 0$ go to Step 2". This would require a new step between 5 and 6: "Keep in Z and W only the elements related to the k clusters with the highest cardinality.", which is the approach used in the original paper (see De Amorim and Mirkin (2012)). Of course, very much like in *ik*-means it is also possible to remove all but the first k tentative centroids from Z.

A suitable Minkowski exponent p can be found using a consensus clustering approach (de Amorim et al., 2017). This requires one to run *imwk*-means with values of p from 1.1 to 5.0 in steps of 0.1, leading to 40 clusterings. The chosen p is that of the clustering with the highest average similarity to all other 39 clusterings, usually measured using the Adjusted Rand Index (ARI) (Rand, 1971). Given two clusterings $S = \{S_1, S_2, ..., S_k\}$ and $U = \{U_1, U_2, ..., U_r\}$, the ARI is defined as

$$ARI(S,U) = \frac{\sum_{ij}\binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}}{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}] - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}] / \binom{n}{2}}, \tag{7}$$

where $n_{ij} = |S_i \cap U_j|$, $a_i = \sum_{j=1}^{r} |S_i \cap U_j|$, $b_j = \sum_{i=1}^{k} |S_i \cap U_j|$. The ARI is corrected for chance.

In terms of hierarchical clustering, the Ward's method (Ward Jr, 1963) is certainly among those most frequently used (see for instance Govender and Sivakumar (2020); Barak and Mokfi (2019); Abboud et al. (2019); Talpalaru et al. (2019), and references there in). This algorithm follows the agglomerative approach so it begins with n clusters, each with a single object assigned to it. It then merges the two clusters which would lead to the lowest increase in the total within-cluster variance. That is, the two clusters with the lowest

$$d_w(S_l, S_t) = \frac{|S_l||S_t|}{|S_l| + |S_t|} d(z_l, z_t), \tag{8}$$

where $d(z_l, z_t)$ is the Euclidean distance between the centroids of S_l and S_t (z_l and z_t, respectively). The merges may proceed until the algorithm reaches a user defined value of k, or until $k = 1$ so that it produces a whole hierarchy of clusters. Thanks to its popularity, the Ward's method is frequently found in software packages used for data analysis (e.g. MATLAB MATLAB (2019), scikit-learn Pedregosa et al. (2011), and R R Core Team (2020)).

DBSCAN (Ester et al., 1996) is a classic clustering algorithm that is still very much in use today (see for instance Lai et al. (2019); Galán (2019); Schubert et al. (2017), and references therein). DBSCAN follows a density-based approach, unlike the other clustering algorithms discussed in this section. In this approach a cluster is simply seen as an area of high-density (i.e. an area with a relatively high number of nearby objects), allowing this algorithm to identify non-spherical clusters. DBSCAN has two extra parameters: ϵ specifying the radius of a neighbourhood with respect to some object, and *minPts* defining the minimum number of objects required to form a dense region. This algorithm begins by identifying the core objects in a data set, that is, the objects which have at least *minPts* neighbours within a distance of ϵ. Any such neighbours are said to be directly reachable from the core object in question. Objects are said to be reachable from one another if there is a path of directly reachable objects between them. Each cluster has at least one core object, as well as all reachable objects from it. In our experiments we set $minPts = 0.1n$, and found a value of ϵ that makes

Paper accepted in Expert Systems with Applications (Elsevier), to appear in 2021. http://doi.org/10.1016/j.eswa.2021.114971

DBSCAN identify the number of clusters we have previously found. This ensures fairness among the algorithms we experiment with, as all identify clusterings with the same number of clusters.

As well as being able to cluster a data set, one must be able to decide whether a given clustering represents the actual structure of the data set without the use of labels. Clustering validity indices (CVIs) are usually used for this purpose. There is no clear evidence in the literature showing a particular CVI to be the best in all cases, however, the average Silhouette width (Rousseeuw, 1987) usually performs well (Arbelaitz et al., 2013).

For a given $x_i \in S_l$, let $a(x_i) = \frac{1}{|S_l|-1} \sum_{x_j \in S_l \setminus \{x_i\}} \sum_{v=1}^{d} (x_{iv} - x_{jv})^2$. That is, $a(x_i)$ is the average distance between x_i and all other objects in its cluster. A low $a(x_i)$ indicates the suitability of the assignment of x_i to S_l. Let $b(x_i) = \min_{S_t \neq S_l} \frac{1}{|S_t|} \sum_{x_j \in S_t} \sum_{v=1}^{d} (x_{iv} - x_{jv})^2$. That is, $b(x_i)$ is the average distance between x_i and the objects of its closest neighbouring cluster. A high $b(x_i)$ indicates the unsuitability of assigning x_i to the closest cluster to S_l. The Silhouette index of x_i is given by

$$s(x_i) = \frac{b(x_i) - a(x_i)}{\max\{a(x_i), b(x_i)\}}. \quad (9)$$

Clearly, $-1 \leq s(x_i) \leq 1$. A $s(x_i)$ close to one indicates x_i is closer to the other objects in its cluster than to objects in other clusters. We can expand this measure to deal with all $x_i \in \mathcal{X}$, that is $\frac{1}{n} \sum_{x_i \in \mathcal{X}} s(x_i)$.

3. Classical data normalisation

Often, data sets contain features with different variances. Features with a higher variance have a higher average distance than features with a lower variance. Hence, the former have a higher contribution to the clustering than the latter. This common issue highlights the importance of data pre-processing. In this paper, we normalise our data sets (for details on the data sets themselves see Section 5) using

$$x'_{iv} = \frac{x_{iv} - \bar{x}_v}{max(x_v) - min(x_v)}, \quad (10)$$

where $\bar{x}_v = \frac{1}{n} \sum_{x_i \in \mathcal{X}} x_{iv}$, the average of feature v over all objects in $\mathcal{X}$. The z-score is also a popular choice in this scenario, it is given by

$$x'_{iv} = \frac{x_{iv} - \bar{x}_v}{\sigma_v},$$

where σ_v is the standard deviation of v over all objects in $\mathcal{X}$. We favoured range normalisation (10) over the z-score because the latter is biased towards features following a unimodal distribution. This is perhaps easier to explain with an example. Let the features v_1 and v_2 be unimodal and multimodal, respectively. The standard deviation of v_2 will be higher than that of v_1, thus, the z-score of v_1 will be higher than that of v_2. Hence, v_1 will have a higher contribution to the clustering than v_2. However, in clustering we would be more interested in the clusters' information in v_2.

Another interesting characteristic of (10) is that if v is a binary feature, then $max(x_v) - min(x_v) = 1$. Hence, the normalised value x'_{iv} is just $x_{iv} - \bar{x}_v$. Note that $\bar{x}_v$ is in fact the frequency of v in the data set $\mathcal{X}$. The higher the frequency of v the lower the normalised value x'_{iv}, and the lower is its contribution to the clustering. This is well-aligned with intuition, a feature that is commonly present (i.e. frequent) is less likely to be discriminative.

4. Iterative cluster-dependent feature rescaling

The normalisation discussed in Section 3 sets all features of a given data set to have about the same contribution to the clustering. This can also be seen as a disadvantage because it means that features with a higher relevance are set to have the same contribution to the clustering as features with a lower relevance. Intuition indicates that features with a higher relevance to the clustering should have a higher contribution. In fact, we can go even further. A given feature v may have different degrees of relevance at each cluster $S_k \in S$, and this should be taken into account during the clustering task.

We can interpret w_{lv} in the distance measure used in *imwk*-means (3) as the degree of relevance of feature v at cluster S_l. Such assertion requires further analysis of *imwk*-means. This algorithms aims at producing a weight w_{lv} for $l = 1, 2, ..., k$ and $v = 1, 2, ..., d$, minimising (4) subject to the conditions in (5). Notice that the dispersion of v at cluster S_l is given by

$$D_{lv} = \sum_{x_i \in S_l} |x_{iv} - z_{lv}|^p,$$

allowing us to re-write (4) as

$$P(U, Z, W) = \sum_{v=1}^{d} \sum_{l=1}^{k} w_{lv}^p D_{lv}.$$

The Lagrangian function of the above is

$$\mathcal{L}(W, \lambda) = \sum_{v=1}^{d} w_{lv}^p D_{lv} + \lambda \left(1 - \sum_{v=1}^{d} w_{lv}\right).$$

Paper accepted in Expert Systems with Applications (Elsevier), to appear in 2021. http://doi.org/10.1016/j.eswa.2021.114971

Allowing us to equate its two partial derivatives to zero.

$$\frac{\partial \mathcal{L}}{\partial w_{lv}} = p w_{lv}^{p-1} D_{lv} - \lambda = 0, \tag{11}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = 1 - \sum_{v=1}^{d} w_{lv} = 0. \tag{12}$$

We can re-arrange (11) to

$$w_{lv} = \left(\frac{\lambda}{p D_{lv}} \right)^{\frac{1}{p-1}}, \tag{13}$$

and substitute (13) into (12)

$$\sum_{v=1}^{d} \left(\frac{\lambda}{p D_{lv}} \right)^{\frac{1}{p-1}} = 1.$$

The above leads to

$$(\lambda)^{\frac{1}{p-1}} = \frac{1}{\sum_{v=1}^{d} \left(\frac{1}{p D_{lv}} \right)^{\frac{1}{p-1}}},$$

and

$$w_{lv} = \left(\sum_{u=1}^{d} \left[\frac{D_{lv}}{D_{lu}} \right]^{1/(p-1)} \right)^{-1}.$$

The weights calculated as per the above Equation minimise (4) by modelling the within-cluster degree of relevance of each feature. This is quite interesting because it allows us to go a step beyond the normalisation described in Section (3) by using these weights as feature rescaling factors. This is quite unusual because each feature $v = 1, 2, ..., d$ will have k factors, but it is fine because these are the weights minimising the clustering criteria (4).

Given a set of weights calculated as per (6), we can re-scale a data set $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ that has been normalised (see Section 3) using

$$x'_{iv} = w_{lv} x_{iv} \text{ for } i = 1, 2, ..., n, \text{ and } v = 1, 2, ..., d, \tag{14}$$

where $x_i \in S_l$. In other words, the rescaling factor applied to x_{iv} depends on both: (i) feature v; (ii) the cluster x_i belongs to.

The method we introduce in this paper also has another novelty. In the first step of *imwk*-means there is no data transformation that separates clusters, and each w_{lv} is set to d^{-1}. While this seems sensible as a starting point to minimise (4), it also means this starting point is suboptimal. To address this, the main part of our method iterates between generating clusterings with *imwk*-means and rescaling the data set using (14). This way, at each iteration *imwk*-means starts from a better position. Given that $\sum_{v=1}^{d} w_{lv} = 1$ for $l = 1, 2, ..., k$, each time the data set is rescaled the values of its entries are lowered. To avoid computational issues related to dealing with very low numbers, we also normalise the data set using (10) between each of these main iterations.

Iterative cluster-dependent feature rescaling (icdfr):

1. For each value of p from 1.1 to 5.0 in steps of 0.1, generate a clustering and a set of weights using *imwk*-means.
2. Calculate the similarity between each pair of clusterings. This similarity can be calculated using the Adjusted Rand Index. Select as optimal p that which is associated to the clustering with the highest average similarity to all other clusterings.
3. Rescale the data set using the weights generated with the optimal p and (14).
4. Normalise the data set using (10).
5. Apply *imwk*-means to the new data set with the optimal p. This will update the weights as well as the clustering. Unless a pre-determined number of iterations has been reached (or the algorithm has converged), go to Step 3.

In the above, steps one and two relate to a consensus approach that has been shown to find suitable values for the Minkowski exponent p (de Amorim et al., 2017). Regarding the number of iterations in Step five, we experimented with 100 although the algorithm would converge much before that.

5. Malware data

In this paper we analyse drive-by download malware. In other words, malicious code downloaded unintentionally to the user's computer. In order to gather useful data one needs to release a malware sample in a safe environment, analyse the malware itself and keep track of any changes it does to such environment. Luckily, there are a number of options in terms of software we could use to accomplish this. This type of software is commonly referred as malware sandbox, and it is used to execute untrusted programs without risking the host machine (for details see Gandotra et al. (2014); Chakkaravarthy et al. (2019), and references therein). In this paper we analyse two data sets collected using the Cuckoo Sandbox (Cuckoo). However, one should note that the significance of the software choice is rather low as our method does not depend on the sandbox in use (see Section 4). All we need is data describing the malware to

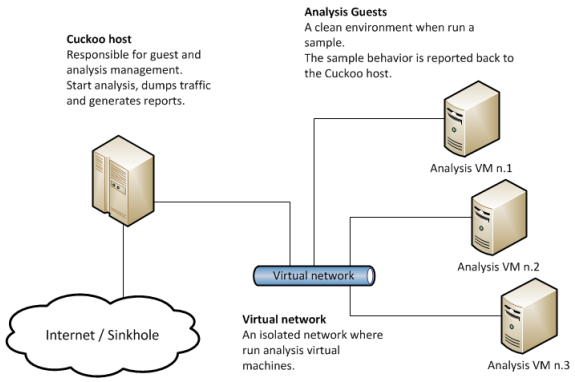


Figure 1: Cuckoo's main architecture. The host runs the management component while the guests run isolated Windows 8 environments. Each environment safely executes a malware, analyses the results, and then gets re-setted to its original (clean) state. Figure from (Cuckoo).

be analysed. Hence, one can use any malware sandbox capable of fulfilling this requirement.

5.1. Data set 1

Cuckoo Sandbox 2.06 (Cuckoo) is a free open-source solution, which has been consistently used in research (see for instance Barakat et al. (2014); Vasilescu et al. (2014); Shijo and Salim (2015)). Cuckoo runs at host-level and manages one or more Windows 8 Virtual Machine (VM) guests (see Figure 1 for a visual representation). The latter is an isolated environment allowing Cuckoo to gather behavioural data (e.g. API calls made by the malware, dropped files, processes spawned, etc.). Cuckoo resets this VM to its original (i.e. clean) state before each experiment with a potential malware. This particular sandbox is also able to extract information from files as part of its static analysis. For each malware Cuckoo lists a number of features related to the behaviour of the malware, as well as its static analysis.

In terms of raw data (i.e. the malware samples themselves), we acquired a total of 2,000 samples from VirusSign (VirusSign). These malware samples were gathered by VirusSign using HoneyPots, submissions, as well as trading and exchange. Each and everyone of them was confirmed by VirusSign to be malicious, using several mainstream AntiVirus software. Each malware also presented features from Cuckoo's behavioural and statical analyses.

Given a list of features obtained with Cuckoo (see Table 1), we can transform our data into an actual data matrix. The process is quite straightforward. First, we must note we have two types of features: (i) binary features, which represent the presence or absence of a particular feature at a particular malware (e.g. whether or not a malware checks if the cursor is in use); (ii) numerical features, which represent the number of times a particular feature was present at a particular malware (e.g. the number of times a malware sent ICMP messages). We have 2,000 samples with 67 features meaning that any given malware sample x_i in our data set $\mathcal{X}$ is described over 67 features (i.e. $n = 2,000$ and $d = 67$).

5.2. Data set 2

We have acquired a second data set, which is freely downloadable from the popular IEEE DataPort (Oliveira, 2019), also originally gathered with Cuckoo. This is a much larger data set containing $47,580$ malware samples described over $1,000$ binary features. These features are relate to the static analysis of each malware, representing the top-1000 imported functions extracted from the 'pe_imports' element of Cuckoo Sandbox reports. The malware samples themselves were downloaded from VirusShare (VirusShare).

This data set has two characteristics that deserve some attention. First, each feature is binary. Hence, standardising this data set using Equation (10) is particularly beneficial. Given a binary feature v, we have that $\max(v) - \min(v) = 1$, and $\bar{x}_v = \frac{1}{n}\sum_{i=1}^{n} x_{iv}$ is the frequency of v over the whole data set $\mathcal{X}$. Hence, the standardised value of a frequent feature will be less than that of a less frequent feature. This is clearly aligned with the intuitive idea that less frequent features tend to be more discerning. Second, this is a high-dimensional data set. Hence, calculating the Minkowski centre many times over such data set may become a slow process. For the experiments with this particular data set we decided to substitute the Minkowski centre calculations by the average. Given the latter can be expressed in closed-form, it is considerably faster. Our results show that by using this approximation we are still able to improve cluster recovery.

6. Clustering results

In this section we analyse the results obtained on each of the data sets we experiment with.

6.1. Data set 1

We began by applying classical normalisation (see Section 3) to this data set (for details on the data set, see Section 5.1). Figure 2a shows the plot of our data over its first and second principal components. Unfortunately, there is no clear evidence of a cluster structure from a Gaussian perspective (i.e. no clearly separable round clusters).

Paper accepted in Expert Systems with Applications (Elsevier), to appear in 2021. http://doi.org/10.1016/j.eswa.2021.114971

Table 1: The list of features identified by Cuckoo in the malware samples we obtained from VirusSign. In the features description a '#' means 'number of', implying the feature is numerical, all other features are binary. 'PE' means Portable Executable.

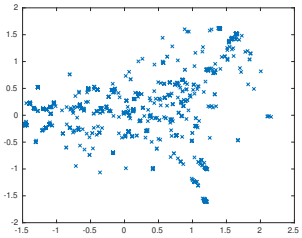
Name	Description	Name	Description
ICMP	# ICMP messages	AllocateVMem	# calls to NtAllocateVirtualMemory
AntiDebug	Use of debugging techniques	Bind	# calls to bind
CheckCursor	Whether a cursor is in use	CloseSocket	# calls to closesocket
CreateFile	# calls to NtCreateFile	CreateMutant	# calls to NtCreateMutant
CryptographyReg	Access to Cryptography registry	CustomLocaleReg	Access to CustomLocale registry
DelayExe	Call to NtDelayExecution	DeviceIO	Use of DeviceIOControl
DroppedFiles	# files dropped	FindFile	# calls used to locate files
FreeVMem	# calls to NtFreeVirtualMemory	GetSysTime	Use of GetSystemTimeAsFileTime
HttpOpenReq	# calls to HttpOpenRequest	HttpSendReq	# calls to HttpSendRequest
IE	Access to IE registry	MapView	# calls to NtMapViewOfSection
OpenFile	# calls to NtOpenFile	OpenMutant	# calls to NtOpenMutant
ProcessNum	# processes spawned	ProtectVMem	# calls to NtProtectVirtualMemory
QueryFile	# queries for information about files	RegCreate	# calls to create registry keys
RegQuery	# queries to the registry	SafeBootReg	Access to SafeBoot registry
Socket	# calls to socket	SortingReg	Access to Nls/Sorting registry
TcpipReg	Access to TCP/IP registry	WriteFile	# calls to NtWriteFile
IReadFile	# calls to InternetReadFile	ToolSS	Use of CreateToolHelp32Snapshot
Dllsloaded	# dlls loaded	SysInfoRef	Access to SystemInformation registry
CryptDecodeObjectX	# calls to CryptDecodeObjectX	Fips	Access to FIPS algorithm policy
CryptCreateHash	# calls to CryptCreateHash	CryptHashData	# calls to CryptHashData
DnscacheReg	Access to DNSCache registry	RegModify	# calls to modify registry keys
DockingInfo	Access to DockingState registry	Persistence	Access to persistence-related registry keys
SCManager	Access to service control manager	CryptExportKey	# calls to CryptExportKey
CryptGenKey	# calls to CryptGenKey	AppInit	Access to DLL-loading registry keys
CryptAcquireContextA	# calls to CryptAcquireContextA	RemoteThread	Creation of remote threads.
Files Recreated	# recreated files	DuplicateProcess	# call to duplicate process
NumSections	# sections in PE file	NumResources	# resources in PE file
NumExports	# exports in PE file	TCP	# tcp packets detected
UDP	# udp packets detected	HTTP	# http packets detected
DroppedBuffers	# dropped buffers	Yara-embedded_ pe	Detects embedded PE file
Yara-LnkHeader	Detects lnk header	DnscacheReg	DnsCache registry modified
Yara-embedded_ win_ api	Detects embedded win api	Yara-shellcode	Detects shellcode in file
Yara-vm_ detect	Use VM detection techniques	CheckDiskSize	Call to CheckDiskSize function
Yara-embedded_ macho	Detects Mach-o file		

As popular as it may be, a clustering algorithm such as k-means++ will identify clusters even if there is no cluster structure in a data set. Hence, one should not jump to conclusions when applying this algorithm. To illustrate this, we applied k-means++ to our data set 100 times. Figure2b shows the clustering we obtained with the lowest criterion output by setting $k = 7$ (given this is just illustrative, the actual value of k is of little consequence). The fact this clustering is meaningless is further reinforced by an average Silhouette of 0.39. The Ward's method and DBSCAN produced the very similar average Silhouettes of 0.38 and 0.37, respectively.

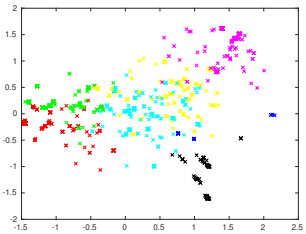
The results we obtained using our icdfr method are certainly more promising. In these experiments we set the *imwk*-means' thereshold $\theta = 3$. We did so to avoid very small clusters in our results, which would not be of particular interest (this choice ended up leading to seven clusters, hence the number of clusters in our illustrative k-means++ example). The first and second steps of our method (described in Section 4) use consensus clustering in order to identify a suitable Minkowski exponent p between 1.1 and 5.0 — higher values of p tend to remove the advantages of feature weighting as (6) will produce more uniform weights. In this experiment the optimal value was found to be $p = 3.9$. Given this value of p we set the number of iterations to 100 and allowed our method to follow Steps 3-5. We were happy to see the method converged in iteration 53 — and even happier to see that the average Silhouette of *imwk*-means on the data set produced by our method was 0.92. The new data set generated by our method also increased the average Silhouette of k-means++ and the Ward's method to around 0.52. Regarding DBSCAN, its average Silhouette increased from 0.37 to 0.47.

Figure 3 shows our method in action. Each of its subfigures (a to f) shows the plot of a *imwk*-means clustering (over its first and second principal components) on a data set generated by our method at a different (but in-

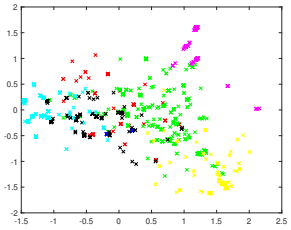
Paper accepted in Expert Systems with Applications (Elsevier), to appear in 2021. http://doi.org/10.1016/j.eswa.2021.114971



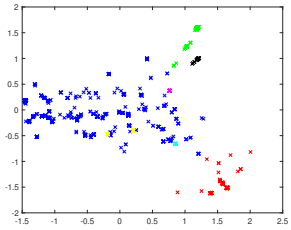
(a) Normalised data set.



(b) The k-means++ clustering of data set 1 with the lowest criterion output over 100 runs, with seven clusters. This clustering has an average Silhouette of 0.39



(c) The Ward's method clustering of data set 1 with seven clusters. This clustering has an average Silhouette of 0.38



(d) The DBSCAN clustering of data set 1 with seven clusters. This clustering has an average Silhouette of 0.37

Figure 2: The normalised data set 1 plotted over its first and second principal components.

creasing) iteration. We can clearly see that our method starts from a chaotic scenario and it quickly starts separating clusters.

All of the above is very positive, but we need to ensure that our final clustering (i.e. Figure 3f) is actually meaningful in the real-world. In order to do this, we analysed each cluster with the help of VirusTotal (VirusTotal). The latter is the most prominent online public service with multiple anti-virus scanners (Sakib et al., 2020). Even with its help, analysing our malware data is far from being a trivial task. VirusTotal is capable of describing each malware in our data set by employing the use of a number of AntiVirus (AV) software. However, this does not mean that each and every AV will agree what a malware sample actually is (or even if the sample is really a malware). Also, different AVs may have different taxonomies. Thus, even if two AVs agree what a malware sample actually is, they may use different names.

Thanks to our analysis we can describe each cluster as follows:

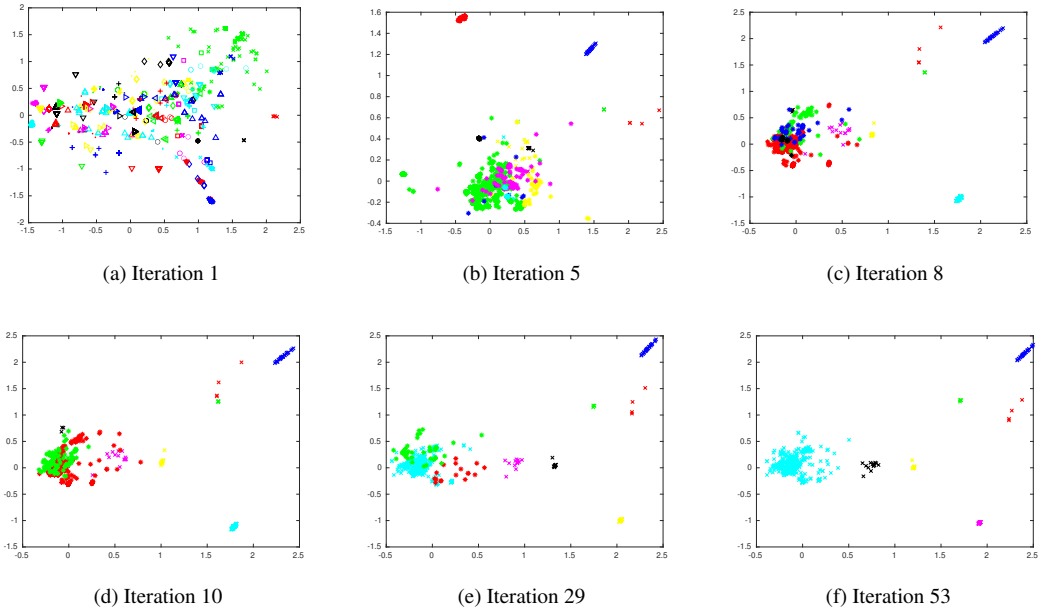
Cluster one: this cluster contains four malware samples, Figure 3f shows these in red. The AV labels for these in VirusTotal seem to be quite different (using names like adware and trojan, which are not mutually exclusive), however, all four samples have exactly the same compilation timestamp. This certainly suggests these malware samples are very much related.

Cluster two: this cluster contains four malware samples, Figure 3f represents these in green. The vast majority of AV's in VirusTotal labels these as trickbot, a trojan designed to steal banking information in particular. All the malware samples in this cluster share the same import hash (imphash), which means they have very similar import tables and are by consequence similar.

Cluster three: this cluster contains 27 malware samples, Figure 3f represents these in blue. These malware samples can be easily characterised by the huge amount of behaviour they exhibit while running in a Virtual Machine, sometimes spawning over 100 processes. According to VirusTotal, over 30 AVs (out of 60) label most of the samples in this cluster as generic malware. Around five AVs consistently label these malware samples with ransomware characteristics, while other five AVs use the term flystudio (adware).

Cluster four: this cluster contains 14 malware samples, Figure 3f represents these in black. Eight of the

Paper accepted in Expert Systems with Applications (Elsevier), to appear in 2021. http://doi.org/10.1016/j.eswa.2021.114971



(a) Iteration 1 (b) Iteration 5 (c) Iteration 8

(d) Iteration 10 (e) Iteration 29 (f) Iteration 53

Figure 3: Clusterings generated by *imwk*-means on the data sets produced by our method, at each iteration. These figures show the separation of clusters until the convergence of our method (iteration 53). In the last iteration the average Silhouette index is of 0.92. This experiment is based on the Cuckoo data set 1.

malware samples appear to be a specific type of trojan (a downloader/installer). All but one of the malware samples has the same compilation timestamp, and share the same imphash.

Cluster five: this cluster contains 14 malware samples, Figure 3f represents these in yellow. These malware samples have different imphashes but the AVs in VirusTotal labels all of these samples as belonging to the ransomware family GandCrab.

Cluster six: this cluster contains 79 malware samples, Figure 3f represents these in magenta. Almost all malware samples in this cluster exhibit extremely similar behaviour given they all share the same imphash value. VirusTotal suggests these as adware in general, and AVs classify them as being in the Adposhel or DNSUnlocker families.

Cluster seven: this cluster contains 1,858 malware samples, Figure 3f represents these in cyan. This is a very large cluster for us to analyse each and every malware sample in VirusTotal. The malware samples in this cluster seem to be associated with different families, but also seem different from the malware samples in other clusters.

The above shows there is a considerable difference in cardinality between clusters, which is certainly expected. Malwares have a tendency to appear in bursts, and their distribution is highly skewed (Song et al., 2016). Our method identifies the most anomalous clusters first. Further analysis could be done by applying our method solely to the data in the largest cluster. We do not pursue this here because we have already clearly achieved our aim. Taking the cluster's descriptions above together with: (i) the average Silhouette of 0.92 given by *imwk*-means; (ii) the increased average Silhouette of *k*-means++, Ward's method, and DBSCAN (see Table 2); (iii) the mathematical model shown in Section 4, we can state our method produces a data set which increases the chances of a meaningful clustering.

6.2. Data set 2

This data set (unlike the previous) does seem to have some evidence of cluster structure. Figure 4 shows our data set 2, after being normalised, plotted over its first and second principal components. We can see that *k*-means++ is able to separate this into four clusters, with an average Silhouette index of 0.6275 (see Figure 4b). The Ward's method produces a similar Silhouette Index of 0.6167, while DBSCAN reached only 0.4055. Such results make this second experiment to be particularly interesting. See, our previous experiment (with data set 1) showed that if all the clustering algorithms we experiment with (i.e. *k*-means++, Ward's method, and

DBSCAN) produced a subpar clustering, our method was able to considerably improve their cluster recovery. This second experiment now shows that even if some of algorithms are able to produce a reasonable clustering (i.e. k-means++ and Ward's method), our method is still able to produce a meaningful improvement.

Our experiments with data set 2 follow the same framework of our experiments with our previous data set. Again, we set the *imwk*-means' thereshold $\theta = 3$ in order to avoid a large number of small clusters in our results. This choice lead to our method identifying four main clusters (this is the reason why we chose $k = 4$ for the k-means++ experiment in Figure 4b). The consensus clustering part of our method (steps one and two, see Section 4) identified $p = 3.3$ to be the most suitable (once again, in a search from 1.1 to 5.0, in steps of 0.1). This time our method produced a clustering with a Silhouette index of 0.7817. The new data set generated by our method also increased the Silhouette index of k-means++ from 0.6275 to 0.7724, Ward's method from 0.6167 to 0.7708, and DBSCAN from 0.4055 to 0.7829.

Figure 5 shows the evolution of our method over the iterations. It began by identifying 32 possible clusters (iteration 1), and then it eventually partitioned the whole data set into four main clusters. We can also see that the main structure of these four clusters was clearly visible in iteration 45 (with a Silhouette index of 0.7778), with a small difference in iterations 85 (with Silhouette index of 0.7814) and iteration 100 (with a Silhouette index of 0.7817). Admittedly, our method did not fully converge but this certainly happened because of our substitution of the Minkowski Centre for the average to increase speed (See Section 5.2).

Table 2 summarises the results of all of our experiments experiments. This clearly shows that our method leads to better clusters, as measured by the popular Silhouette index.

7. Conclusion

In this paper we faced the problem of finding meaningful clusters in drive-by-download malware data. The patterns in this type of data can be difficult to identify, particularly if using a distance based clustering algorithm (see Figures 2a and 2b, as well as the results under normalised data in Table 2). We identified as the main reason for this the fact that classical data normalisation treats all features equally, instead of favouring those that are more relevant.

In order to address the above, we introduced a data pre-processing method called iterative cluster-dependent feature rescaling (for details see Section 4). This method makes use of cluster-dependent feature weights to iteratively separate the clusters in a data set (see Figures 3 and 5). This mathematically sound method leads to higher average Silhouettes. For instance, k-means++ saw an increase from 0.39 to 0.52 in terms of average Silhouette index when using our feature rescaling method, while *imwk*-means went as high as 0.92 — on data set 1. Hence, our method helps to produce more meaningful clusters.

We foresee our method being used in the data preprocessing stage of a malware clustering task, or perhaps even in other clustering tasks. In the future we intend to investigate the use of this method in supervised classification problems.

References

Abboud, A., Cohen-Addad, V., Houdrougé, H., 2019. Subquadratic high-dimensional hierarchical clustering, in: Advances in Neural Information Processing Systems, pp. 11580–11590.

Aggarwal, C.C., Reddy, C.K., 2014. Data clustering. Algorithms and applications. Chapman&Hall/CRC Data mining and Knowledge Discovery series, Londra .

Cordeiro de Amorim, R., Komisarczuk, P., 2012. On partitional clustering of malware, in: The First International Workshop on Cyber Patterns: Unifying Design Patterns with Security, Attack and Forensic Patterns. (Cyberpatterns), pp. 47–51.

de Amorim, R.C., Shestakov, A., Mirkin, B., Makarenkov, V., 2017. The minkowski central partition as a pointer to a suitable distance exponent and consensus partitioning. Pattern Recognition 67, 62–72.

Arbelaitz, O., Gurrutxaga, I., Muguerza, J., PéRez, J.M., Perona, I., 2013. An extensive comparative study of cluster validity indices. Pattern Recognition 46, 243–256.

Arthur, D., Vassilvitskii, S., 2007. k-means++: The advantages of careful seeding, in: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics. pp. 1027–1035.

Asquith, M., 2016. Extremely scalable storage and clustering of malware metadata. Journal of Computer Virology and Hacking Techniques 12, 49–58.

Barak, S., Mokfi, T., 2019. Evaluation and selection of clustering methods using a hybrid group mcdm. Expert Systems with Applications 138, 112817.

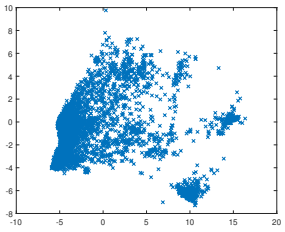
Barakat, O.L., Hashim, S.J., Abdullah, R.S.A.B.R., Ramli, A.R., Hashim, F., Samsudin, K., Ab Rahman, M., 2014. Malware analysis performance enhancement using cloud computing. Journal of Computer Virology and Hacking Techniques 10, 1–10.

BBC, 2020a. Ransomware attack takes us maritime base offline. BBC URL: https://www.bbc.co.uk/news/technology-50972890.

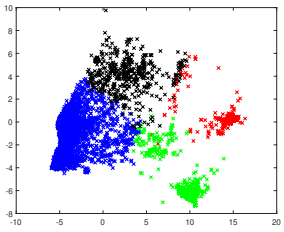
BBC, 2020b. Us-government issued phones run 'chinese malware'. BBC URL: https://www.bbc.co.uk/news/technology-51054901.

Chakkaravarthy, S.S., Sangeetha, D., Vaidehi, V., 2019. A survey on malware analysis and mitigation techniques. Computer Science Review 32, 1–23.

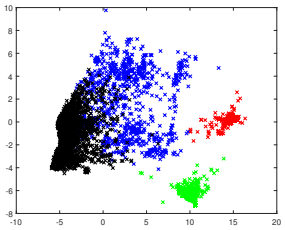
Cuckoo, . Cuckoo Sandbox. https://cuckoosandbox.org/. Accessed: 2019-09-30.



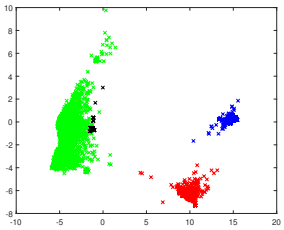
(a) Normalised data set.



(b) The *k*-means++ clustering of data set 2 with the lowest criterion output over 100 runs, with four clusters. This clustering has an average Silhouette of 0.6275.



(c) The Ward's method clustering of data set 2, with four clusters. This clustering has an average Silhouette of 0.6167.



(d) The DBSCAN clustering of data set 2, with four clusters. This clustering set a number of malware samples as "noise", leading to an average Silhouette of 0.4055.

Figure 4: The normalised data set 2 plotted over its first and second principal components.

Table 2: The Silhouette index produced by our experiments with *k*-means and *imwk*-means under usual normalisation, and our icdf method.

	normalised data				icdf data			
	k-means++	Ward	DBSCAN	*imwk*-means	*k*-means++	Ward	DBSCAN	*imwk*-means
Data set 1	0.3909	0.3837	0.3708	0.7117	0.5233	0.5203	0.4735	0.9237
Data set 2	0.6275	0.6167	0.4055	0.0035	0.7724	0.7708	0.7829	0.7817

De Amorim, R.C., Mirkin, B., 2012. Minkowski metric, feature weighting and anomalous cluster initializing in k-means clustering. Pattern Recognition 45, 1061–1075.

Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al., 1996. A density-based algorithm for discovering clusters in large spatial databases with noise., in: Kdd, pp. 226–231.

Faridi, H., Srinivasagopalan, S., Verma, R., 2018. Performance evaluation of features and clustering algorithms for malware, in: 2018 IEEE International Conference on Data Mining Workshops (ICDMW), IEEE. pp. 13–22.

Fox, C., 2017a. Massive ransomware infection hits computers in 99 countries. BBC URL: `https://www.bbc.co.uk/news/technology-39901382`.

Fox, C., 2017b. Nhs cyber-attack: Gps and hospitals hit by ransomware. BBC URL: `https://www.bbc.co.uk/news/health-39899646`.

Fränti, P., Sieranoja, S., 2019. How much can k-means be improved by using better initialization and repeats? Pattern Recognition 93, 95–112.

Galán, S.F., 2019. Comparative evaluation of region query strategies for dbscan clustering. Information Sciences 502, 76–90.

Gandotra, E., Bansal, D., Sofat, S., 2014. Malware analysis and classification: A survey. Journal of Information Security 2014.

Gibert, D., Mateu, C., Planes, J., 2020. Hydra: A multimodal deep learning framework for malware classification. Computers & Security , 101873.

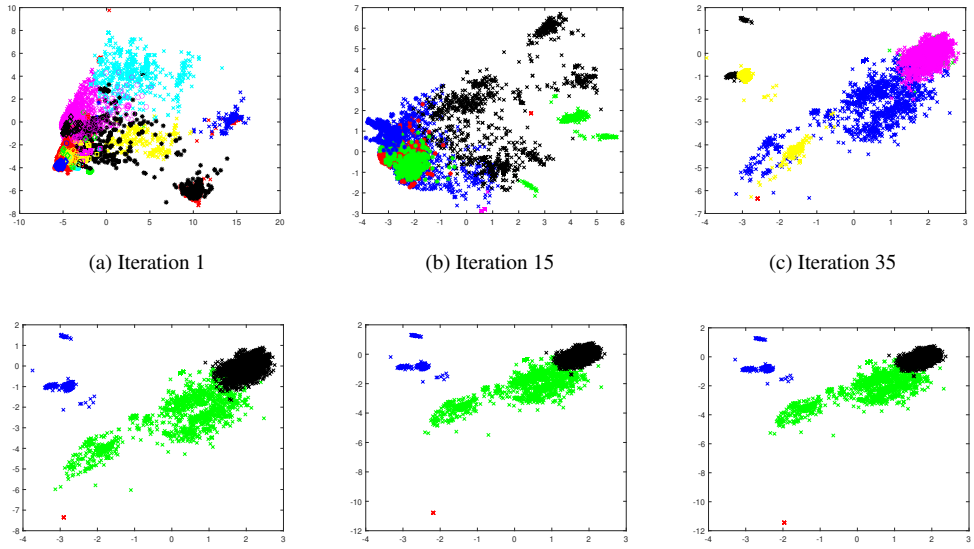
Govender, P., Sivakumar, V., 2020. Application of k-means and hierarchical clustering techniques for analysis of air pollution: A review (1980–2019). Atmospheric Pollution Research 11, 40–56.

Hennig, C., 2015. What are the true clusters? Pattern Recognition Letters 64, 53–62.

Jain, A.K., 2010. Data clustering: 50 years beyond k-means. Pattern recognition letters 31, 651–666.

Komatwar, R., Kokare, M., 2020. A survey on malware detection and classification. Journal of Applied Security Research , 1–31.

Lai, W., Zhou, M., Hu, F., Bian, K., Song, Q., 2019. A new dbscan parameters determination method based on improved mvo. IEEE Access 7, 104085–104095.



(a) Iteration 1 (b) Iteration 15 (c) Iteration 35

(d) Iteration 45 (e) Iteration 85 (f) Iteration 100

Figure 5: Clusterings generated by *imwk*-means on the data sets produced by our method, at each iteration. These figures show our method initially identifying up to 32 possible clusters, and then iteratively separating the whole data into four of these. In the last iteration the average Silhouette is 0.7817. This experiment is based on data set 2.

MacQueen, J., et al., 1967. Some methods for classification and analysis of multivariate observations, in: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, Oakland, CA, USA.. pp. 281–297.

MATLAB, 2019. version 9.7.0 (R2019b). The MathWorks Inc., Natick, Massachusetts.

Mirkin, B., 2012. Clustering: a data recovery approach. CRC Press.

Oliveira, A., 2019. Malware analysis datasets: Top-1000 pe imports. URL: `https://dx.doi.org/10.21227/004e-v304`, doi:`10.21227/004e-v304`.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830.

R Core Team, 2020. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing. Vienna, Austria. URL: `https://www.R-project.org`.

Rand, W.M., 1971. Objective criteria for the evaluation of clustering methods. Journal of the American Statistical association 66, 846–850.

Rousseeuw, P.J., 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of computational and applied mathematics 20, 53–65.

Sakib, M.N., Huang, C.T., Lin, Y.D., 2020. Maximizing accuracy in multi-scanner malware detection systems. Computer Networks 169, 107027.

Schubert, E., Sander, J., Ester, M., Kriegel, H.P., Xu, X., 2017. Dbscan revisited, revisited: why and how you should (still) use dbscan. ACM Transactions on Database Systems (TODS) 42, 1–21.

Shijo, P., Salim, A., 2015. Integrated static and dynamic analysis for malware detection. Procedia Computer Science 46, 804–811.

Song, L., Huang, H., Zhou, W., Wu, W., Zhang, Y., 2016. Learning from big malwares, in: Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems, pp. 1–8.

Steinley, D., 2006. K-means clustering: a half-century synthesis. British Journal of Mathematical and Statistical Psychology 59, 1–34.

Symantec, 2019. Internet Security Threat Report. Technical Report 24. Symantec.

Talpalaru, A., Bhagwat, N., Devenyi, G.A., Lepage, M., Chakravarty, M.M., 2019. Identifying schizophrenia subgroups using clustering and supervised learning. Schizophrenia Research 214, 51–59.

Tidy, J., 2019. How a ransomware attack cost one firm £45m. BBC URL: `https://www.bbc.co.uk/news/business-48661152`.

Vasan, D., Alazab, M., Wassan, S., Naeem, H., Safaei, B., Zheng, Q., 2020. Imcfn: Image-based malware classification using fine-tuned convolutional neural network architecture. Computer Networks 171, 107138.

Vasilescu, M., Gheorghe, L., Tapus, N., 2014. Practical malware analysis based on sandboxing, in: 2014 RoEduNet Conference 13th Edition: Networking in Education and Research Joint Event RENAM 8th Conference, IEEE. pp. 1–6.

VirusShare, . Virusshare. `https://www.virusshare.com/`. Accessed: 2019-12-01.

VirusSign, . Virussign. `https://www.virussign.com/`. Accessed: 2019-10-02.

VirusTotal, . Virustotal. `https://www.virustotal.com/`. Accessed: 2020-02-12.

Ward Jr, J.H., 1963. Hierarchical grouping to optimize an objective function. Journal of the American statistical association 58, 236–244.

Xiao, G., Li, J., Chen, Y., Li, K., 2020. Malfcs: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks. Journal of Parallel and Distributed Computing .

Yuan, B., Wang, J., Liu, D., Guo, W., Wu, P., Bao, X., 2020. Byte-level malware classification based on markov images and deep

learning. Computers & Security 92, 101740.