ISR project
Threat intelligence – proposal #2

Emese PADANYI

Ede KOMJATI

Date: 2019.12.11.

1. Data preparation

To be able to make a good cluster we should prepare our data for it, choose features, transform them. We have just a few options to choose from. Every column: VirusTotal, file, path, rabin2, strings, yara. From these columns we decided to not choose:

- VirusTotal – Because it already made analysis from the malwares so it wouldn't make sense to reuse it.
- File – Because it is just a dictionary contained hashes from the sample. Every sample has different has so it doesn't make sense to use it for the unsupervised learning.
- Path – Because it is just a file path where the sample came from. It wouldn't help us to decide which sample goes where.

We considered rabin2, but in the start we haven't stick with it.

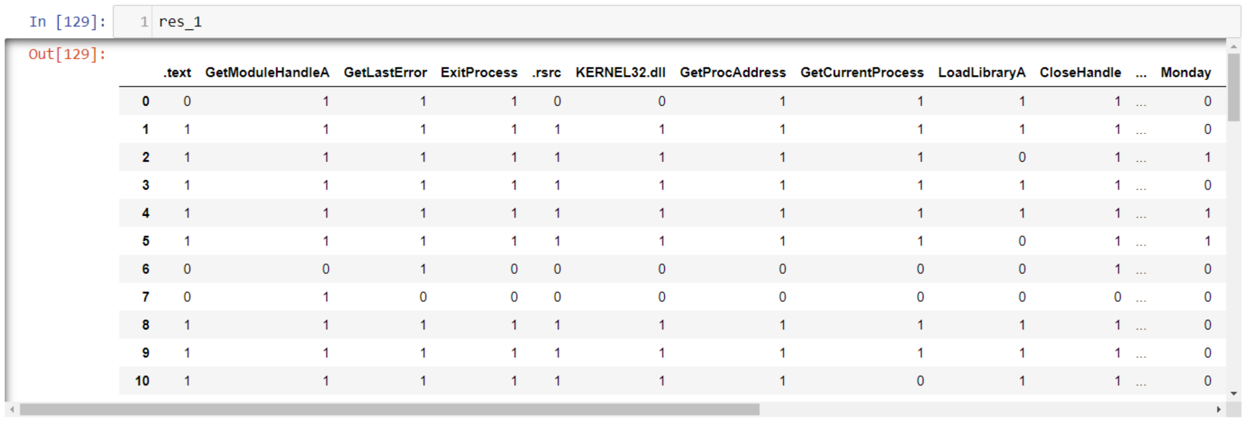
For the start, we've chosen the **Strings** column to create features from it. But to be able to use those strings we should make them understandable to the clustering algorithms.

First, we are cleaning up the strings a little bit. We implemented a function which gets rid of too short or too long strings from the string's lists, the user can change these parameters to fine tune the algorithms. Also, we are getting rid of those strings which aren't represented in too many samples, so if a string's count, in the samples, is lower than a value then we are getting rid of that string.

After this procedure the remaining strings will represent the **relevant_strings_list**. From this list we are creating columns for each string named by the string value. If the sample contains that string, then we set the relevant string column to 1 if it isn't contains it then 0.

Why are we doing the string list separation to columns? We were thinking a lot about it. First we just wanted to encode every string with a sequence of numbers, but this means implicitly with the kmeans for example that there are strings which are more far from each other than others. This isn't our case because we want to map string patterns to malware families. Also we were thinking about onehot encoded strings because they got rid of the above mentioned problem. But the onehot raise another problem: memory inefficient usage. Explained with an example: we have 120 sample, we have ~220.000 unique string. We are filtering (decreasing) them to 7000 unique string but still. If we onehot them that means for one word needs 7000 column. One sample has a ~300-1000 unique general set of strings, that means we need at least 300 words for each sample. 300 words * 7000 column (each word represented by 7000 column because of the onehot). We don't have the infrastructure to optimize this with kmeans, etc…Also it is a bad approach because the count to strings can be different for each sample. That is why we chose the string columns.

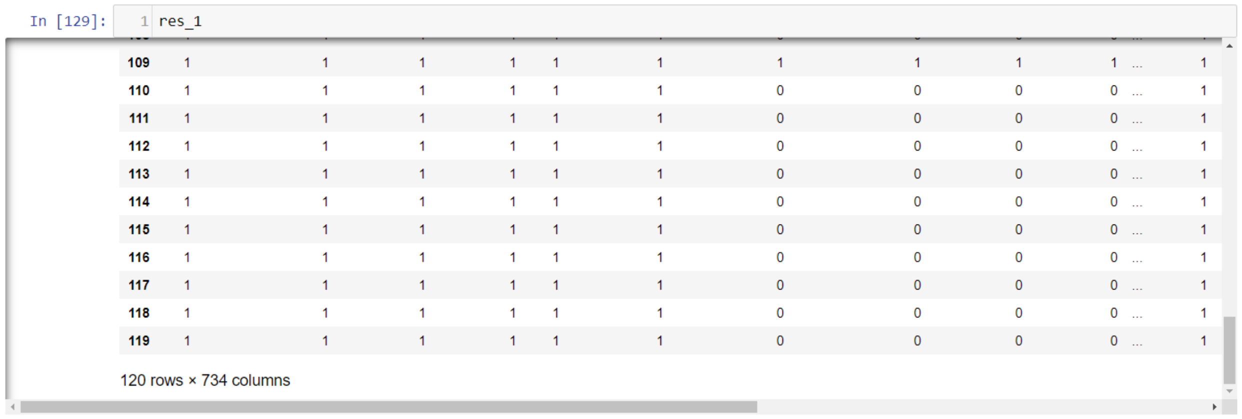
This looks like this:



```
In [129]: res_1
```

Out[129]:

	.text	GetModuleHandleA	GetLastError	ExitProcess	.rsrc	KERNEL32.dll	GetProcAddress	GetCurrentProcess	LoadLibraryA	CloseHandle	...	Monday
0	0	1	1	1	0	0	1	1	1	1	...	0
1	1	1	1	1	1	1	1	1	1	1	...	0
2	1	1	1	1	1	1	1	1	0	1	...	1
3	1	1	1	1	1	1	1	1	1	1	...	0
4	1	1	1	1	1	1	1	1	1	1	...	1
5	1	1	1	1	1	1	1	1	0	1	...	1
6	0	0	1	0	0	0	0	0	0	1	...	0
7	0	1	0	0	0	0	0	0	0	0	...	0
8	1	1	1	1	1	1	1	1	1	1	...	0
9	1	1	1	1	1	1	1	1	1	1	...	0
10	1	1	1	1	1	1	1	0	1	1	...	0



```
In [129]: res_1
```

109	1	1	1	1	1	1	1	1	1	1	...	1
110	1	1	1	1	1	1	0	0	0	0	...	1
111	1	1	1	1	1	1	0	0	0	0	...	1
112	1	1	1	1	1	1	0	0	0	0	...	1
113	1	1	1	1	1	1	0	0	0	0	...	1
114	1	1	1	1	1	1	0	0	0	0	...	1
115	1	1	1	1	1	1	0	0	0	0	...	1
116	1	1	1	1	1	1	0	0	0	0	...	1
117	1	1	1	1	1	1	0	0	0	0	...	1
118	1	1	1	1	1	1	0	0	0	0	...	1
119	1	1	1	1	1	1	0	0	0	0	...	1

120 rows × 734 columns

Here as above mentioned earlier the columns are came from the **relevant_strings_list.** If the sample (row) contains it then the value in the relevant column will be 1 if not then 0.

2. Clustering data

For the first tests we used k-means++ algorithm from scikit learn with different n_cluster. After a few sample we tried out with the dbscan and the spectral clustering too to see what is the difference between the clustering algorithms. In the end we also tried out what we get with the Agglomerative Clustering and with the Mean Shift.

K-means principles

It follows a simple procedure of classifying a given data set into a number of clusters, defined by the letter "k," which is fixed beforehand. The clusters are then positioned as points and all observations or data points are associated with the nearest cluster, computed, adjusted and then the process starts over using the new adjustments until a desired result is reached.

The algorithm:

1. K points are placed into the object data space representing the initial group of centroids.
2. Each object or data point is assigned into the closest k.
3. After all objects are assigned, the positions of the k centroids are recalculated.
4. Steps 2 and 3 are repeated until the positions of the centroids no longer move.

DBSCAN principles

Consider a set of points in some space to be clustered. Let ε be a parameter specifying the radius of a neighborhood with respect to some point. For the purpose of DBSCAN clustering, the points are classified as core points, (density-)reachable points and outliers, as follows:

1. A point p is a core point if at least minPts points are within distance ε of it (including p).
2. A point q is directly reachable from p if point q is within distance ε from core point p. Points are only said to be directly reachable from core points.
3. A point q is reachable from p if there is a path p1, ..., pn with p1 = p and pn = q, where each pi+1 is directly reachable from pi. Note that this implies that all points on the path must be core points, except for q.
4. All points not reachable from any other point are outliers or noise points.

Now if p is a core point, then it forms a cluster together with all points (core or non-core) that are reachable from it. Each cluster contains at least one core point; non-core points can be part of a cluster, but they form its "edge", since they cannot be used to reach more points.

Reachability is not a symmetric relation since, by definition, no point may be reachable from a non-core point, regardless of distance (so a non-core point may be reachable, but nothing can be reached from it). Therefore, a further notion of connectedness is needed to formally define the extent of the clusters found by DBSCAN. Two points p and q are density-connected if there is a point of such that both p and q are reachable from o. Density-connectedness is symmetric.

Spectral clustering principles

Spectral clustering is a technique with roots in graph theory, where the approach is used to identify communities of nodes in a graph based on the edges connecting them. The method is flexible and allows us to cluster non graph data as well.

Spectral clustering uses information from the eigenvalues (spectrum) of special matrices built from the graph or the data set. We'll learn how to construct these matrices, interpret their spectrum, and use the eigenvectors to assign our data to clusters.

Agglomerative Clustering

This is a subset of the hierarchical clustering. This is a "bottom-up" approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.

Mean shift

Mean shift clustering using a flat kernel.

Mean shift clustering aims to discover "blobs" in a smooth density of samples. It is a centroid-based algorithm, which works by updating candidates for centroids to be the mean of the points within a given region. These candidates are then filtered in a post-processing stage to eliminate near-duplicates to form the final set of centroids.

Seeding is performed using a binning technique for scalability.

3. Validate results

We did a lot of clustering with different parameters. But to know which one is good we had to use some metrics and visualization.

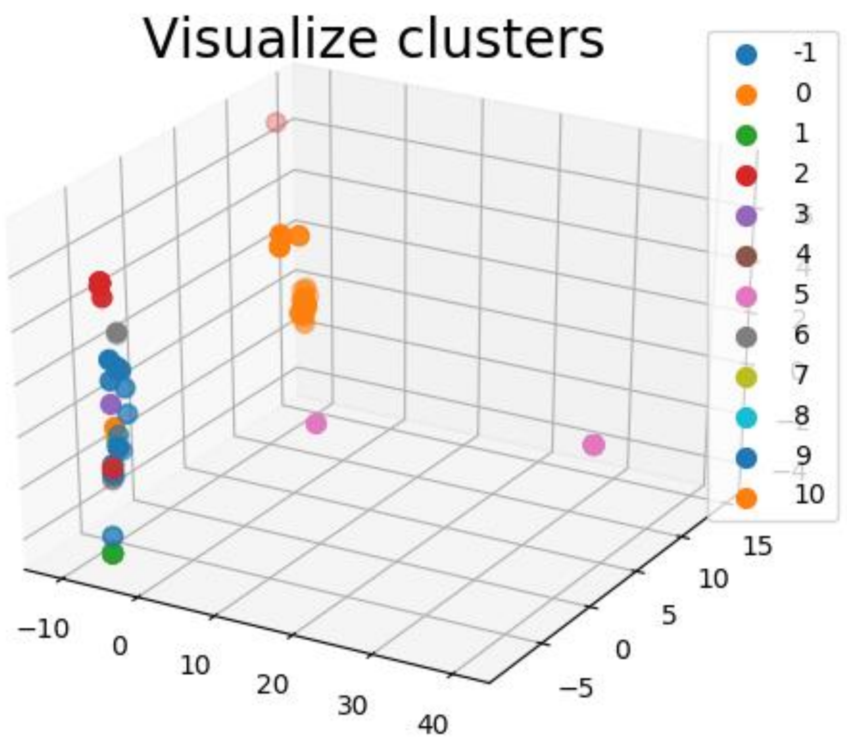
That is when we came up with the idea of Silhouette score and with the tagged samples. We needed evidence what we are doing is good or not. So we created a set of Main yara rules for the malware families and tag each sample with it. After doing this we were able to check the created clusters which malware families are mapped to it. Unfortunately we couldn't rely on the Silhouette all the time because with very high score (close to 0.9) we could't got the same accuracy with the clusters and the malware families.

The created main yara rules:

malware_families = { 'Empty': -1, 'Shohdi':0, 'Bublik':1, 'Mira':2, 'spyeye':3, 'Njrat':4, 'Shifu':5, 'sakula_v1_3':6, 'Wabot':7, 'Warp':8, 'Nanocore_RAT_Gen_2':9, 'Wimmie':10, 'xRAT':11}

Empty has the meaning that it might be any malware included the others.

The real dataset plotted in 3d with the help of PCA looks like this:



The best results are these after a lot of trying:

First the these results came by these data preparation parameters:

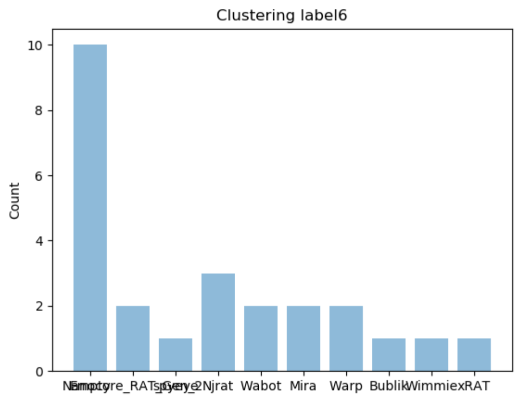
Minimum length of string = 2

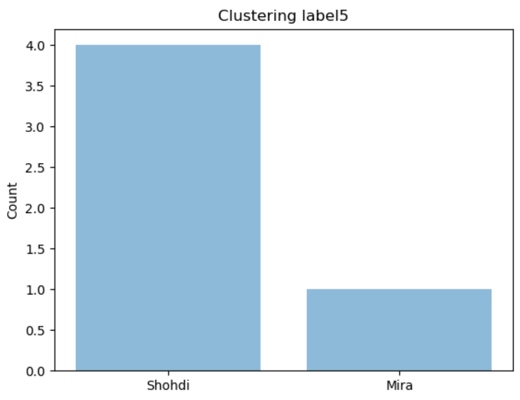
Maximum length of string = 22

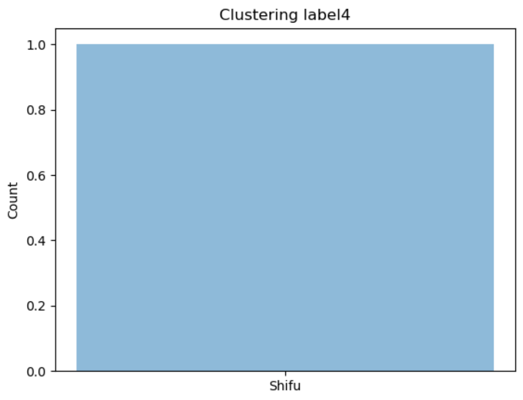
Count of sample which contains it = 17

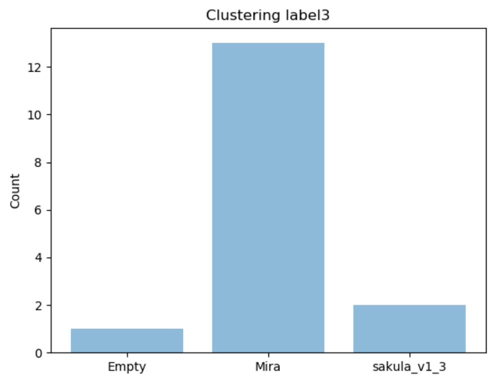
Kmeans, 1 cluster result on the samples:

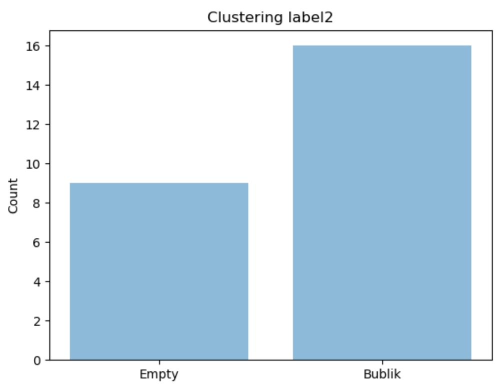
Here we've got 7 cluster. In the label 6 there are 10 empty sample and all of the other type of malwares. Unfortunately we couldn't separated them. I think this is because the lack of samples for that type of malware. As you can see label 4 and 1 manage to cluster the Shifu family. Because the clustering managed to cluster them without any other type of malware we can derive that this malware type uses some sort of string in the malware which easily guess the malware family. This is almost the case with the Shohdi malware family. It can be seen in label5 and 0. It is almost because one mira malware appeared too but this clusters are also accurate clusters. Mira family is also in almost one cluster: label 3. There are some malware (3 of them) which doesn't belong there but this isn't too much. There were some trouble with label 2 because there are some empty files where Bubik and Empty contains the same strings almost so the Bubik family isn't properly separated but contains all of the Bubik malware so it is a larger set. In label 6 there are the other types because their prevalence rate were low only a few sample labeled with these families so we didn't have enough sample to categorize them properly.

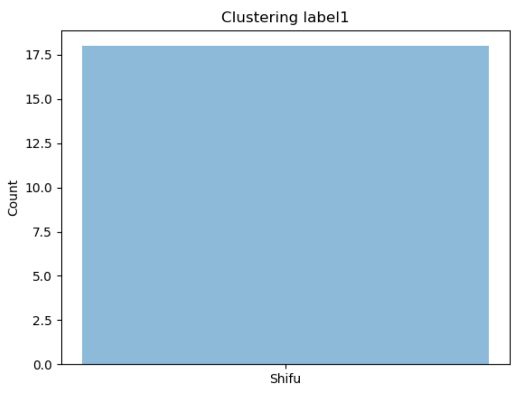


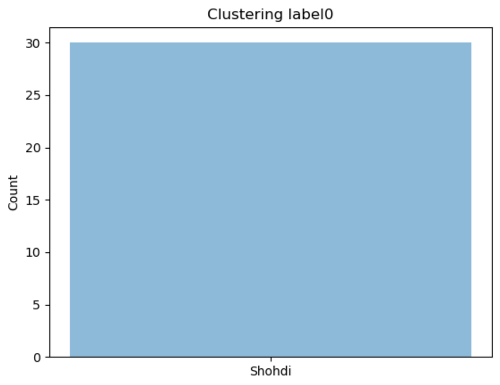


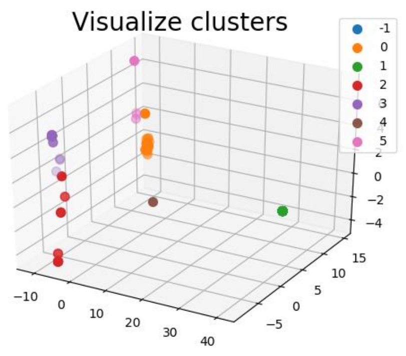








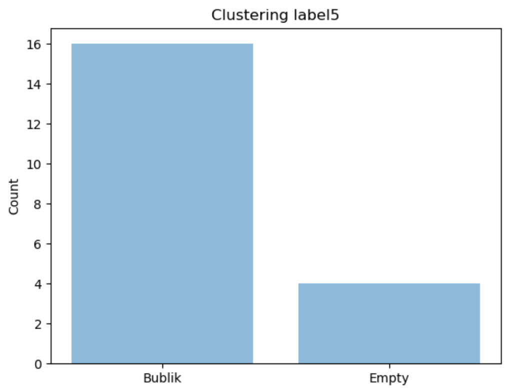


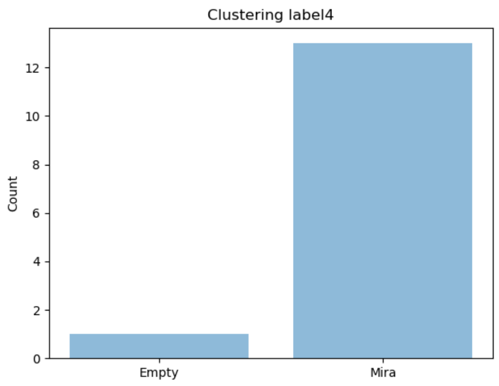


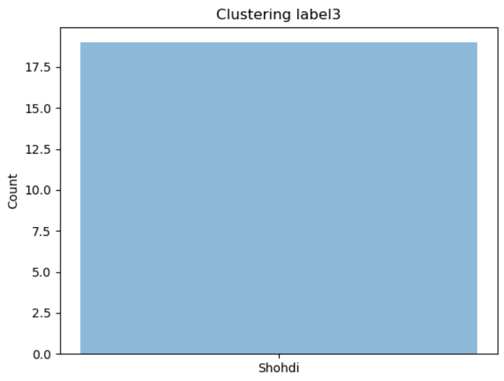
The last picture contains the 120 sample in 3d plotted, reduced the dimensions by the PCA algorithm.

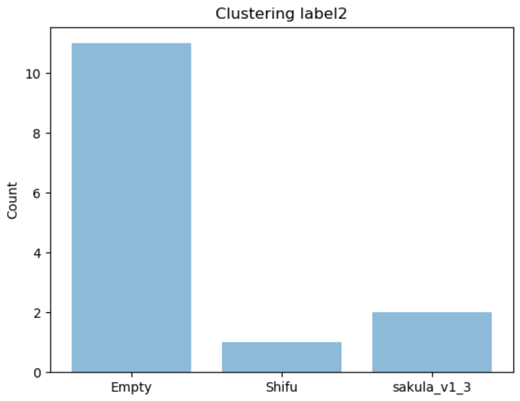
The 2nd good result gave by the Spectral Clustering algorithm:

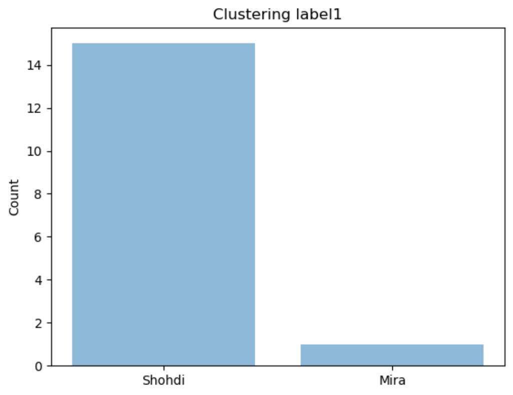
Here we used only 6 cluster, the string list were prepared with the same params as above. Here the Shifu malw. family is with the other tiny families (label 0). Shohdi (label 1, 4), Mira (label 4,0,1), Bublik (label 5), Empty (which contains every other family, label 5,4,2). The bar charts can be seen below.

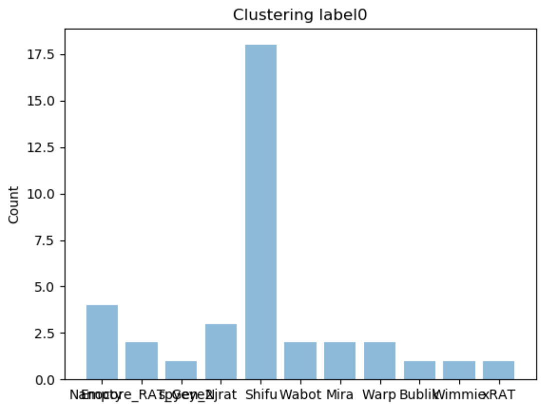

Clustering label5
Count
0
2
4
6
8
10
12
14
16
Bublik
Empty

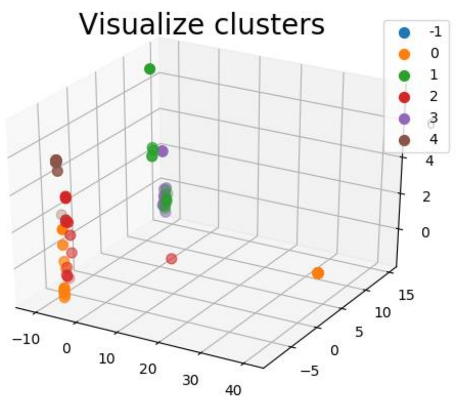

Clustering label4
Count
0
2
4
6
8
10
12
Empty
Mira


Clustering label3
Count
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Shohdi


Clustering label2
Count
0
2
4
6
8
10
Empty
Shifu
sakula_v1_3







How we tried out with a lot of different parameters? We implemented a function which goes through all parameters what we set for ex.: shortlengths = range(2,5) , longlengths = range(12,23, 2), etc… It goes through in an embedded for loop tries out every parameter creates clusters and saves the results. It can be seen in the jupyter notebook file not in the python files. We used jupyter notebook because we felt it faster to try out things in Python.

Before all of this we also did a lot of research, unfortunately there aren't a lot of site which talks about Threat Intelligence, instead there were a lot of paper about it. We considered the Levenshtein distance method to calculate string distance with the help of it but we didn't think it would be paying off the effort. In summary we learnt a lot about different clustering algorithms, but we learnt more about data preparation, because that part really decides how accurate the results going to be. We also learnt about which malware families are more detectable from their strings because of our work.

Our source code and documentation can be found in https://github.com/caspien6/isr_project .