

Detecting Ransomware Addresses on the Bitcoin Blockchain using Random Forest and Self Organizing Maps

HarvardX PH125.9x Final Capstone CYO Project

Kaylee Robert Tejeda

10/31/2021

Abstract

Ransomware is a persistent and growing threat in the world of cybersecurity. A specific area of focus involves detecting and tracking payments made to ransomware operators. While many attempts towards this goal have not made use of sophisticated machine learning methods, even those that have often result in models with poor precision or other performance issues. A two-step method is developed to address the issue of false positives and improve on previous results.

Contents

Introduction	3
Data	4
Goal	4
Outline of Steps Taken	4
Data Analysis	5
Hardware Specification	5
Data Preparation	5
Exploration and Visualization (graphic rework from here to Modeling Approach)	5
Insights Gained from Exploration (graphic rework ends here)	7
Modeling approach	7
Method Part 0: Binary SOMS as a first attempt to isolate ransomware addresses.	8
Method Part 1: Binary Random Forest to isolate ransomware addresses before categorization.	8
Method Part 2: Categorical SOMs to categorize predicted ransomware addresses.	9
Clustering Visualizations: Heatmaps and K-means clustering	10
Results & Performance	17
Results	17
Performance	17
Summary	17
Comparison to results from original paper	17
Limitations	17
Future Work	18
Conclusion	18
References	18
Appendix:	19
Categorical SOM ransomware family prediction table and confusion matrix	19

Introduction

Ransomware attacks have gained the attention of security professionals, law enforcement, and financial regulatory officials.^[1] The pseudo-anonymous Bitcoin network provides a convenient method for ransomware attackers to accept payments without revealing their identity or location. The victims (usually hospitals or other large organizations) come to find that much if not all of their important organizational data have been encrypted with a secret key by an unknown attacker. They are instructed to make a payment to a specific Bitcoin address before a certain deadline to have their data decrypted before being deleted permanently.

The legal and financial implications of ransomware attacks are not of concern for the purpose of this analysis. Many parties are interested in tracking illicit activity (such as ransomware payments) around the Bitcoin blockchain as soon as possible to minimize financial losses. Daniel Goldsmith explains some of the reasons and methods of blockchain analysis at Chainalysis.com.^[2] A ransomware attack could be perpetrated on an illegal darknet market site, for example. The news of such an attack might not be published at all, let alone in popular media. By analyzing the transaction record with a blockchain explorer such as BTC.com, suspicious activity could be flagged in real time given a sufficiently robust model. It may, in fact, be the first public notice of such an event. Any suspicious addresses could then be blacklisted or banned from using other services.

Lists of known ransomware payment addresses have been compiled and analyzed using various methods. One well known paper entitled “BitcoinHeist: Topological Data Analysis for Ransomware Detection on the Bitcoin Blockchain”^[3] will be the source of our data set and the baseline to which we will compare our results. In that paper, Akcora, et al. use Topological Data Analysis (TDA) to classify addresses on the Bitcoin blockchain into one of 28 known ransomware address groups. Addresses with no known ransomware associations are classified as “white”. The blockchain is then considered as a heterogeneous Directed Acyclic Graph (DAG) with two types of nodes describing *addresses* and *transactions*. Edges are formed between the nodes when a transaction can be associated with a particular address.

Addresses on the Bitcoin network may appear many times, with different inputs and outputs each time. The Bitcoin network data has been divided into 24-hour time intervals with the UTC-6 timezone as a reference. Speed is defined as the number of blocks the coin appears in during a 24-hour period and provides information on how quickly a coin moves through the network. Speed can be an indicator of money laundering or coin mixing, as normal payments only involve a limited number of addresses in a 24 hour period, and thus have lower speeds when compared to “mixed” coins. The temporal data can also help distinguish transactions by geolocation, as criminal transactions tend to cluster in time.

With the graph defined as such, the following six numerical features^[2] are associated with a given address:

- 1) *Income* - the total amount of coins sent to an address
- 2) *Neighbors* - the number of transactions that have this address as one of its output addresses
- 3) *Weight* - the sum of fraction of coins that reach this address from address that do not have any other inputs within the 24-hour window, which are referred to as “starter transactions”
- 4) *Length* - the number of non-starter transactions on its longest chain, where a chain is defined as an acyclic directed path originating from any starter transaction and ending at the address in question
- 5) *Count* - The number of starter addresses connected to this address through a chain
- 6) *Looped* - The number of starter addresses connected to this address by more than one path

These variables are defined rather abstractly, viewing the blockchain as a topological graph with nodes and edges. The rationale behind this approach is to quantify specific transaction patterns. Akcora^[3] gives a thorough explanation in the original paper of how and why these features were chosen. We shall treat the features as general numerical variables and will not seek to justify their definitions. Several machine learning methods will be applied to the original data set from the paper by Akcora^[3], and the results will be compared.

Data

This data set was discovered while exploring the UCI Machine Learning Repository^[4] as suggested in the project instructions. The author of this report, interested in Bitcoin and other cryptocurrencies since (unsuccessfully) mining them on an ASUS netbook in rural Peru in late 2010, used *cryptocurrency* as a preliminary search term. This brought up a single data set entitled “BitcoinHeist: Ransomware Address Data Set”. The data set was downloaded and the exploration began.

A summary of the data set tells the range of values and size of the sample.

address	year	day	length	weight	count	looped	neighbors	income	label
Length:2916697	Min. :2011	Min. : 1.0	Min. : 0.00	Min. : 0.0000	Min. : 1.0	Min. : 0.0	Min. : 1.000	Min. :3.000e+07	Length:2916697
Class	1st Qu.:2013	1st Qu.: 92.0	1st Qu.: 2.00	1st Qu.: 0.0215	1st Qu.: 1.0	1st Qu.: 0.0	1st Qu.: 1.000	1st Qu.:7.429e+07	Class
:character	Median	Median :181.0	Median : 8.00	Median : 0.2500	Median : 1.0	Median : 0.0	Median : 2.000	Median :2.000e+08	:character
Mode	:2014	Mean	Mean :181.5	Mean : 45.01	Mean : 721.6	Mean : 238.5	Mean : 2.207	Mean :4.465e+09	Mode
:character	NA	Qu.:2016	Qu.:271.0	Qu.:108.00	Qu.:56.0	Qu.:0.0	Qu.:2.000	Qu.:9.940e+08	:character
NA	Max.	Max.	Max.	Max.	Max.	Max.	Max.	Max.	NA
NA	:2018	:365.0	:144.00	:1943.7488	:14497.0	:14496.0	:12920.000	:4.996e+13	NA

A listing of the first ten rows provides a sample of the features associated with each observation.

address	year	day	length	weight	count	looped	neighbors	income	label
111K8kZAEjg245r2cM6y9zgJGHZtJPy6	2017	11	18	0.0083333	1	0	2	100050000	princetonCerber
1123pJv8jzeFQaCV4w644pzQJzVWay2zcA	2016	132	44	0.0002441	1	0	1	100000000	princetonLocky
112536im7hy6wtKbpH1qYDWtTyMRAcA2p7	2016	246	0	1.0000000	1	0	2	200000000	princetonCerber
1126eDRw2wgSkWosjTCre8cjjQW8sSeWH7	2016	322	72	0.0039063	1	0	2	71200000	princetonCerber
1129TSjKtx65E35GiUo4AYVeyo48twbrGX	2016	238	144	0.0728484	456	0	1	200000000	princetonLocky
112AmFATxzhSpvtz1hfpz3Zrw3BG276pc	2016	96	144	0.0846140	2821	0	1	50000000	princetonLocky

This data set has 2,916,697 observations of ten features associated with a sample of transactions from the Bitcoin blockchain. The ten features include *address* as a unique identifier, the six features defined previously (*income*, *neighbors*, *weight*, *length*, *count*, *loop*), two temporal features in the form of *year* and *day* (of the year as 1 to 365), and a categorical feature called *label* that categorizes each address as either “white” (meaning not connected to any ransomware activity), or one of 28 known ransomware groups as identified by three independent ransomware analysis teams (Montreal, Princeton, and Padua)^[3].

The original research team downloaded and parsed the entire Bitcoin transaction graph from January 2009 to December 2018. Based on a 24 hour time interval, daily transactions on the network were extracted and the Bitcoin graph was formed. Network edges that transferred less than \$0.3 were filtered out since ransom amounts are rarely below this threshold. Ransomware addresses are taken from three widely adopted studies: Montreal, Princeton and Padua. “White” Bitcoin addresses were capped at one thousand per day while the entire network has up to 800,000 addresses daily.^[5]

Goal

The goal of this project is to apply different machine learning algorithms to the same data set used in the original paper to produce an acceptable predictive model for categorizing ransomware addresses correctly. Improving on the results of the original paper in some way, while not strictly necessary for the purposes of the project, would be a notable sign of success.

Outline of Steps Taken

1. Analyze data set numerically and visually. Notice any pattern, look for insights.
2. Binary separation using Self Organizing Maps.
3. Improved Binary separation using Random Forest.
4. Categorical classification using Self Organizing Maps.
5. Visualize clustering to analyze results further.
6. Generate Confusion Matrix to quantify results.

Table 3: Ransomware group labels and frequency counts for full data set

	x		x
montrealAPT	11	montrealRazy	13
montrealComradeCircle	1	montrealSam	1
montrealCryptConsole	7	montrealSamSam	62
montrealCryptXXX	2419	montrealVenusLocker	7
montrealCryptoLocker	9315	montrealWannaCry	28
montrealCryptoTorLocker2015	55	montrealXLocker	1
montrealDMALocker	251	montrealXLockerv5.0	7
montrealDMALockerv3	354	montrealXTPLocker	8
montrealEDA2	6	paduaCryptoWall	12390
montrealFlyper	9	paduaJigsaw	2
montrealGlobe	32	paduaKeRanger	10
montrealGlobeImposter	55	princetonCerber	9223
montrealGlobev3	34	princetonLocky	6625
montrealJigSaw	4	white	2875284
montrealNoobCrypt	483		

Data Analysis

Hardware Specification

All of the analysis in this report was conducted on a single laptop computer, a Lenovo Yoga S1 from late 2013 with the following specs:

- CPU: Intel i7-4600U @ 3.300GHz (4th Gen quad-core i7)
- RAM: 8217MB DDR3L @ 1600 MHz (8 GB)
- OS: Slackware64-current (15.0 RC1) `x86_64-slackware-linux-gnu` (64-bit GNU/Linux)
- R version 4.0.0 (2020-04-24) – “Arbor Day” (built from source using scripts from slackbuilds.org)
- RStudio Version 1.4.1106 “Tiger Daylily” (2389bc24, 2021-02-11) for CentOS 8 (converted using `rpm2tgz`)

Data Preparation

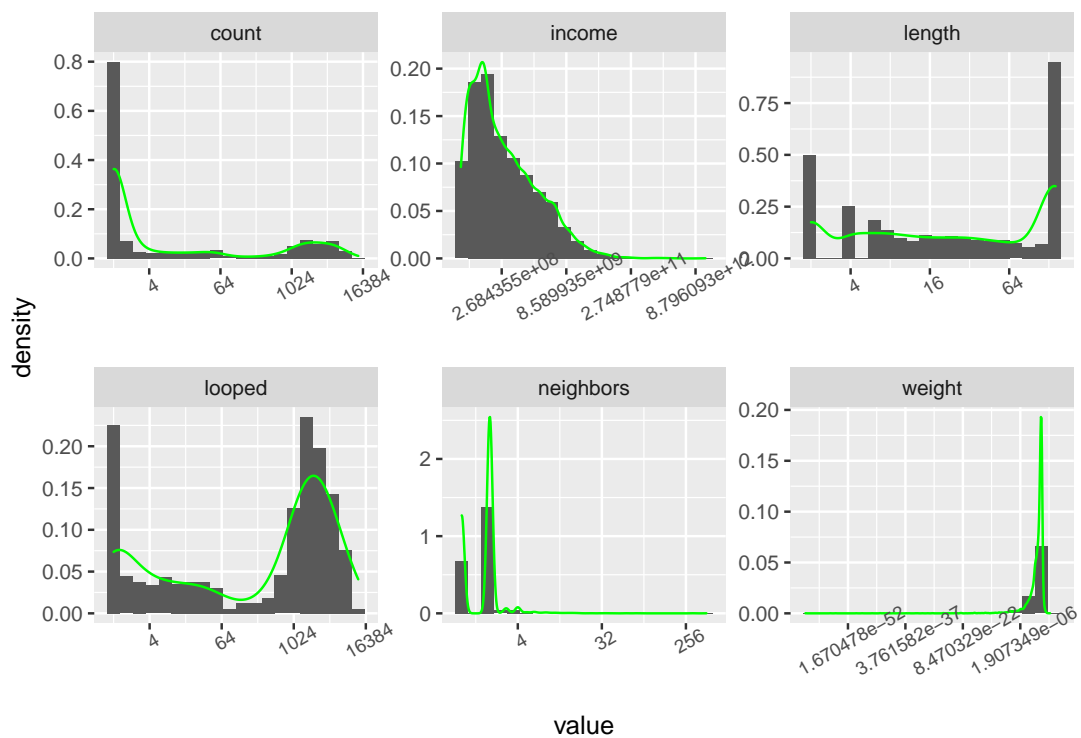
It is immediately apparent that this is a rather large data set. The usual practice of partitioning out eighty to ninety percent of the data for a training set results in a data set that is too large to process given the hardware available. For reasons that no longer apply, the original data set was first split in half with 50% reserved as “validation set” and the other 50% used as the “working set”. This working set was again split in half, to give a “training set” that was of a reasonable size to deal with. At this point the partitions were small enough to work with, so the sample partitions were not further refined. This is a potential area for later optimization. Careful sampling was carried out to ensure that the ransomware groups were represented in each sample.

Exploration and Visualization (graphic rework from here to Modeling Approach)

The ransomware addresses make up less than 2% of the overall data set. This presents a challenge as the target observations are sparse within the data set, especially when we consider that this is then divided into 28 subsets. In fact, some of the ransomware groups have only a single member, making categorization a dubious task. At least there are no missing values to worry about.

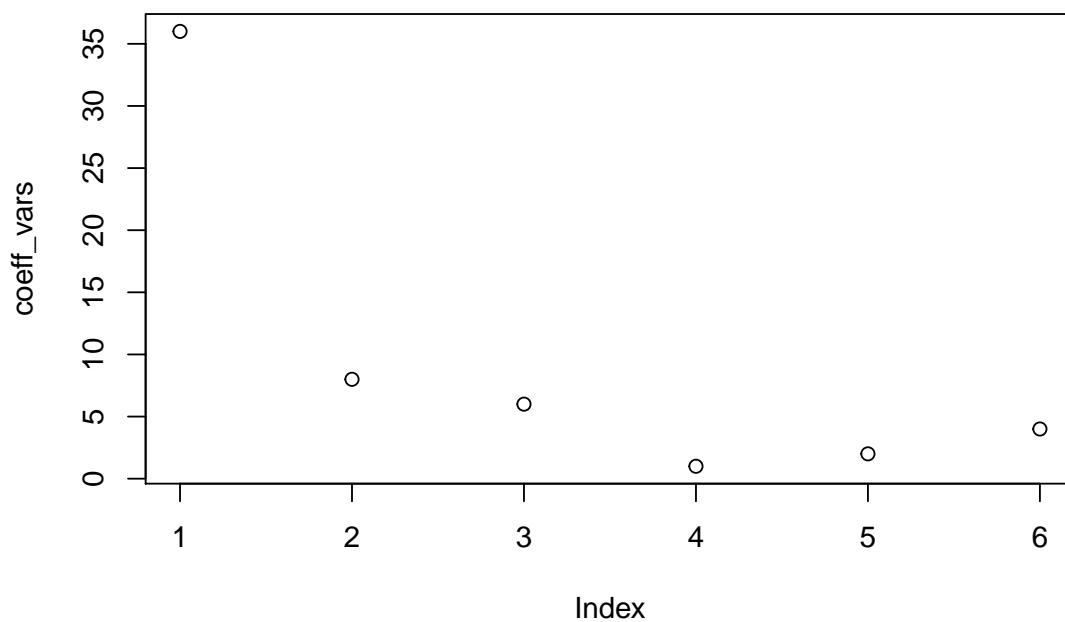
The proportion of ransomware addresses in the original data set is 0.0141986. The total number of NA or missing values in the original data set is 0.

Let’s take a look at the distribution of the different features. Note how skewed the non-temporal features are, some of them being bimodal. Looks better on a log scale x-axis.



Now we can compare the relative spread of each feature by calculating the coefficient of variation for each column. Larger coefficients of variation indicate larger relative spread compared to other columns.

	x
income	36
neighbors	8
weight	6
length	1
count	2
looped	4



From this, it appears that income has the widest range of variability, followed by neighbors. These are also the features that are most strongly skewed to the right, meaning that a few addresses have really high values for each of these features while the bulk of the data set has very low values for these numbers.

Now do the following (after filling in methods, results, and conclusions, since those are done already):

!) Break into groups somehow. Graph variables per group? Show how the variables are distributed for each ransomware group? Percent ransomware per each day of the week, for example. Is ransomware more prevalent on a particular day of the week? Break other numerical values into bins, and graph percentage per bin. Look for trends and correlations between groups/variables, and display them here. MORE OF THIS....

The percentage of wallets with less than one full bitcoin as their balance is 0.0134611 .

Insights Gained from Exploration (graphic rework ends here)

From the previous visual and statistical exploration of the data, it becomes clear what the challenge is. Ransomware related addresses are very sparse in the data set, making up less than 2% of all addresses. That small percentage is also further classified into 28 groups. Perhaps the original paper was a bit too ambitious in trying to categorize all the addresses into 29 categories, including the “white” addresses. To simplify our approach, we will categorize the addresses in a binary way, either “white” or “black”, where “black” signifies an association with ransomware transactions. Asking this as a “ransomware or not-ransomware” question allows for application of methods that are known to be impractical otherwise.

Modeling approach

Akcora et al. applied a Random Forest approach to the data, however “Despite improving data scarcity, [...] tree based methods (i.e., Random Forest and XGBoost) fail to predict any ransomware family”.[3, 11] Considering all ransomware addresses as belonging to a single group might improve the predictive power of such methods, making Random Forest worth another try.

The topological description of the data set inspired a search for topological machine learning methods, although one does not necessitate the other. Searching for *topo* in the documentation for the `caret` package [6] resulted in the entry for Self Organizing Maps, supplied by the `kohonen` package. The description at CRAN [7] was intriguing enough to merit further investigation.

Initially, the categorization of ransomware into the 28 different families was attempted using SOMs. This proved to be very resource intensive, requiring more time and RAM than was available at the time. Although it did help to illuminate how SOMs are configured, the resource requirements of the script became a deterrent. It was at this point that the SOMS were applied in a binary way, classifying all ransomware addresses as merely “black”. This seemed to reduce RAM usage to the point of being feasible on the available hardware.

Being unsure of the SOM method, since it was not covered in the coursework at any point, a familiar method was sought out to compare the results to. This is when Random Forest was applied to the data set in a binary way. Much to the surprise of the author, not only did the Random Forest approach result in an acceptable model, it surpassed the model produced by the SOM approach.

The author was tempted to leave it there and write up a comparison of the two approaches to the binary problem. However, a nagging feeling that more could be done eventually inspired a second look at the categorical problem of categorizing the ransomware addresses into the 28 families. Given the high accuracy and precision of the Random Forest approach to the binary problem, it became apparent that the sparseness of the ransomware in the larger set had been eliminated completely, as had any chances of false positives. There are a few cases of false negatives, depending on how the randomization is done during the sampling process. However, the Random Forest method has never produced a false positive yet, meaning it never seems to predict a truly white address as being black. Hence, by applying the Random Forest method first, we have essentially filtered out any possibility of false positives, which is exactly what plagued the original paper by Akcora et al.[3]

Finally, a two-part method was devised to first separate the addresses into black and white groups, and then further classify the black addresses into ransomware families. Let’s explore each of these separately.

Method Part 0: Binary SOMS as a first attempt to isolate ransomware addresses.

Lets see how well the SOM approach can model the data in a black/white fashion.

Here are the results of the binary SOM model.

Test set:

	black	white
black	10353	0
white	0	718806

Validation set:

	black	white
black	20706	0
white	1	1437617

This winds up being the most resource intensive and least accurate method out of those tried. It was left out of the final version of the script and is not really worth running except to compare to the next method.

Method Part 1: Binary Random Forest to isolate ransomware addresses before categorization.

A Random Forest model is trained using ten-fold cross validation and a tuning grid with the number of variables randomly sampled as candidates at each split `mtry` set to the values = 2, 4, 6, 8, 10, 12, each one being checked for optimization.

The confusion matrix for the test set shows excellent results, specifically in the areas of accuracy and precision.

	black	white
black	104	0
white	0	7188

Here are the overall results...

	x
Accuracy	1.0000000
Kappa	1.0000000
AccuracyLower	0.9994942
AccuracyUpper	1.0000000
AccuracyNull	0.9857378
AccuracyPValue	0.0000000
McnemarPValue	NaN

Here are the results by class...

	x
Sensitivity	1.0000000
Specificity	1.0000000
Pos Pred Value	1.0000000
Neg Pred Value	1.0000000

	x
Precision	1.0000000
Recall	1.0000000
F1	1.0000000
Prevalence	0.0142622
Detection Rate	0.0142622
Detection Prevalence	0.0142622
Balanced Accuracy	1.0000000

The confusion matrix for the full ransomware set is very similar to that of the test set.

Here is the confusion matrix for the full ransomware data set.

	black	white
black	40496	0
white	917	2875284

Here are the big overall results. . . .

	x
Accuracy	0.9996856
Kappa	0.9886452
AccuracyLower	0.9996646
AccuracyUpper	0.9997056
AccuracyNull	0.9858014
AccuracyPValue	0.0000000
McnemarPValue	0.0000000

Here are the big set results by class. . . .

	x
Sensitivity	0.9778572
Specificity	1.0000000
Pos Pred Value	1.0000000
Neg Pred Value	0.9996812
Precision	1.0000000
Recall	0.9778572
F1	0.9888046
Prevalence	0.0141986
Detection Rate	0.0138842
Detection Prevalence	0.0138842
Balanced Accuracy	0.9889286

Method Part 2: Categorical SOMs to categorize predicted ransomware addresses.

Now we train a new model after throwing away all “white” addresses. The predictions from the Random Forest model are used to isolate all “black” addresses for further classification into ransomware addresses using SOMs.

When selecting the grid size for a Self Organizing Map, there are at least two different schools of thought. The two that were tried here are explained (with supporting documentation) on a Researchgate forum.[8] The first method is based on the size of the training set, and in this case results in a larger, more accurate map. The second method

is based on the number of known categories to classify the data into, and in this case results in a smaller, less accurate map. For this script, a grid size of 27 has been selected.

A summary of the results for the categorization of black addresses into ransomware families follows. For the full table of predictions and statistics, see the Appendix.

Here are the overall results.

	x
Accuracy	0.9990118
Kappa	0.9987173
AccuracyLower	0.9984742
AccuracyUpper	0.9993963
AccuracyNull	0.3062556
AccuracyPValue	0.0000000
McnemarPValue	NaN

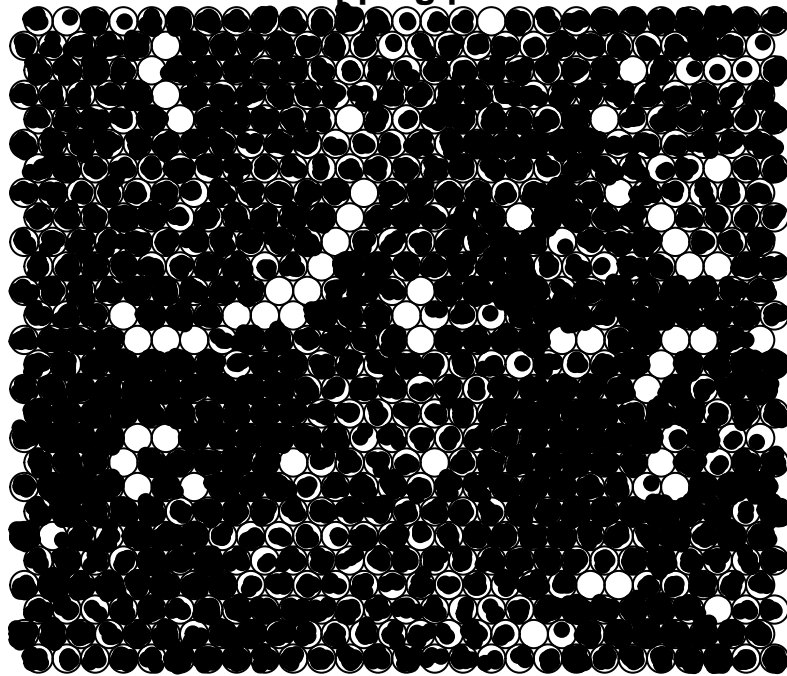
Here are the results by class.

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: montrealAPT	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealComradeCircle	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealCryptConsole	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealCryptXXX	0.9975450	1.0000000	1.0000000	0.9998423	1.0000000	0.9975450	0.9987710	0.0603815	0.0602332	0.0602332	0.9987725
Class: montrealCryptoLocker	0.9993645	0.9996778	0.9989414	0.9998066	0.9989414	0.9993645	0.9991529	0.2332740	0.2331258	0.2333729	0.9995212
Class: montrealCryptoTorLocker2015	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealDMALocker	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealDMALocker3	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealEDA2	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealFlyper	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealGlobe	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montreal-GlobeImposter	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealGlobev3	0.9285714	1.0000000	1.0000000	0.9999506	1.0000000	0.9285714	0.9629630	0.0006918	0.0006424	0.0006424	0.9642857
Class: montrealJigSaw	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealNoobCrypt	0.9957265	0.9999500	0.9957265	0.9999500	0.9957265	0.9957265	0.9957265	0.0115624	0.0115130	0.0115624	0.9978383
Class: montrealRazy	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealSam	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealSamSam	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealVenusLocker	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealWannaCry	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealXLocker	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealXLockerv5.0	0.3333333	1.0000000	1.0000000	0.9999012	1.0000000	0.3333333	0.5000000	0.0001482	0.0000494	0.0000494	0.6666667
Class: montrealXTPLocker	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: paduaCryptoWall	0.9993546	0.9995726	0.9990323	0.9997151	0.9990323	0.9993546	0.9991934	0.3062556	0.3060579	0.3063544	0.9994636
Class: paduaJigsaw	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: paduaKeRanger	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: princetonCerber	1.0000000	0.9995520	0.9984845	1.0000000	0.9984845	1.0000000	0.9992417	0.2278881	0.2278881	0.2282340	0.9997760
Class: princetonLocky	0.9981447	0.9999412	0.9996903	0.9996472	0.9996903	0.9981447	0.9989169	0.1597984	0.1595019	0.1595513	0.9990430
Class: white	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA

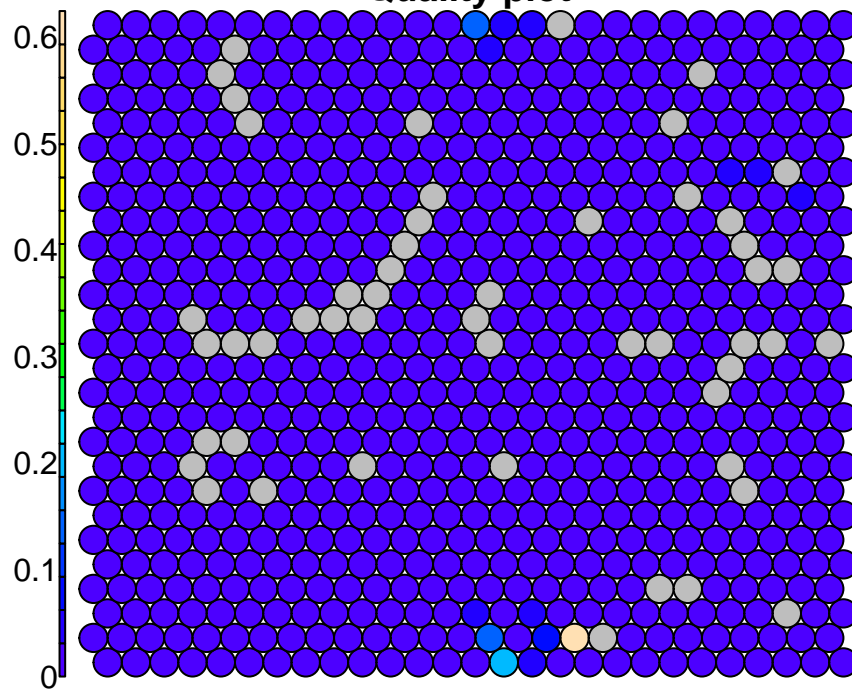
Clustering Visualizations: Heatmaps and K-means clustering

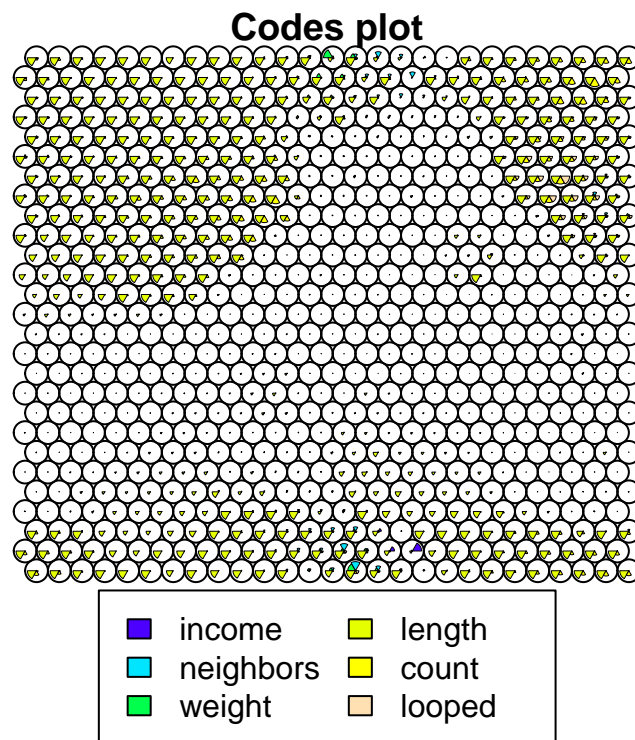
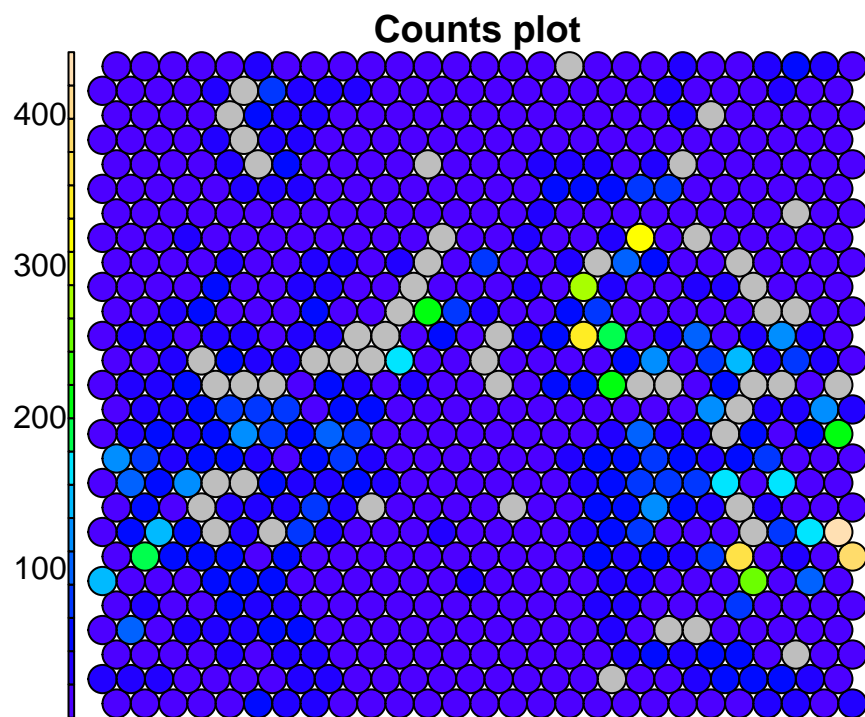
Here are some graphs, tell a bit more about them.

Mapping plot

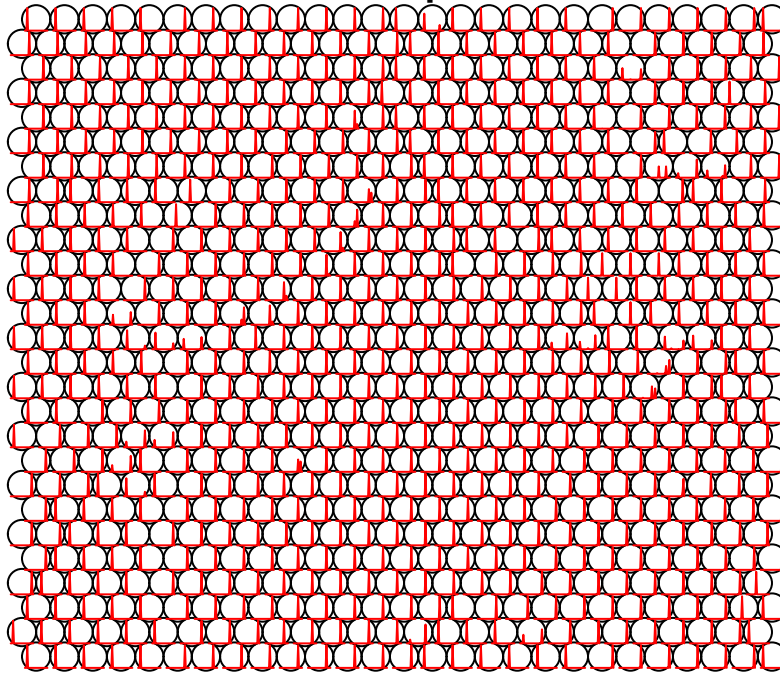


Quality plot

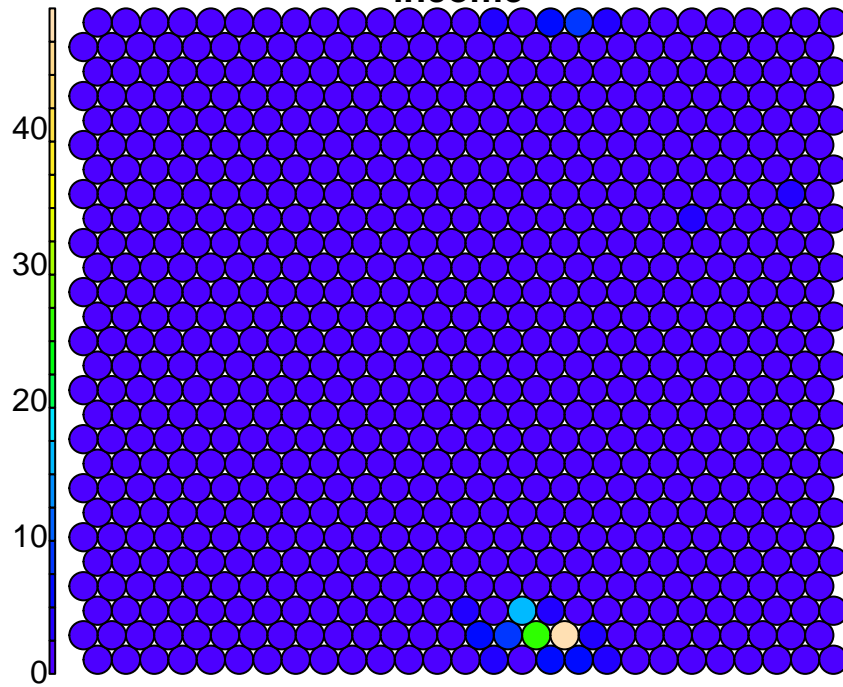


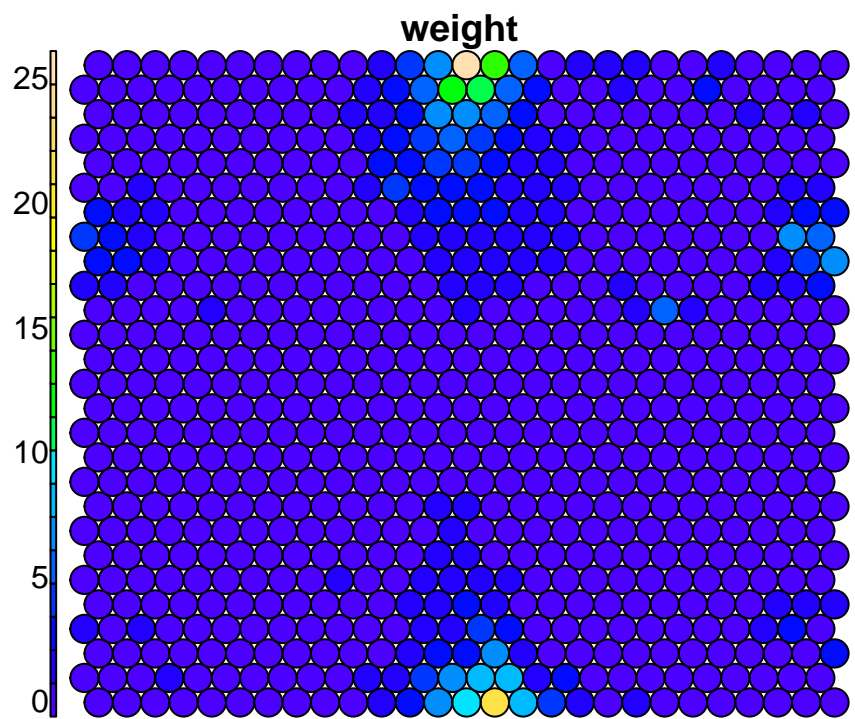
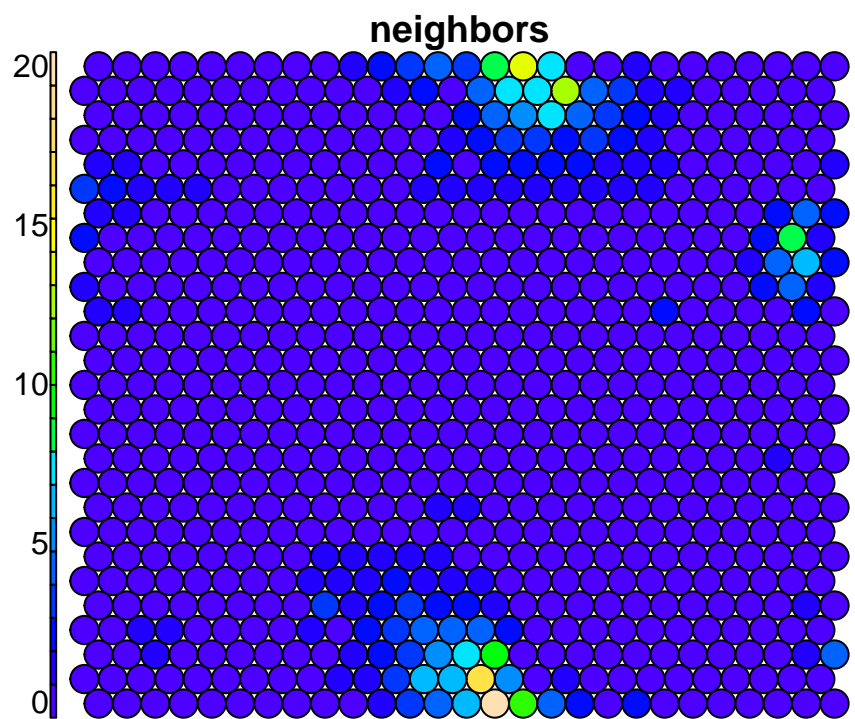


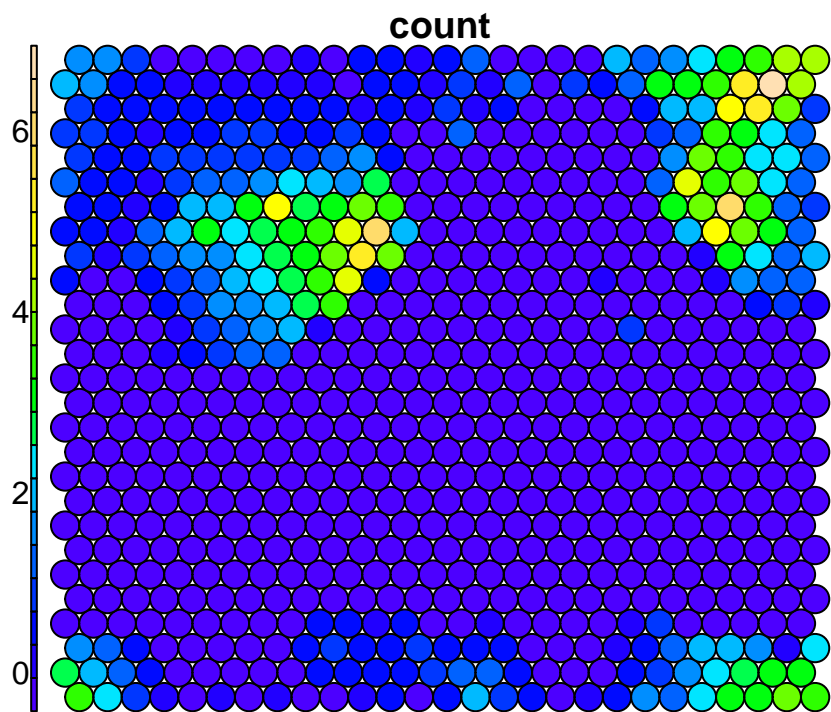
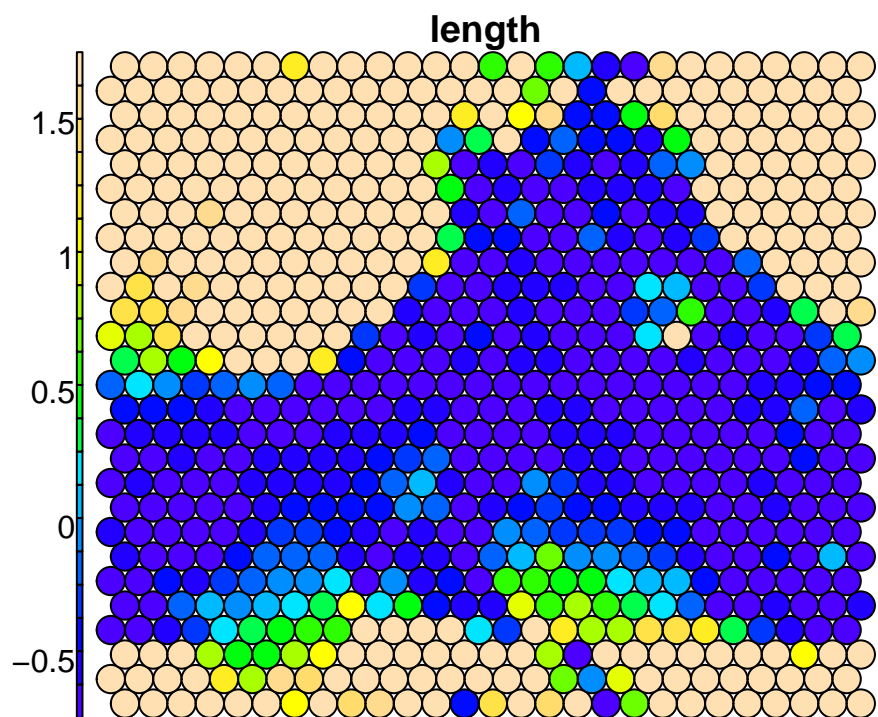
Codes plot

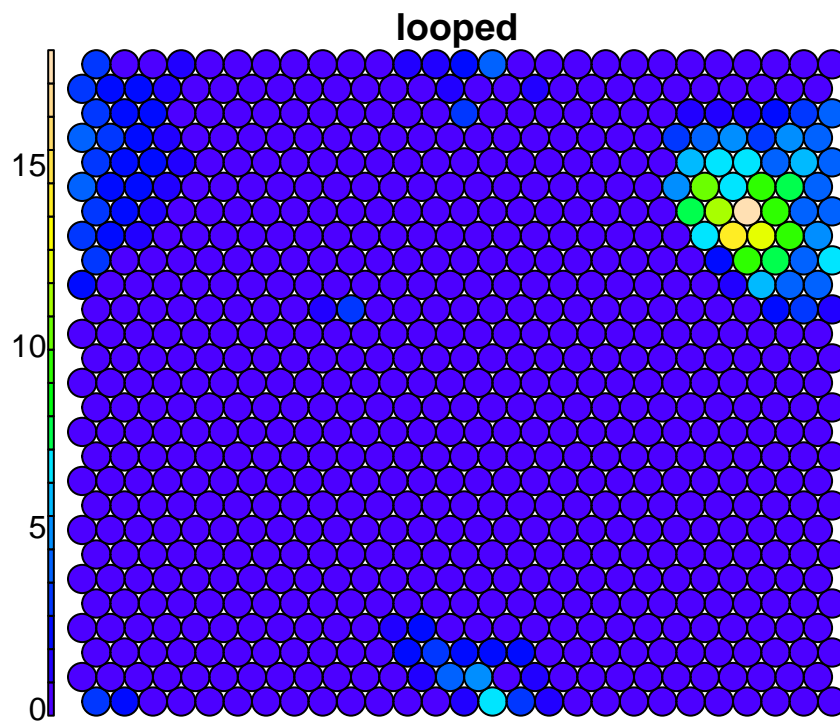


income



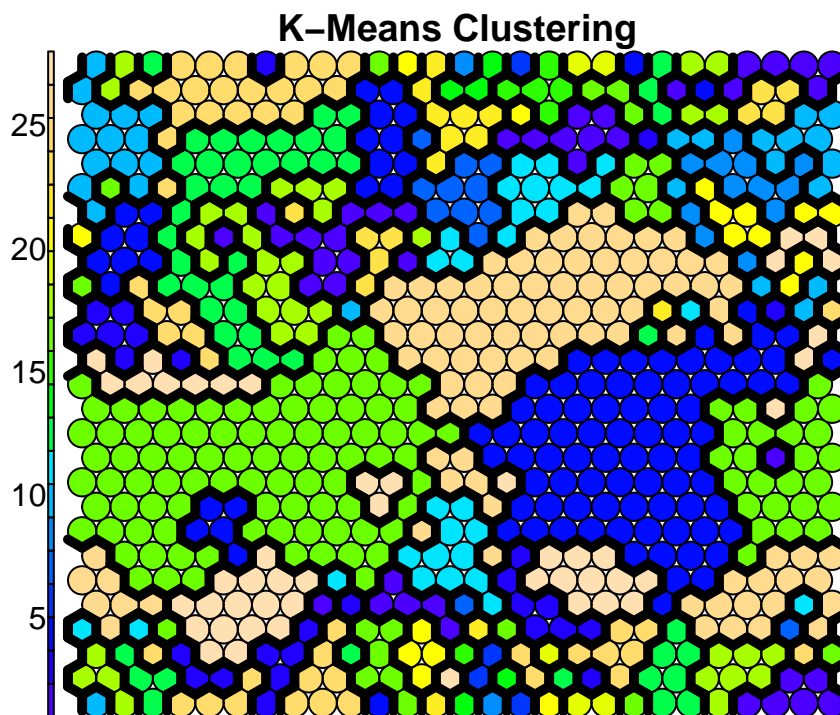






K-means clustering offers a nice way of visualizing the final SOM grid and the categorical boundaries that were formed by the model.

K-means clustering categorizes the SOM grid by adding boundaries to the classification groups. This is the author's favorite graph in the entire report.



Results & Performance

Results

The first attempt to isolate ransomware using SOMs resulted in a model with an accuracy of 0.999999314281326 and precision 1.

The the second attempt to isolate ransomware using Random forest resulted in a model with an accuracy of 0.999685603269726 and precision 1.

Classifying the ransomware predicted by the second attempt into 28 ransomware families resulted in a model with an overall accuracy of 0.999011760055341 and minimum nonzero precision of 0.995726495726496.

Performance

The script runs on the aforementioned hardware in less than five minutes and uses less than 4GB of RAM. Given that the Bitcoin network produces one new block every ten minutes on average, then real-time analysis could theoretically be conducted on each block as it is announced using even moderate computing resources. Just for kicks, the final script was also run on a more humble computer with the following specifications:

- CPU: Intel Atom N2600 @ 1.600GHz (64-bit Intel Atom quad-core)
- RAM: 3911MB DDR3 @ 800 MT/s (4 GB)

This is a computer known for being slow and clunky. Even on this device, which runs the same operating system and software as the hardware listed previously, the total run time for the script is around 1665 seconds. At nearly 28 minutes, this is not fast enough to analyze the Bitcoin blockchain in real time, but it does show that the script can be run on very modest hardware to completion.

Summary

Comparison to results from original paper

In the original paper by Akcora et al., they tested several different sets of parameters on their TDA model. According to them, “In the best TDA models for each ransomware family, we predict **16.59 false positives for each true positive**. In turn, this number is 27.44 for the best non-TDA models.”[3] In fact, the highest Precision (a.k.a. Positive Predictive Value, defined as $TP/(TP+FP)$) they achieved was only 0.1610. By comparison, although several of our predicted classes had zero or NA precision values, the lowest non-zero precision value is 0.995726495726496, with many well above that, approaching one in a few cases.

One might say that we are comparing apples to oranges in a sense, because their method was one single model, while these results are from a two-method stack. Still, given the run time of the final script, I think the two-model approach is superior in this case, especially when measured in terms of precision and avoiding false positives.

Limitations

SOMs seem like they are easy to misconfigure. Perhaps a dual Random Forest approach would be better. this has not been attempted yet, as the two method approach presented here was satisfactory enough to present in a report.

Future Work

I only scratched the surface of the SOM algorithm which seems to have many implementations and parameters that could be investigated further and possibly optimized via cross-validation. Also, a dual Random Forest approach to first isolate the ransomware addresses and also

The script itself has a few areas that could be further optimization. The sampling method does what it needs to do, but the ratios taken for each set could possibly be optimized.

Conclusion

This paper/report presents a reliable method for classifying Bitcoin addresses into known ransomware families, while at the same time avoiding false positives by filtering them out using a binary method before classifying them further. It leaves the author of the paper wondering how much harder it would be to perform the same task for ransomware that uses privacy coins. Certain cryptocurrency networks utilize privacy coins, such as Monero, that obfuscate transactions from being analyzed in the same way that the Bitcoin network has been analyzed here. Some progress has been made towards analyzing such networks[9], but the developers of such networks continually evolve the code to complicate transaction tracking. This could be another good area for future research.

References

- [1] Adam Brian Turner, Stephen McCombie and Allon J. Uhlmann (November 30, 2020) Analysis Techniques for Illicit Bitcoin Transactions
- [2] Daniel Goldsmith, Kim Grauer and Yonah Shmalo (April 16, 2020) Analyzing hack subnetworks in the bitcoin transaction graph
- [3] Cuneyt Gurcan Akcora, Yitao Li, Yulia R. Gel, Murat Kantarcioglu (June 19, 2019) BitcoinHeist: Topological Data Analysis for Ransomware Detection on the Bitcoin Blockchain
- [4] UCI Machine Learning Repository <https://archive.ics.uci.edu/ml/index.php>
- [5] BitcoinHeist Ransomware Address Dataset
<https://archive.ics.uci.edu/ml/datasets/BitcoinHeistRansomwareAddressDataset>
- [6] Available Models - The `caret` package <http://topepo.github.io/caret/available-models.html>
- [7] Ron Wehrens and Johannes Kruisselbrink, Package ‘`kohonen`’ @ CRAN (2019) <https://cran.r-project.org/web/packages/kohonen/kohonen.pdf>
- [8] How many nodes for self-organizing maps? (Oct 22, 2021) <https://www.researchgate.net/post/How-many-nodes-for-self-organizing-maps>
- [9] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin (April 23, 2018) An Empirical Analysis of Traceability in the Monero Blockchain

Appendix:

Categorical SOM ransomware family prediction table and confusion matrix

Here are the full prediction results for the categorization of black addresses into ransomware families. It is assumed that all white address have already been removed.

Confusion Matrix and Statistics

##

##

	Reference	
## Prediction	montrealAPT	montrealComradeCircle
## montrealAPT	0	0
## montrealComradeCircle	0	0
## montrealCryptConsole	0	0
## montrealCryptXXX	0	0
## montrealCryptoLocker	0	0
## montrealCryptoTorLocker2015	0	0
## montrealDMALocker	0	0
## montrealDMALockerv3	0	0
## montrealEDA2	0	0
## montrealFlyper	0	0
## montrealGlobe	0	0
## montrealGlobeImposter	0	0
## montrealGlobev3	0	0
## montrealJigSaw	0	0
## montrealNoobCrypt	0	0
## montrealRazy	0	0
## montrealSam	0	0
## montrealSamSam	0	0
## montrealVenusLocker	0	0
## montrealWannaCry	0	0
## montrealXLocker	0	0
## montrealXLockerv5.0	0	0
## montrealXTPLocker	0	0
## paduaCryptoWall	0	0
## paduaJigsaw	0	0
## paduaKeRanger	0	0
## princetonCerber	0	0
## princetonLocky	0	0
## white	0	0

##

	Reference	
## Prediction	montrealCryptConsole	montrealCryptXXX
## montrealAPT	0	0
## montrealComradeCircle	0	0
## montrealCryptConsole	0	0
## montrealCryptXXX	0	1219
## montrealCryptoLocker	0	1
## montrealCryptoTorLocker2015	0	0
## montrealDMALocker	0	0
## montrealDMALockerv3	0	0
## montrealEDA2	0	0
## montrealFlyper	0	0
## montrealGlobe	0	0
## montrealGlobeImposter	0	0
## montrealGlobev3	0	0
## montrealJigSaw	0	0

##	montrealNoobCrypt	0	0
##	montrealRazy	0	0
##	montrealSam	0	0
##	montrealSamSam	0	0
##	montrealVenusLocker	0	0
##	montrealWannaCry	0	0
##	montrealXLocker	0	0
##	montrealXLockerv5.0	0	0
##	montrealXTPLocker	0	0
##	paduaCryptoWall	0	0
##	paduaJigsaw	0	0
##	paduaKeRanger	0	0
##	princetonCerber	0	2
##	princetonLocky	0	0
##	white	0	0
##		Reference	
##	Prediction	montrealCryptoLocker	montrealCryptoTorLocker2015
##	montrealAPT	0	0
##	montrealComradeCircle	0	0
##	montrealCryptConsole	0	0
##	montrealCryptXXX	0	0
##	montrealCryptoLocker	4718	0
##	montrealCryptoTorLocker2015	0	0
##	montrealDMALocker	0	0
##	montrealDMALockerv3	0	0
##	montrealEDA2	0	0
##	montrealFlyper	0	0
##	montrealGlobe	0	0
##	montrealGlobeImposter	0	0
##	montrealGlobev3	0	0
##	montrealJigSaw	0	0
##	montrealNoobCrypt	0	0
##	montrealRazy	0	0
##	montrealSam	0	0
##	montrealSamSam	0	0
##	montrealVenusLocker	0	0
##	montrealWannaCry	0	0
##	montrealXLocker	0	0
##	montrealXLockerv5.0	0	0
##	montrealXTPLocker	0	0
##	paduaCryptoWall	3	0
##	paduaJigsaw	0	0
##	paduaKeRanger	0	0
##	princetonCerber	0	0
##	princetonLocky	0	0
##	white	0	0
##		Reference	
##	Prediction	montrealDMALocker	montrealDMALockerv3
##	montrealAPT	0	0
##	montrealComradeCircle	0	0
##	montrealCryptConsole	0	0
##	montrealCryptXXX	0	0
##	montrealCryptoLocker	0	0
##	montrealCryptoTorLocker2015	0	0
##	montrealDMALocker	0	0
##	montrealDMALockerv3	0	0

##	montrealEDA2	0	0
##	montrealFlyper	0	0
##	montrealGlobe	0	0
##	montrealGlobeImposter	0	0
##	montrealGlobev3	0	0
##	montrealJigSaw	0	0
##	montrealNoobCrypt	0	0
##	montrealRazy	0	0
##	montrealSam	0	0
##	montrealSamSam	0	0
##	montrealVenusLocker	0	0
##	montrealWannaCry	0	0
##	montrealXLocker	0	0
##	montrealXLockerv5.0	0	0
##	montrealXTPLocker	0	0
##	paduaCryptoWall	0	0
##	paduaJigsaw	0	0
##	paduaKeRanger	0	0
##	princetonCerber	0	0
##	princetonLocky	0	0
##	white	0	0
##		Reference	
##	Prediction	montrealEDA2	montrealFlyper
##	montrealAPT	0	0
##	montrealComradeCircle	0	0
##	montrealCryptConsole	0	0
##	montrealCryptXXX	0	0
##	montrealCryptoLocker	0	0
##	montrealCryptoTorLocker2015	0	0
##	montrealDMALocker	0	0
##	montrealDMALockerv3	0	0
##	montrealEDA2	0	0
##	montrealFlyper	0	0
##	montrealGlobe	0	0
##	montrealGlobeImposter	0	0
##	montrealGlobev3	0	0
##	montrealJigSaw	0	0
##	montrealNoobCrypt	0	0
##	montrealRazy	0	0
##	montrealSam	0	0
##	montrealSamSam	0	0
##	montrealVenusLocker	0	0
##	montrealWannaCry	0	0
##	montrealXLocker	0	0
##	montrealXLockerv5.0	0	0
##	montrealXTPLocker	0	0
##	paduaCryptoWall	0	0
##	paduaJigsaw	0	0
##	paduaKeRanger	0	0
##	princetonCerber	0	0
##	princetonLocky	0	0
##	white	0	0
##		Reference	
##	Prediction	montrealGlobeImposter	montrealGlobev3
##	montrealAPT	0	0
##	montrealComradeCircle	0	0

##	montrealCryptConsole	0	0
##	montrealCryptXXX	0	0
##	montrealCryptoLocker	0	0
##	montrealCryptoTorLocker2015	0	0
##	montrealDMALocker	0	0
##	montrealDMALockerv3	0	0
##	montrealEDA2	0	0
##	montrealFlyper	0	0
##	montrealGlobe	0	0
##	montrealGlobeImposter	0	0
##	montrealGlobev3	0	13
##	montrealJigSaw	0	0
##	montrealNoobCrypt	0	0
##	montrealRazy	0	0
##	montrealSam	0	0
##	montrealSamSam	0	0
##	montrealVenusLocker	0	0
##	montrealWannaCry	0	0
##	montrealXLocker	0	0
##	montrealXLockerv5.0	0	0
##	montrealXTPLocker	0	0
##	paduaCryptoWall	0	0
##	paduaJigsaw	0	0
##	paduaKeRanger	0	0
##	princetonCerber	0	0
##	princetonLocky	0	1
##	white	0	0

##		Reference		
##	Prediction	montrealJigSaw	montrealNoobCrypt	montrealRazy
##	montrealAPT	0	0	0
##	montrealComradeCircle	0	0	0
##	montrealCryptConsole	0	0	0
##	montrealCryptXXX	0	0	0
##	montrealCryptoLocker	0	1	0
##	montrealCryptoTorLocker2015	0	0	0
##	montrealDMALocker	0	0	0
##	montrealDMALockerv3	0	0	0
##	montrealEDA2	0	0	0
##	montrealFlyper	0	0	0
##	montrealGlobe	0	0	0
##	montrealGlobeImposter	0	0	0
##	montrealGlobev3	0	0	0
##	montrealJigSaw	0	0	0
##	montrealNoobCrypt	0	233	0
##	montrealRazy	0	0	0
##	montrealSam	0	0	0
##	montrealSamSam	0	0	0
##	montrealVenusLocker	0	0	0
##	montrealWannaCry	0	0	0
##	montrealXLocker	0	0	0
##	montrealXLockerv5.0	0	0	0
##	montrealXTPLocker	0	0	0
##	paduaCryptoWall	0	0	0
##	paduaJigsaw	0	0	0
##	paduaKeRanger	0	0	0
##	princetonCerber	0	0	0

##	princetonLocky	0	0	0
##	white	0	0	0
##		Reference		
##	Prediction	montrealSam	montrealSamSam	montrealVenusLocker
##	montrealAPT	0	0	0
##	montrealComradeCircle	0	0	0
##	montrealCryptConsole	0	0	0
##	montrealCryptXXX	0	0	0
##	montrealCryptoLocker	0	0	0
##	montrealCryptoTorLocker2015	0	0	0
##	montrealDMALocker	0	0	0
##	montrealDMALockerv3	0	0	0
##	montrealEDA2	0	0	0
##	montrealFlyper	0	0	0
##	montrealGlobe	0	0	0
##	montrealGlobeImposter	0	0	0
##	montrealGlobev3	0	0	0
##	montrealJigSaw	0	0	0
##	montrealNoobCrypt	0	0	0
##	montrealRazy	0	0	0
##	montrealSam	0	0	0
##	montrealSamSam	0	0	0
##	montrealVenusLocker	0	0	0
##	montrealWannaCry	0	0	0
##	montrealXLocker	0	0	0
##	montrealXLockerv5.0	0	0	0
##	montrealXTPLocker	0	0	0
##	paduaCryptoWall	0	0	0
##	paduaJigsaw	0	0	0
##	paduaKeRanger	0	0	0
##	princetonCerber	0	0	0
##	princetonLocky	0	0	0
##	white	0	0	0
##		Reference		
##	Prediction	montrealWannaCry	montrealXLocker	
##	montrealAPT	0	0	
##	montrealComradeCircle	0	0	
##	montrealCryptConsole	0	0	
##	montrealCryptXXX	0	0	
##	montrealCryptoLocker	0	0	
##	montrealCryptoTorLocker2015	0	0	
##	montrealDMALocker	0	0	
##	montrealDMALockerv3	0	0	
##	montrealEDA2	0	0	
##	montrealFlyper	0	0	
##	montrealGlobe	0	0	
##	montrealGlobeImposter	0	0	
##	montrealGlobev3	0	0	
##	montrealJigSaw	0	0	
##	montrealNoobCrypt	0	0	
##	montrealRazy	0	0	
##	montrealSam	0	0	
##	montrealSamSam	0	0	
##	montrealVenusLocker	0	0	
##	montrealWannaCry	0	0	
##	montrealXLocker	0	0	

##	montrealXLockerv5.0	0	0
##	montrealXTPLocker	0	0
##	paduaCryptoWall	0	0
##	paduaJigsaw	0	0
##	paduaKeRanger	0	0
##	princetonCerber	0	0
##	princetonLocky	0	0
##	white	0	0

##		Reference	
##	Prediction	montrealXLockerv5.0	montrealXTPLocker
##	montrealAPT	0	0
##	montrealComradeCircle	0	0
##	montrealCryptConsole	0	0
##	montrealCryptXXX	0	0
##	montrealCryptoLocker	0	0
##	montrealCryptoTorLocker2015	0	0
##	montrealDMALocker	0	0
##	montrealDMALockerv3	0	0
##	montrealEDA2	0	0
##	montrealFlyper	0	0
##	montrealGlobe	0	0
##	montrealGlobeImposter	0	0
##	montrealGlobev3	0	0
##	montrealJigSaw	0	0
##	montrealNoobCrypt	0	0
##	montrealRazy	0	0
##	montrealSam	0	0
##	montrealSamSam	0	0
##	montrealVenusLocker	0	0
##	montrealWannaCry	0	0
##	montrealXLocker	0	0
##	montrealXLockerv5.0	1	0
##	montrealXTPLocker	0	0
##	paduaCryptoWall	2	0
##	paduaJigsaw	0	0
##	paduaKeRanger	0	0
##	princetonCerber	0	0
##	princetonLocky	0	0
##	white	0	0

##		Reference		
##	Prediction	paduaCryptoWall	paduaJigsaw	paduaKeRanger
##	montrealAPT	0	0	0
##	montrealComradeCircle	0	0	0
##	montrealCryptConsole	0	0	0
##	montrealCryptXXX	0	0	0
##	montrealCryptoLocker	3	0	0
##	montrealCryptoTorLocker2015	0	0	0
##	montrealDMALocker	0	0	0
##	montrealDMALockerv3	0	0	0
##	montrealEDA2	0	0	0
##	montrealFlyper	0	0	0
##	montrealGlobe	0	0	0
##	montrealGlobeImposter	0	0	0
##	montrealGlobev3	0	0	0
##	montrealJigSaw	0	0	0
##	montrealNoobCrypt	1	0	0

##	montrealRazy	0	0	0
##	montrealSam	0	0	0
##	montrealSamSam	0	0	0
##	montrealVenusLocker	0	0	0
##	montrealWannaCry	0	0	0
##	montrealXLocker	0	0	0
##	montrealXLockerv5.0	0	0	0
##	montrealXTPLocker	0	0	0
##	paduaCryptoWall	6194	0	0
##	paduaJigsaw	0	0	0
##	paduaKeRanger	0	0	0
##	princetonCerber	0	0	0
##	princetonLocky	0	0	0
##	white	0	0	0
##		Reference		
##	Prediction	princetonCerber	princetonLocky	white
##	montrealAPT	0	0	0
##	montrealComradeCircle	0	0	0
##	montrealCryptConsole	0	0	0
##	montrealCryptXXX	0	0	0
##	montrealCryptoLocker	0	0	0
##	montrealCryptoTorLocker2015	0	0	0
##	montrealDMALocker	0	0	0
##	montrealDMALockerv3	0	0	0
##	montrealEDA2	0	0	0
##	montrealFlyper	0	0	0
##	montrealGlobe	0	0	0
##	montrealGlobeImposter	0	0	0
##	montrealGlobev3	0	0	0
##	montrealJigSaw	0	0	0
##	montrealNoobCrypt	0	0	0
##	montrealRazy	0	0	0
##	montrealSam	0	0	0
##	montrealSamSam	0	0	0
##	montrealVenusLocker	0	0	0
##	montrealWannaCry	0	0	0
##	montrealXLocker	0	0	0
##	montrealXLockerv5.0	0	0	0
##	montrealXTPLocker	0	0	0
##	paduaCryptoWall	0	1	0
##	paduaJigsaw	0	0	0
##	paduaKeRanger	0	0	0
##	princetonCerber	4612	5	0
##	princetonLocky	0	3228	0
##	white	0	0	0
##				
##	Overall Statistics			
##				
##		Accuracy : 0.999		
##		95% CI : (0.9985, 0.9994)		
##		No Information Rate : 0.3063		
##		P-Value [Acc > NIR] : < 2.2e-16		
##				
##		Kappa : 0.9987		
##				
##	McNemar's Test P-Value : NA			

```

##
## Statistics by Class:
##
##          Class: montrealAPT Class: montrealComradeCircle
## Sensitivity          NA          NA
## Specificity          1          1
## Pos Pred Value      NA          NA
## Neg Pred Value      NA          NA
## Prevalence          0          0
## Detection Rate       0          0
## Detection Prevalence 0          0
## Balanced Accuracy    NA          NA
##
##          Class: montrealCryptConsole Class: montrealCryptXXX
## Sensitivity          NA          0.99755
## Specificity          1          1.00000
## Pos Pred Value      NA          1.00000
## Neg Pred Value      NA          0.99984
## Prevalence          0          0.06038
## Detection Rate       0          0.06023
## Detection Prevalence 0          0.06023
## Balanced Accuracy    NA          0.99877
##
##          Class: montrealCryptoLocker
## Sensitivity          0.9994
## Specificity          0.9997
## Pos Pred Value      0.9989
## Neg Pred Value      0.9998
## Prevalence          0.2333
## Detection Rate       0.2331
## Detection Prevalence 0.2334
## Balanced Accuracy    0.9995
##
##          Class: montrealCryptoTorLocker2015
## Sensitivity          NA
## Specificity          1
## Pos Pred Value      NA
## Neg Pred Value      NA
## Prevalence          0
## Detection Rate       0
## Detection Prevalence 0
## Balanced Accuracy    NA
##
##          Class: montrealDMALocker Class: montrealDMALockerv3
## Sensitivity          NA          NA
## Specificity          1          1
## Pos Pred Value      NA          NA
## Neg Pred Value      NA          NA
## Prevalence          0          0
## Detection Rate       0          0
## Detection Prevalence 0          0
## Balanced Accuracy    NA          NA
##
##          Class: montrealEDA2 Class: montrealFlyper
## Sensitivity          NA          NA
## Specificity          1          1
## Pos Pred Value      NA          NA
## Neg Pred Value      NA          NA
## Prevalence          0          0
## Detection Rate       0          0
## Detection Prevalence 0          0

```

## Balanced Accuracy	NA	NA
##	Class: montrealGlobe	Class: montrealGlobeImposter
## Sensitivity	NA	NA
## Specificity	1	1
## Pos Pred Value	NA	NA
## Neg Pred Value	NA	NA
## Prevalence	0	0
## Detection Rate	0	0
## Detection Prevalence	0	0
## Balanced Accuracy	NA	NA
##	Class: montrealGlobev3	Class: montrealJigSaw
## Sensitivity	0.9285714	NA
## Specificity	1.0000000	1
## Pos Pred Value	1.0000000	NA
## Neg Pred Value	0.9999506	NA
## Prevalence	0.0006918	0
## Detection Rate	0.0006424	0
## Detection Prevalence	0.0006424	0
## Balanced Accuracy	0.9642857	NA
##	Class: montrealNoobCrypt	Class: montrealRazy
## Sensitivity	0.99573	NA
## Specificity	0.99995	1
## Pos Pred Value	0.99573	NA
## Neg Pred Value	0.99995	NA
## Prevalence	0.01156	0
## Detection Rate	0.01151	0
## Detection Prevalence	0.01156	0
## Balanced Accuracy	0.99784	NA
##	Class: montrealSam	Class: montrealSamSam
## Sensitivity	NA	NA
## Specificity	1	1
## Pos Pred Value	NA	NA
## Neg Pred Value	NA	NA
## Prevalence	0	0
## Detection Rate	0	0
## Detection Prevalence	0	0
## Balanced Accuracy	NA	NA
##	Class: montrealVenusLocker	Class: montrealWannaCry
## Sensitivity	NA	NA
## Specificity	1	1
## Pos Pred Value	NA	NA
## Neg Pred Value	NA	NA
## Prevalence	0	0
## Detection Rate	0	0
## Detection Prevalence	0	0
## Balanced Accuracy	NA	NA
##	Class: montrealXLocker	Class: montrealXLockerv5.0
## Sensitivity	NA	3.333e-01
## Specificity	1	1.000e+00
## Pos Pred Value	NA	1.000e+00
## Neg Pred Value	NA	9.999e-01
## Prevalence	0	1.482e-04
## Detection Rate	0	4.941e-05
## Detection Prevalence	0	4.941e-05
## Balanced Accuracy	NA	6.667e-01
##	Class: montrealXTPLocker	Class: paduaCryptoWall

## Sensitivity	NA	0.9994	
## Specificity	1	0.9996	
## Pos Pred Value	NA	0.9990	
## Neg Pred Value	NA	0.9997	
## Prevalence	0	0.3063	
## Detection Rate	0	0.3061	
## Detection Prevalence	0	0.3064	
## Balanced Accuracy	NA	0.9995	
##	Class: paduaJigsaw	Class: paduaKeRanger	
## Sensitivity	NA	NA	
## Specificity	1	1	
## Pos Pred Value	NA	NA	
## Neg Pred Value	NA	NA	
## Prevalence	0	0	
## Detection Rate	0	0	
## Detection Prevalence	0	0	
## Balanced Accuracy	NA	NA	
##	Class: princetonCerber	Class: princetonLocky	Class: white
## Sensitivity	1.0000	0.9981	NA
## Specificity	0.9996	0.9999	1
## Pos Pred Value	0.9985	0.9997	NA
## Neg Pred Value	1.0000	0.9996	NA
## Prevalence	0.2279	0.1598	0
## Detection Rate	0.2279	0.1595	0
## Detection Prevalence	0.2282	0.1596	0
## Balanced Accuracy	0.9998	0.9990	NA

4286.987 sec elapsed