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

HarvardX PH125.9x Final Capstone CYO Project

Kaylee Robert Tejeda

11/11/2021

Abstract

Ransomware is a persisent 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.

Contents

Introduction

Ransomware attacks are of interest to 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 learn 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 by a certain deadline to have the data decrypted or else it will all be deleted automatically.

The deeper legal and financial implications of ransomware attacks are inconsequential to the work in this report, as we are merely interested in being able to classify Bitcoin addresses by their connection to ransomware transactions. Many researchers are already tracking illicit activity (such as ransomware payments) around the Bitcoin blockchain in order to minimize financial losses. Daniel Goldsmith explains some of the reasons and methods of blockchain analysis at Chainalysis.com.[2] For example, consider a ransomware attack conducted towards an illegal darknet market site. The news of such an attack might not be announced at all to prevent loss of trust among its users. By analyzing the global 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, if so desired.

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.

Any given address on the Bitcoin network may appear many times, possibly 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, allowing for variables to be defined in a specific and meaningful way. For example, *speed* can be 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* may be an indicator of money laundering or "coin mixing", as typical payments only involve a limited number of addresses in a given 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 specified as such, the following six numerical features[2] are associated with a given address:

1) *Income* - the total amount of bitcoins 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 bitcoins 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 somewhat conceptually, by viewing the blockchain as a topological graph with nodes and edges. The rationale behind this approach is to facilitate quantification of specific transaction patterns. Akcora, et al.[3] give a thorough explanation 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 beyond that. Machine learning methods will be applied to the original data set from the same paper, and the new results will be compared to the original ones.

Data

The data set was found 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 for 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 shows the range of values and size of the sample. Some of the features, such as *weight* for example, already appear to be very skewed just from the quartiles. In the case of *weight*, the third quartile is only 0.8819482, meaning that 75% of the data is at or below this value for *weight* (with a minimum of $3.6064687 \times 10^{-94}$). The maximum *weight* value, however, is 1943.7487933. This means that nearly the entire range of values occurs in the upper 25%. In fact, many of the numerical features are similarly skewed, as you can see in the following summary.

Table 1: Summary of data set

year	day	length	weight	count	looped	neighbors	income
Min. :2011	Min. : 1.0	Min. : 0.00	Min. : 0.0000	Min. : 1.0	Min. : 0.0	Min. : 1.000	Min. :3.000e+07
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
Median :2014	Median :181.0	Median : 8.00	Median : 0.2500	Median : 1.0	Median : 0.0	Median : 2.000	Median :2.000e+08
Mean :2014	Mean :181.5	Mean : 45.01	Mean : 0.5455	Mean : 721.6	Mean : 238.5	Mean : 2.207	Mean :4.465e+09
3rd Qu.:2016	3rd Qu.:271.0	3rd Qu.:108.00	3rd Qu.: 0.8819	3rd Qu.: 56.0	3rd Qu.: 0.0	3rd Qu.: 2.000	3rd Qu.:9.940e+08
Max. :2018	Max. :365.0	Max. :144.00	Max. :1943.7488	Max. :14497.0	Max. :14496.0	Max. :12920.000	Max. :4.996e+13

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 numerical features defined previously (*income, neighbors, weight, length, count, loop*), two temporal features in the form of *year* and *day* (day of the year as an integer from 1 to 365), and a categorical feature called *label* that categorizes each address as either *white* (i.e. 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]. A listing of the first ten rows provides a sample of the features associated with each observation.

Table 2: First ten entries of data set

address	year	day	length	weight	count	looped	neighbors	income	label
111K8kZAEnJg245r2cM6y9zgJGHZtJPy6	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
1126eDRw2wqSkWosjTCre8cjjQW8sSeWH7	2016	322	72	0.0039063	1	0	2	71200000	princetonCerber
1129TSjKtx65E35GiUo4AYVeyo48twbrGX	2016	238	144	0.0728484	456	0	1	200000000	princetonLocky
112AmFATxzhuSpvtz1hfpa3Zrw3BG276pc	2016	96	144	0.0846140	2821	0	1	50000000	princetonLocky

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, whereas the entire network sees 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, producing a practical predictive model for categorizing ransomware addresses with an acceptable degree of accuracy. Increasing the precision, 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, look for insights in any patterns.
2. Binary separation using Self Organizing Maps.

3. Faster 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.

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 specifications.

- CPU: Intel i7-4600U @ 3.300GHz (4th Gen quad-core i7 `x86_64`)
- 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 80% to 90% of the data for training results in a training set that is too large to process given the hardware limitations. For reasons that are no longer relevant, 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. This produced partitions that were small enough to work with, so the partition size ratio was not further refined. This is a potential area for later optimization. A better partitioning scheme can surely be optimized further. Careful sampling was carried out to ensure that the ransomware groups were represented in each sample as much as possible.

Exploration and Visualization

The proportion of ransomware addresses in the original data set is 0.0141986. Thus, they make up less than 2% of all observations. This presents a challenge as the target observations are sparse within the data set, especially when we consider that this small percentage is then further divided into 28 subsets. In fact, some of the ransomware groups have only a single member, making categorization a dubious task.

The total number of `NA` or missing values in the original data set is 0. At least there are no missing values to worry about. The original data set is clean in that sense.

A listing of all ransomware families in the full original data set, plus a member count for each family is shown in Table 3. As can be seen, 11 of the 28 families have less than 10 addresses associated with them. We shall keep this in mind for later.

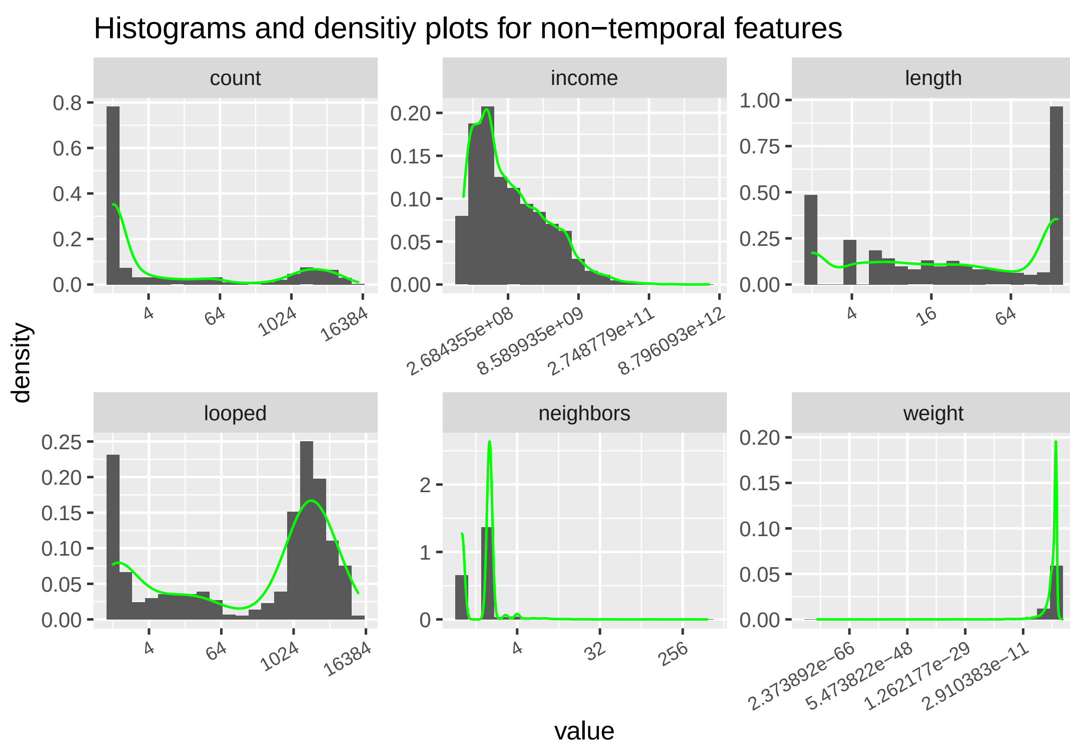
We can take a look at the overall distribution of the different features. The temporal features have been left out. Those plots are essentially flat due to the capped nature of the address collection, making each day of the year equally represented across the set. The skewed nature of the non-temporal features causes the plots to look better on a $\log_2$ scale x-axis.

Table 3: Ransomware families and membership counts

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

Table 4: Coefficients of Variation

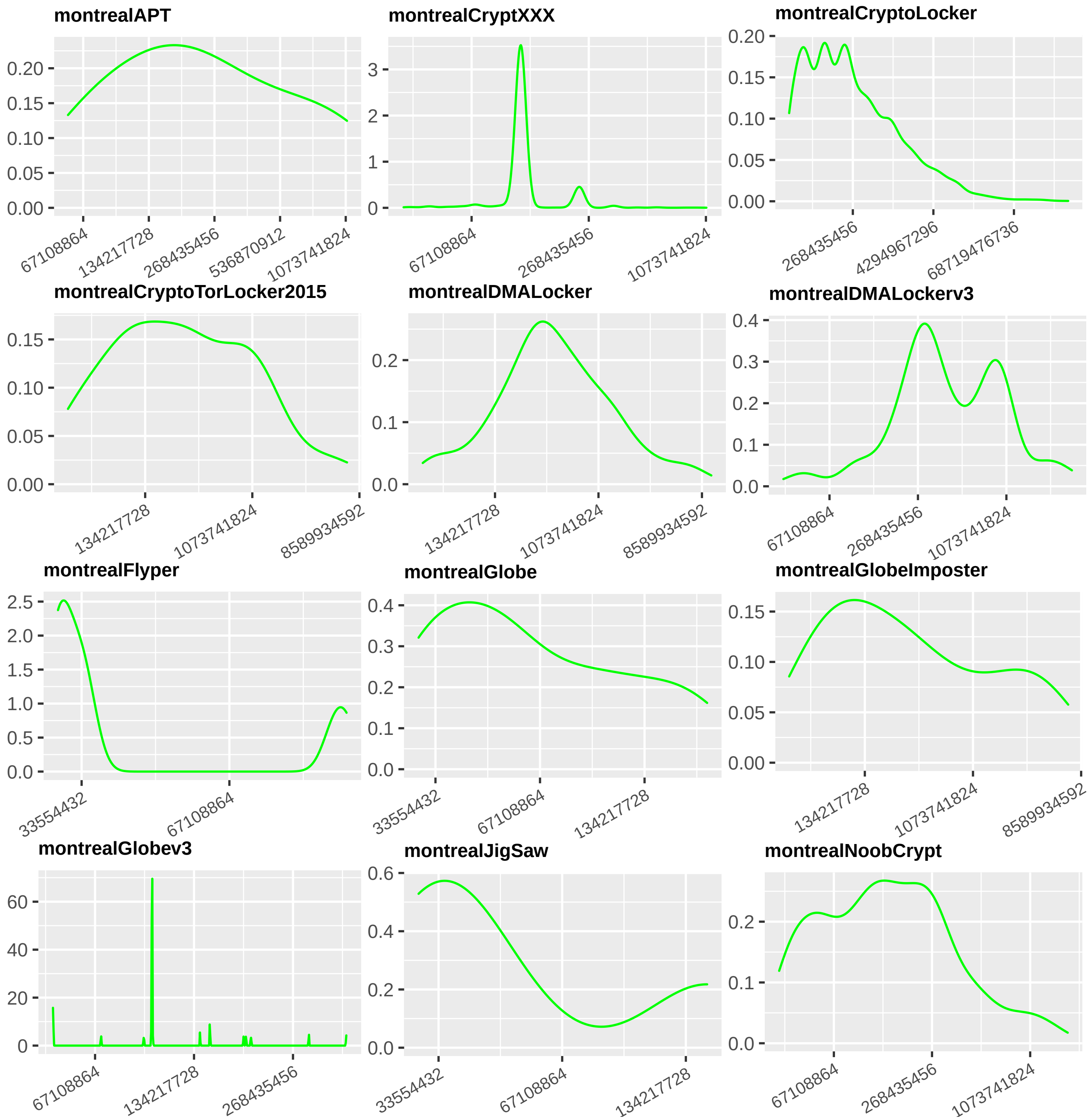
	CV		CV		CV
income	36	weight	6	count	2
neighbors	8	length	1	looped	4

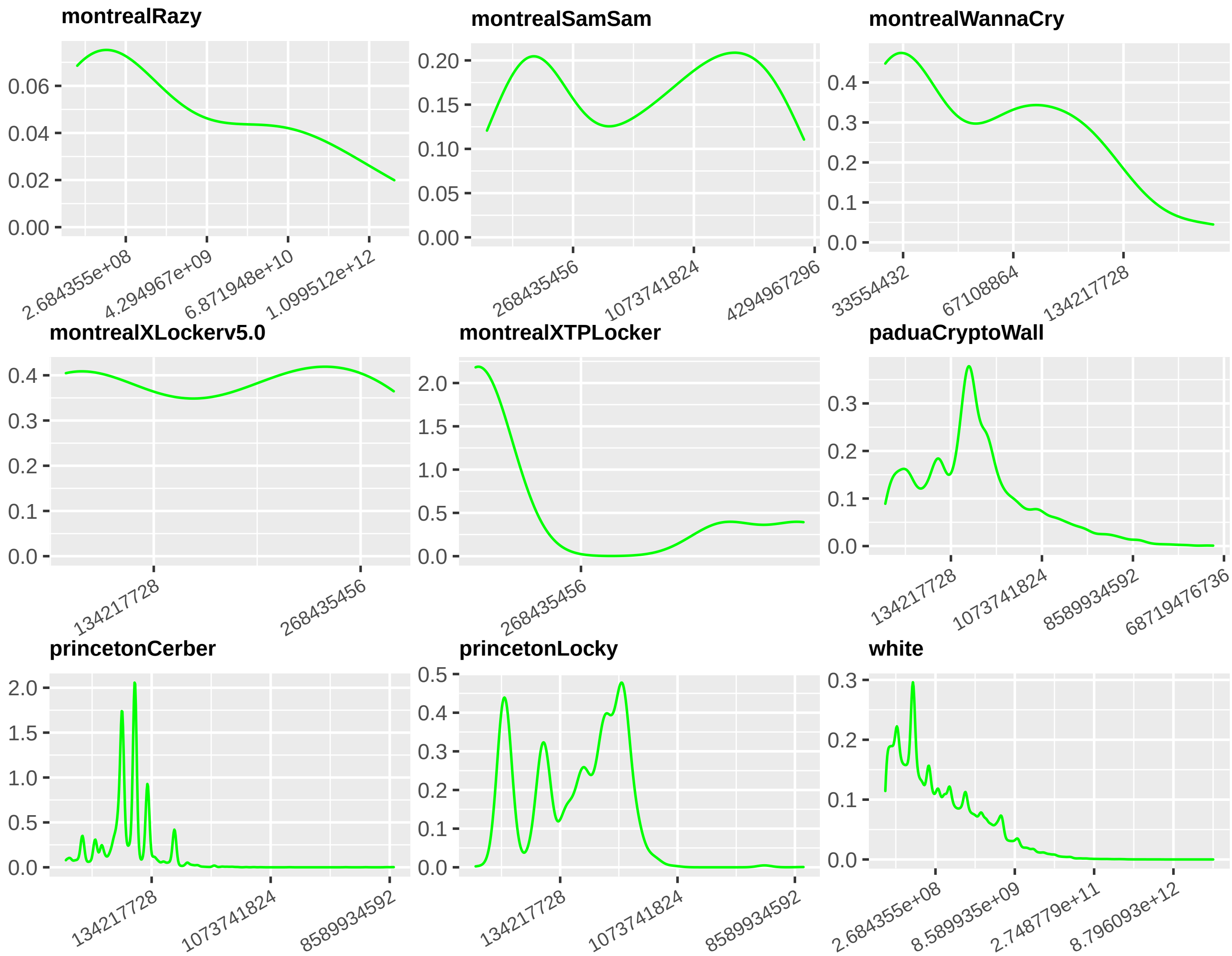


We can easily compare the relative spread of each feature by calculating the coefficient of variation for each column. Larger coefficients of variation indicate larger relative spread compared to other columns. A listing of the coefficients of variation for the non-temporal features is shown in Table 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.

Taking the feature with the highest variation income, we can take a look at the distribution for individual ransomware families to see if there is a similarity across families. This can be done for all the features, but we will focus on income in the interest of saving space and to avoid repetition and redundancy. The distribution plots for income show the most variation since it is the feature with the highest coefficient of variation, so it is a good one to focus on.





It appears that, although the income distribution for ransomware groups does differ from the distribution pattern for *white* addresses, it also varies from group to group. For this reason, this makes a good feature to use in the training of the models.

The percentage of wallets with less than one hundred bitcoins as their balance is 0.0147151. I have no idea why this is meaningful, but I can calculate it at least. **What else can I do here?** [A few more of these calculations might be good enough to wrap this section up, actually.]

Insights gained from exploration

After visually and numerically exploring the data, it becomes clear what the challenge is. Ransomware-related addresses are very sparse, comprising 1.4198595% of all addresses. This small percentage is also further classified into 28 groups. Perhaps the original paper was a overly ambitious in trying to categorize all the addresses into 29 categories, including the vastly prevalent *white* addresses. To simplify our approach, we will categorize the addresses in a binary way: as 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 have been shown 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] Considering all ransomware addresses as belonging to a single group may help to 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 (SOMs), supplied by the `kohonen` package.[11] The description at CRAN[7] was intriguing enough to merit further investigation.

Initially, the categorization of ransomware into the 29 different families (including *white*) was attempted using SOMs. This proved to be very resource intensive, requiring more time and RAM than was available. Although it did help to illuminate how SOMs are configured, the resource requirements of the algorithm became a deterrent. It was at this point that the SOMs were applied in a binary way, classifying all ransomware addresses as merely *black*, initially in an attempt to simply get the algorithm to run to completion without error. This reduced RAM usage to the point of being feasible on the available hardware.

Self Organizing Maps were not covered in the coursework at any point, therefore a familiar method was sought out to compare the results to. Random Forest was chosen and applied to the data set in a binary way, classifying every address as either *white* or *black*, ignoring the ransomware families. Surprisingly, not only did the Random Forest approach result in an acceptable model, it did so much quicker than expected, taking only a few minutes to produce results.

It was very tempting to leave it there and write up a comparison of the two approaches to the binary problem by classifying all ransomware related addresses as *black*. However, a nagging feeling that more could be done eventually inspired a second look at the categorical problem of grouping the ransomware addresses into the 28 known families. Given the high accuracy and precision of the binary Random Forest approach, the sparseness of the ransomware in the larger set has been eliminated completely, along with 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 does not seem to produce many false positive (if any), meaning it never seems to predict a truly white address as being black. Hence, by applying the Random Forest method first, we have effectively filtered out any possibility of false positives by correctly identifying a very large set of purely *white* addresses, which are then removed from the set. The best model used in the original paper by Akcora, et al. resulted in more false positives than true positives. This low precision rate is what made it impractical for real-world usage.[3]

All of these factors combined to inspire a two-part method: first to separate the addresses into *black* and *white* groups, and then to further classify the *black* addresses into ransomware families. We shall explore each of these steps separately.

Method Part 0: Binary SOMs

The first working model that ran to completion without exhausting computer resources ignored the ransomware family labels and instead used the two categories of *black* and *white*. The `kohonen` package provides algorithms for both supervised and unsupervised model building, using both Self Organizing Maps and Super Organizing Maps respectively.[11] A supervised approach was used since the data set includes information about the membership of ransomware families that can be used to train the model.

```
##############################################################################
## This is a first attempt using SOMs to model the data set as "black" and
## "white" addresses only.
##
## NOTE:  This is the most computationally heavy part of the paper and takes
## several hours to run to completion.  It is also completely optional, only
## used to compare with the better method. If, for some reason, you want to
## compile the report without this section, you can just comment it all out
## or remove it because nothing is needed from Method Part 0 for any of the
```

```r
## other methods.  In other words, it can be safely skipped if you are short on
## time or RAM.
################################################################################

# Start timer
tic("Binary SOMs", quiet = FALSE, func.tic = my.msg.tic)
```

```
## Starting Binary SOMs...
```

```r
# Keep only numeric columns, ignoring dates and looped.
som1_train_num <- train_set %>% select(length, weight, count, neighbors, income)

# SOM function can only work on matrices
som1_train_mat <- as.matrix(scale(som1_train_num))

# Switching to supervised SOMs
som1_test_num <- test_set %>% select(length, weight, count, neighbors, income)

# Note that when we rescale our testing data we need to scale it
# according to how we scaled our training data.
som1_test_mat <-
  as.matrix(scale(som1_test_num, center = attr(som1_train_mat, "scaled:center"),
                  scale = attr(som1_train_mat, "scaled:scale")))

# Binary outputs, black=ransomware, white=non-ransomware, train set
som1_train_bw <- train_set$bw %>% classvec2classmat()

# Same for test set
som1_test_bw <- test_set$bw %>% classvec2classmat()

# Create Data list for supervised SOM
som1_train_list <-
  list(independent = som1_train_mat, dependent = som1_train_bw)

##############################################################################
## Calculate idea grid size according to:
## https://www.researchgate.net/post/How-many-nodes-for-self-organizing-maps
##############################################################################

# Formulaic method 1
grid_size <- round(sqrt(5*sqrt(nrow(train_set))))
# Based on categorical number, method 2
#grid_size = ceiling(sqrt(length(unique(ransomware$bw))))

# Create SOM grid
som1_train_grid <-
  somgrid(xdim=grid_size, ydim=grid_size, topo="hexagonal", toroidal = TRUE)

## Now build the model.
som_model1 <- xyf(som1_train_mat, som1_train_bw,
                 grid = som1_train_grid,
                 rlen = 100,
                 mode="pbatch",
                 cores = n_cores,
                 keep.data = TRUE
)
```

```r
# Now test predictions

som1_test_list <- list(independent = som1_test_mat, dependent = som1_test_bw)

ransomware.prediction1 <- predict(som_model1, newdata = som1_test_list)


# Confusion matrix
som1_cm_bw <-
  confusionMatrix(ransomware.prediction1$prediction[[2]], test_set$bw)

# Now test predictions of validation set

# Switching to supervised SOMs
valid_num <- validation %>% select(length, weight, count, neighbors, income)

# Note that when we rescale our testing data we need to scale it
# according to how we scaled our training data.
valid_mat <-
  as.matrix(scale(valid_num, center = attr(som1_train_mat,  "scaled:center"),
                  scale = attr(som1_train_mat, "scaled:scale")))

valid_bw <- validation$bw

valid_list <- list(independent = valid_mat, dependent = valid_bw)

# Requires up to 16GB of RAM, skip if resources are limited
ransomware.prediction1.validation <- predict(som_model1, newdata = valid_list)

# Confusion matrix
cm_bw.validation <-
  confusionMatrix(ransomware.prediction1.validation$prediction[[2]],
                  validation$bw)

# End timer
toc(quiet = FALSE, func.toc = my.msg.toc, info = "Run Time")
```

```
## Run Time: Binary SOMs: 4039.724 seconds elapsed
```

After training the model, we obtain the confusion matrices for the test set and the validation set, separately. As you can see in Tables 5 and 6, the results are very good in both cases.

Table 5: Test set confusion matrix

	black	white
black	10353	0
white	0	718813

Table 6: Validation set confusion matrix

	black	white
black	20706	0
white	1	1437630

This is a very intensive method compared to what follows. It was left out of the final version of the script and has been included here only for model comparison and to track developmental evolution.

Method Part 1: Binary Random Forest

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.

```
###############################################################################
## This is a better attempt using Random Forest to model the data set as
## "black" and "white" addresses only.
###############################################################################

# Start timer
tic("Random Forest", quiet = FALSE, func.tic = my.msg.tic)
```

```
## Starting Random Forest...
```

```
# Cross Validation, ten fold
control <- trainControl(method="cv", number = 10)

# Control grid with variation on mtry
grid <- data.frame(mtry = c(2, 4, 6, 8, 10, 12))

# Run Cross Validation using control and grid set above
rf_model <- train(train_num, train_bw, method="rf",
                  trControl = control, tuneGrid=grid)

# Supervised fit of model using cross validated optimization
fit_rf <- randomForest(train_samp, train_bw,
                       minNode = rf_model$bestTune$mtry)

# Measure accuracy of model against test sample
y_hat_rf <- predict(fit_rf, test_samp)
cm_test <- confusionMatrix(y_hat_rf, test_bw)


# Measure accuracy of model against full ransomware set
ransomware_y_hat_rf <- predict(fit_rf, ransomware)
cm_ransomware <- confusionMatrix(ransomware_y_hat_rf, ransomware$bw)

# End timer
toc(quiet = FALSE, func.toc = my.msg.toc, info = "Run Time")
```

```
## Run Time: Random Forest: 123.951 seconds elapsed
```

The confusion matrix for the test set shows very good results, specifically in the areas of accuracy and precision. Although not as good as the SOM model used previously, the results are good enough to justify the time saved.

Tables 7 and 8 show the confusion matrices for the test set and the full set resulting from the Random Forest model, respectively. Note the absence of false negatives (upper right hand corners), meaning that no truly *black* addresses were predicted to be *white*. The converse is not necessarily true, a few truly *white* addresses get marked as *black* (lower left hand corners).

Tables 9 and 10 show the accuracy intervals for the test set and the full set, respectively.

Tables 11 and 12 show the overall results for each set.

As can be seen from these results, Random Forest is a much quicker way of removing most of the *white* addresses, while providing a comparable level of accuracy and precision. This method will be used in the final composite model to save time.

Table 7: Test set confusion matrix

	black	white
black	102	0
white	2	7188

Table 8: Full set confusion matrix

	black	white
black	40706	0
white	707	2875284

Table 9: Test set accuracy

	score
Accuracy	0.9997257
Kappa	0.9901522
AccuracyLower	0.9990096
AccuracyUpper	0.9999668
AccuracyNull	0.9857378
AccuracyPValue	0.0000000
McnemarPValue	0.4795001

Table 10: Full set accuracy

	score
Accuracy	0.9997576
Kappa	0.9912676
AccuracyLower	0.9997391
AccuracyUpper	0.9997751
AccuracyNull	0.9858014
AccuracyPValue	0.0000000
McnemarPValue	0.0000000

Table 11: Test set results

	score
Sensitivity	0.9807692
Specificity	1.0000000
Pos Pred Value	1.0000000
Neg Pred Value	0.9997218
Precision	1.0000000
Recall	0.9807692
F1	0.9902913
Prevalence	0.0142622
Detection Rate	0.0139879
Detection Prevalence	0.0139879
Balanced Accuracy	0.9903846

Table 12: Full set results

	score
Sensitivity	0.9829281
Specificity	1.0000000
Pos Pred Value	1.0000000
Neg Pred Value	0.9997542
Precision	1.0000000
Recall	0.9829281
F1	0.9913905
Prevalence	0.0141986
Detection Rate	0.0139562
Detection Prevalence	0.0139562
Balanced Accuracy	0.9914640

Method Part 2: Categorical SOMs

Now we train a new model after removing 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. The reduced set is then categorized using a supervised SOM method with the 28 ransomware families as the target classification groups.

```
# Start timer
tic("Categorical SOMs", quiet = FALSE, func.tic = my.msg.tic)
```

```
## Starting Categorical SOMs...
```

```
# Now use this prediction to reduce the original set to only "black" addresses
# First append the full set of predictions to the original set.
ransomware$prediction <- ransomware_y_hat_rf

# Filter out all the predicted "white" addresses,
# leaving only predicted "black" addresses
black_addresses <- ransomware %>% filter(prediction=="black")

# Split the reduced black-predictions into a training set and a test set @ 50%
```

```r
test_index <- createDataPartition(y = black_addresses$prediction,
                                  times = 1, p = .5, list = FALSE)

train_set <- black_addresses[-test_index,]
test_set <- black_addresses[test_index,]

# Keep only numeric columns, ignoring temporal variables.
train_num <- train_set %>%
  select(income, neighbors, weight, length, count, looped)

# SOM function can only work on matrices.
train_mat <- as.matrix(scale(train_num))

# Select non-temporal numerical features only
test_num <- test_set %>%
  select(income, neighbors, weight, length, count, looped)

# Testing data is scaled according to how we scaled our training data.
test_mat <- as.matrix(scale(test_num,
                            center = attr(train_mat, "scaled:center"),
                            scale = attr(train_mat, "scaled:scale")))

# Categorical labels for training set
train_label <- train_set$label %>% classvec2classmat()

# Same for test set
test_label <- test_set$label %>% classvec2classmat()

# Create data list for supervised SOM
train_list <- list(independent = train_mat, dependent = train_label)

############################################################################
## Calculate idea grid size according to:
## https://www.researchgate.net/post/How-many-nodes-for-self-organizing-maps
############################################################################

# Formulaic method 1, makes a larger graph in this case
grid_size <- round(sqrt(5*sqrt(nrow(train_set))))

# Based on categorical number, method 2, smaller graph with less cells
#grid_size = ceiling(sqrt(length(unique(ransomware$label))))

# Create SOM grid
train_grid <- somgrid(xdim=grid_size, ydim=grid_size,
                      topo="hexagonal", toroidal = TRUE)

## Now build the SOM model using the supervised method xyf()
som_model2 <- xyf(train_mat, train_label,
                  grid = train_grid,
                  rlen = 100,
                  mode="pbatch",
                  cores = n_cores,
                  keep.data = TRUE
)

# Now test predictions of test set, create data list for test set
```

```r
test_list <- list(independent = test_mat, dependent = test_label)

# Generate predictions
ransomware_group.prediction <- predict(som_model2, newdata = test_list)

# Confusion matrix
cm_labels <- confusionMatrix(ransomware_group.prediction$prediction[[2]],
                             test_set$label)

# End timer
toc(quiet = FALSE, func.toc = my.msg.toc, info = "Run Time")
```

```
## Run Time: Categorical SOMs: 59.68 seconds elapsed
```

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[8] forum. 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.

Table 13 shows the overall results of the final categorization.

Table 13: Overall categorization results

	score
Accuracy	0.9982792
Kappa	0.9977734
AccuracyLower	0.9976075
AccuracyUpper	0.9988011
AccuracyNull	0.3065047
AccuracyPValue	0.0000000
McnemarPValue	NaN

Table 14 shows the final results by class. It appears that many of the families with lower membership were not predicted at all. In fact, all the addresses classified as *black* by the Random Forest method have been grouped into only 7 families, a quarter of the actual 28. The relatively high accuracy rate would suggest that the larger families were predicted correctly, and that the smaller families were lumped in with the most similiar of the larger families. This could be an area for further refinement of the second SOM algorithm.

Table 14: Categorization results by class

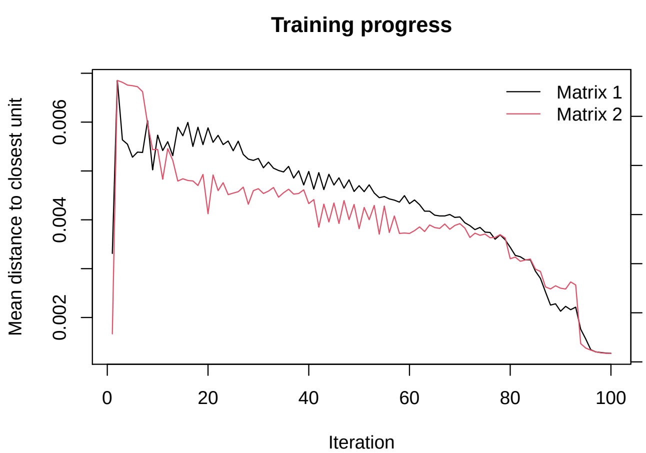
	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: montrealAPT	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealComradeCircle	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealCryptConsole	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealCryptXXX	0.9934319	1.0000000	1.0000000	0.9995818	1.0000000	0.9934319	0.9967051	0.0598850	0.0594916	0.0594916	0.9967159
Class: montrealCryptoLocker	0.9980599	0.9991720	0.9972001	0.9994266	0.9972001	0.9980599	0.9976298	0.2280840	0.2276415	0.2282806	0.9986160
Class: montrealCryptoTorLocker2015	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealDMALocker	0.9925373	0.9999505	0.9925373	0.9999505	0.9925373	0.9925373	0.9925373	0.0065883	0.0065392	0.0065883	0.9962439
Class: montrealDMALockerv3	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: montrealGlobeImposter	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA

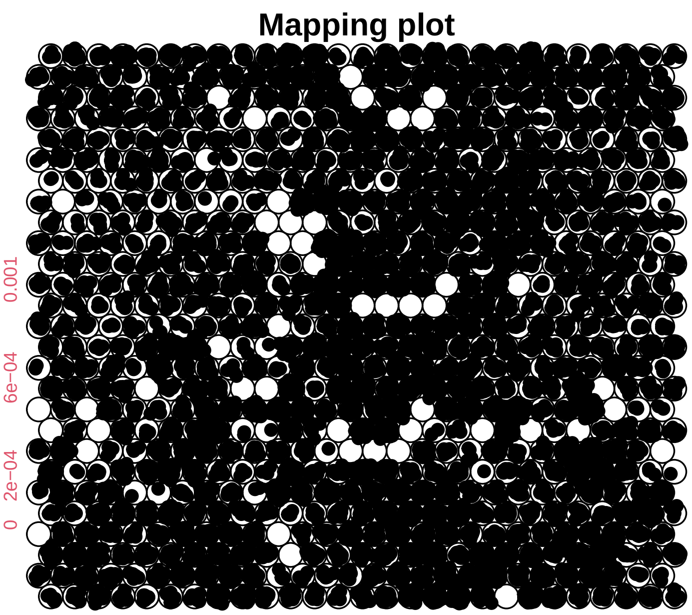
	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: montrealGlobev3	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealJigSaw	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealNoobCrypt	0.9912664	1.0000000	1.0000000	0.9999006	1.0000000	0.9912664	0.9956140	0.0112592	0.0111608	0.0111608	0.9956332
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	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: montrealXTPLocker	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA
Class: paduaCryptoWall	0.9987167	0.9991492	0.9980763	0.9994327	0.9980763	0.9987167	0.9983964	0.3065047	0.3061114	0.3067014	0.9989330
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	0.9997831	0.9995550	0.9984835	0.9999364	0.9984835	0.9997831	0.9991329	0.2266581	0.2266090	0.2269531	0.9996690
Class: princetonLocky	0.9981679	0.9998828	0.9993886	0.9996485	0.9993886	0.9981679	0.9987779	0.1610207	0.1607257	0.1608240	0.9990254
Class: white	NA	1.0000000	NA	NA	NA	NA	NA	0.0000000	0.0000000	0.0000000	NA

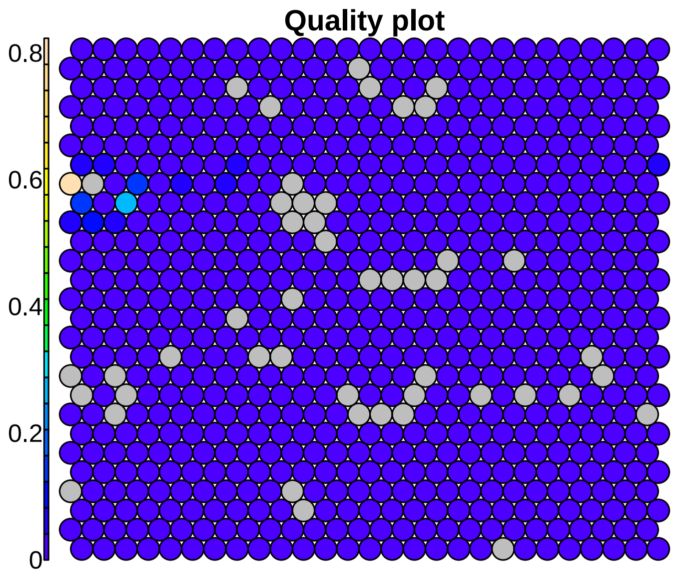
Map Visualizations and Clusterings

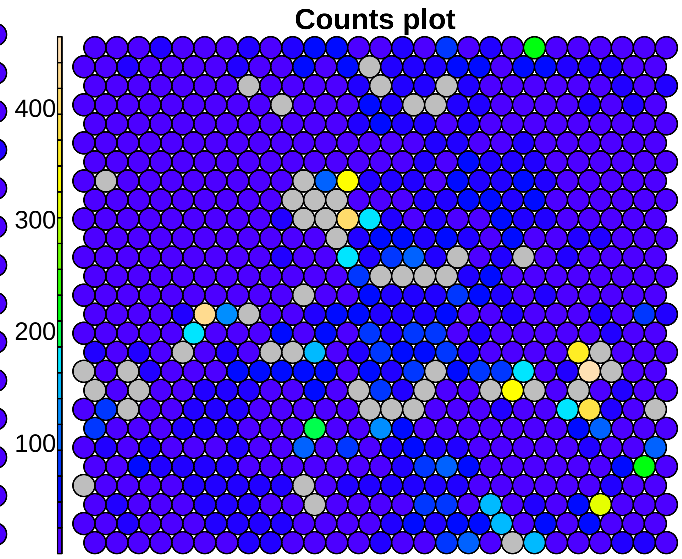
Toroidal neural node maps are used to generate the models, and can be visualized in a number of ways. The toroidal nature means that the top and bottom edges can be matched together, and the same with the left and right edges, forming a toroid, or donut shape.

The Training progress plot shows how many iterations the model had to undergo before the distances on the map stabilized. The Mapping plot is a visual representation of the individual observations and where they lie in the two-dimensional grid generated by the model. The Quality plot shows the average distance between addresses in each cell. The Counts plot gives a measure of the number of observations in each cell of the grid.

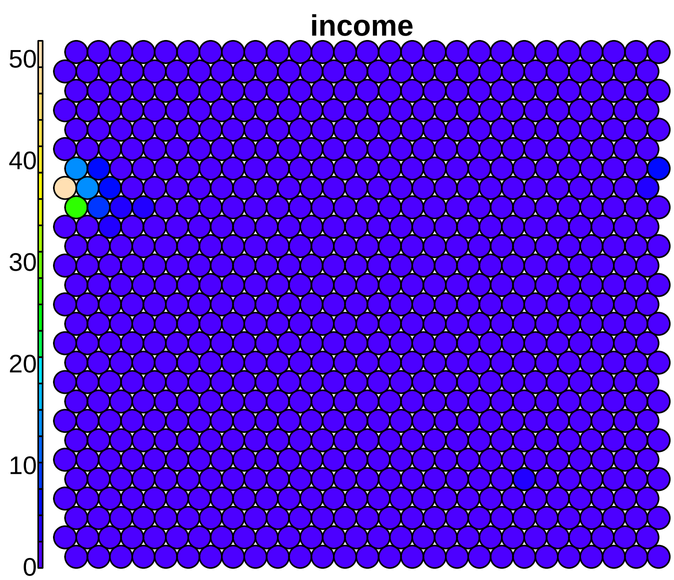


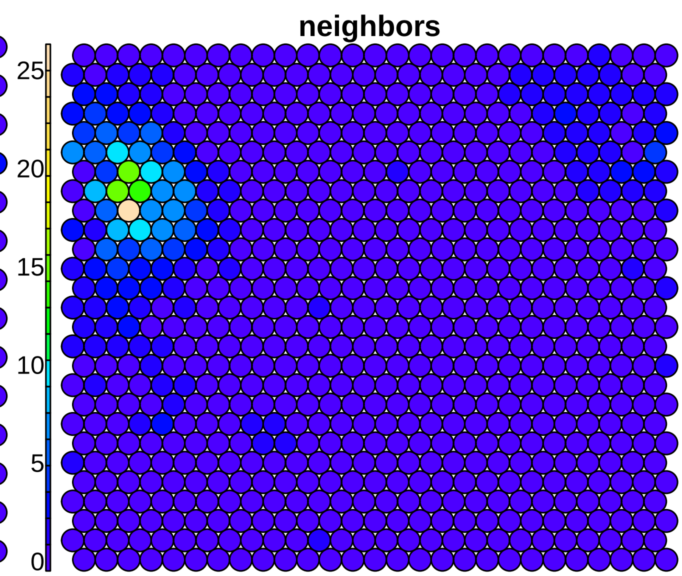


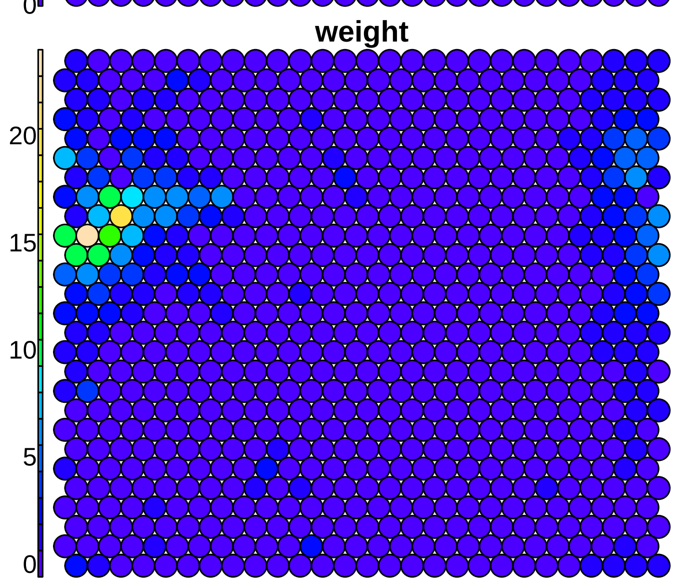


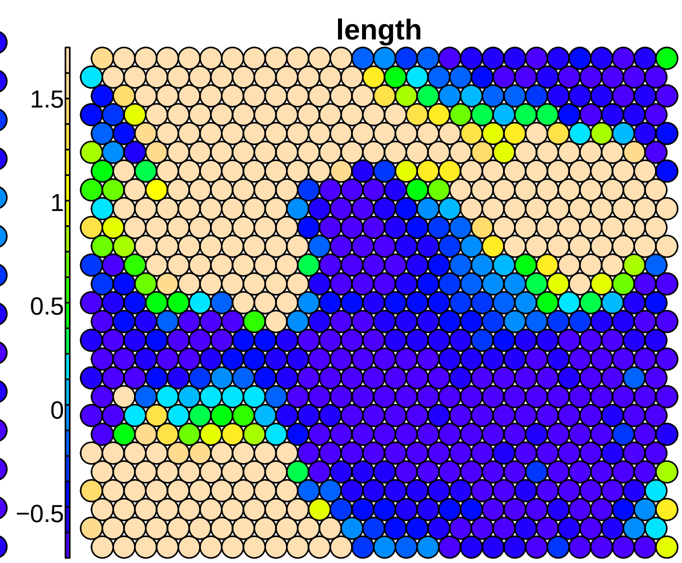


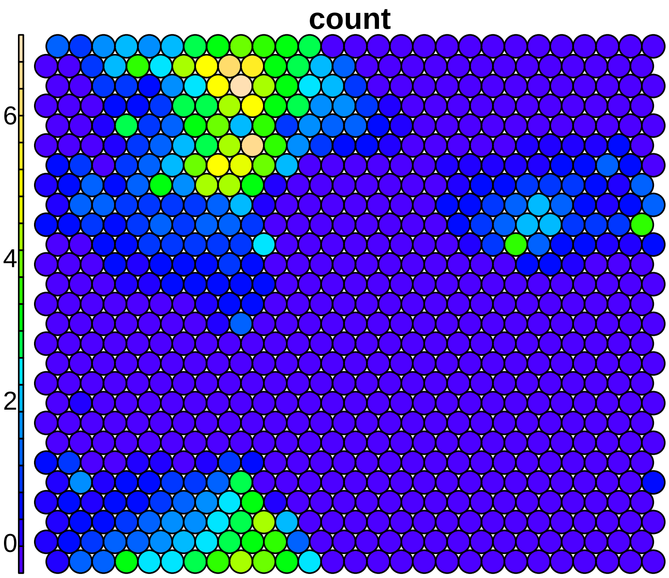
We can also look at heatmaps for each of the non-temporal features. This is where the grouping and the toroidal nature of the maps starts to become apparent. The color represents the average value for that feature in that cell.

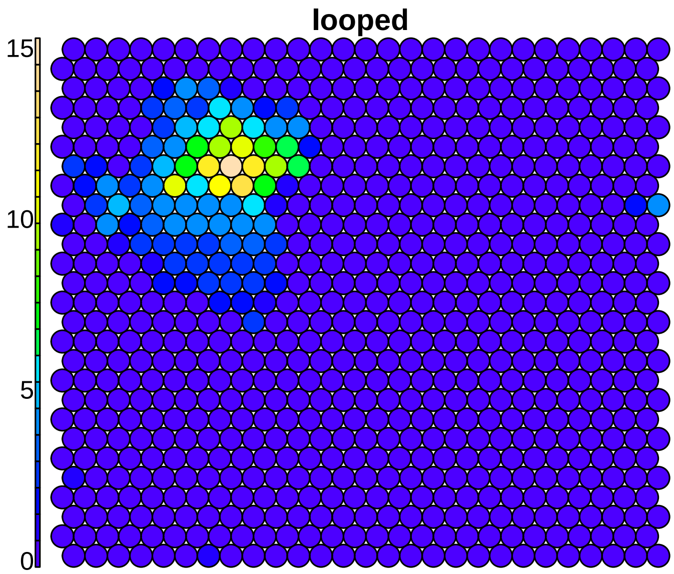




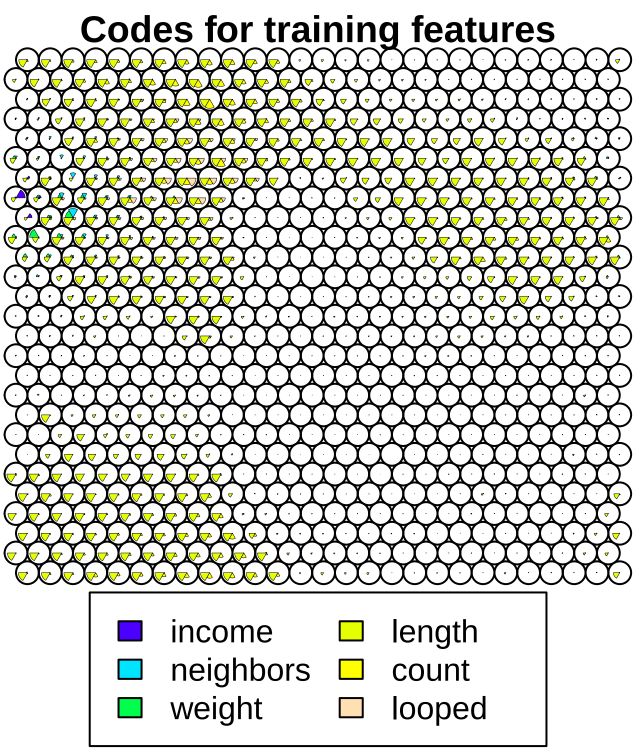


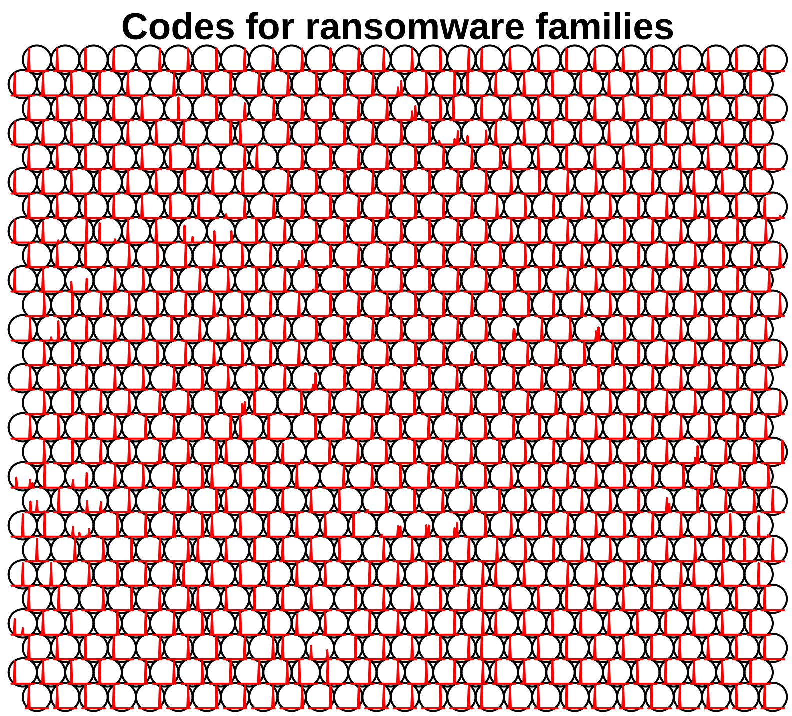






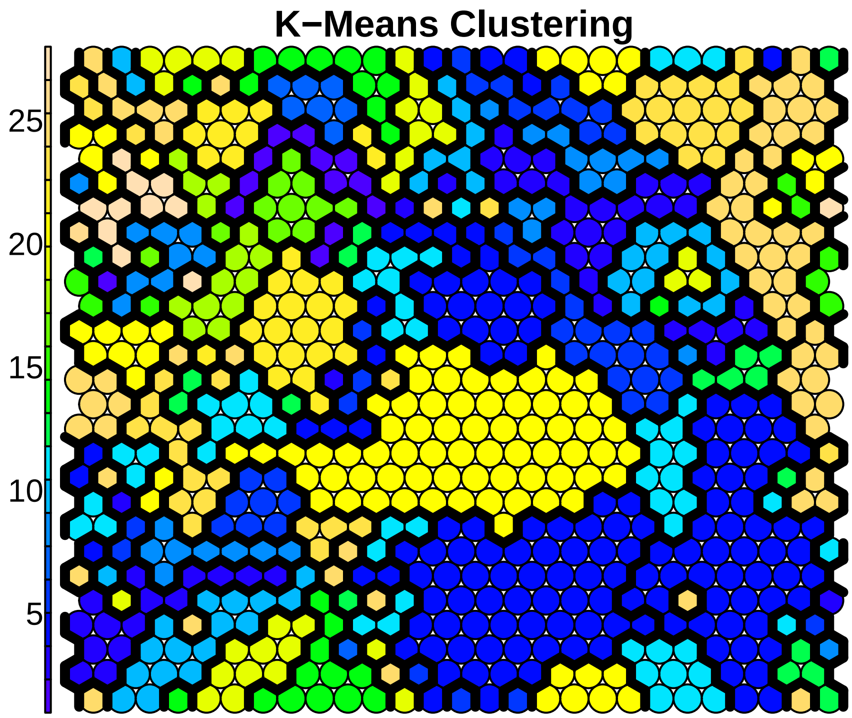
The code plots show how much of each feature is represented by each cell in the map. For large numbers of categories (such as with the ransomware families), the default behavior is to make a line plot instead of a segment plot, which leads to the density-like patterns to the right. In the left plot, the codebook vectors of the features used in the model are shown. These can be directly interpreted as an indication of how likely a given class is at a certain unit. The standard code plot creates these pie representations of the corresponding vectors for the grid cells. The radius of a wedge represents the magnitude in a particular dimension. From these, visual patterns start to emerge, as similar addresses are grouped together according to similarities of pie representations.





Clustering offers a nice way of visualizing the final SOM grid and the categorical boundaries that were formed by the model. Ideally, it is a visual representation of the final grouping. There are multiple algorithms for doing this.

K-means clustering is said to be better for smaller maps, while Hierarchical clustering is supposed to be better for larger maps. In this case, Hierarchical clustering does not converge on the right number of groups, while K-means requires the number of groups be specified ahead of time. Since we already know how many ransomware families are represented by the data set, K-means clustering is used to visualize the final categorization of the data on the map.



Results & Performance

Results

The first attempt to isolate ransomware from *white* addresses using SOMs resulted in a model with an accuracy of 0.999999314287438 and precision 1.

The second attempt to isolate ransomware from *white* addresses using Random Forest resulted in a model with an accuracy of 0.999757602520934 and precision 1.

Classifying the ransomware predicted by the second attempt into 28 ransomware families using SOMs resulted in a model with an overall accuracy of 0.998279168100693 and minimum nonzero precision of 0.992537313432836.

Performance

The script runs on the aforementioned hardware in 235 seconds 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 they are announced using even moderate computing resources. Just for comparison, the final script was also run on lower powered machines with the following specifications:

ASUS Eee PC 1025C

- CPU: Intel Atom N2600 @ 1.6GHz (64-bit Intel Atom quad-core x86)
- 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, although it does show that the script can be run on very modest hardware to completion.

Pine64 Quartz64 Model A

- CPU: Rockchip RK3566 SoC `aarch64` @1.8GHz (64-bit quad-core ARM)
- RAM: DDR4 8080MB (8 GB)

This is a single board computer / development board, which runs the same software as the others (ported to `aarch64`), except for Rstudio. It is of personal interest to benchmark a modern 64-bit ARM processor in addition to the two Intel CPUs. The script runs in about 860 seconds on this platform, nearly half of that for the Atom processor above. Still not fast enough to analyze each block in real time, but a significant improvement given the low power usage of such processors.

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), where TP = the number of true positives, and FP = the number of false positives] they achieved was only 0.1610. By comparison, although several of our predicted classes had zero or NA precision values due to low family membership in some cases, the **lowest** non-zero precision value is 0.992537313432836, with many well above that, equaling one in a few cases.

One might say that we are comparing apples to oranges by benchmarking single method model with a two-method stack. The two-model approach is justified and seems superior in this case, especially when measured in terms of total run time and having the benefit of avoiding false positives to a great degree.

Limitations

SOMs have many different parameters that seem easy to misconfigure, and usually require significantly more computing resources than less sophisticated algorithms. Perhaps a dual Random Forest approach would be better, if the loss of accuracy or precision was worth the time gain.

Future Work

I only scratched he surface of the SOM algorithm, which seems to have many implementations and parameters that could be investigated further and possibly optimized via cross-validation. For example, the grid size used to train the SOM was calculated using an algorithm based on the size of the training set, and while this performed better than a grid size based on the number of categories, it may not be ideal. Optimization around grid size could still be carried out. Hexagonal grids with toroidal topology were the only type used. Other types, such as square grids and non-toroidal topology are also possible, and may also be worth investigating.

A dual Random Forest approach could be used to first isolate the ransomware addresses as well as classify them might be quick enough to run in under ten minutes on all the hardware listed. Conversely, a dual SOM method could be created for maximum precision if the necessary computing resources were available.

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 further. The second SOM algorithm could be optimized to correctly predict more of the low-membership families.

Hierarchical clustering was attempted in addition to K-means clustering. The correct number of families was difficult to achieve, whereas it is a direct input of the K-means method. Another look at the clustering techniques might yield different results. Other clustering techniques exist, such as "Hierarchical K-Means"[13], which could be explored for even more clustering visualizations.

Conclusion

This 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 wondering how much harder it would be to perform the same task for ransomware that uses privacy-oriented coins. Certain cryptocurrency networks, such as Monero, utilize privacy features 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 these networks[9]. At the same time, 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

[10] KR Tejeda, Detecting Bitcoin Ransomware, https://git.disroot.org/shelldweller/ransomware

[11b] Wehrens R, Kruisselbrink J (2018). "Flexible Self-Organizing Maps in kohonen 3.0." *Journal of Statistical Software*, *87*(7), 1-18. doi: 10.18637/jss.v087.i07 (URL: https://doi.org/10.18637/jss.v087.i07).

[11a] Wehrens R, Buydens LMC (2007). "Self- and Super-Organizing Maps in R: The kohonen Package." *Journal of Statistical Software*, *21*(5), 1-19. doi: 10.18637/jss.v021.i05 (URL: https://doi.org/10.18637/jss.v021.i05).

[12] Difference between K means and Hierarchical Clustering (Jul 07, 2021) https://www.geeksforgeeks.org/difference-between-k-means-and-hierarchical-clustering/

[13] Hierarchical K-Means Clustering: Optimize Clusters (Oct 15 2021) https://www.datanovia.com/en/lessons/hierarchical-k-means-clustering-optimize-clusters/

Appendix:

Categorical SOM 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             1210
##   montrealCryptoLocker                             0                4
##   montrealCryptoTorLocker2015                      0                0
##   montrealDMALocker                                0                1
##   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                 1
##   white                                        0                 0
##                             Reference
## Prediction                   montrealCryptoLocker montrealCryptoTorLocker2015
##   montrealAPT                                  0                           0
##   montrealComradeCircle                        0                           0
##   montrealCryptConsole                         0                           0
##   montrealCryptXXX                             0                           0
##   montrealCryptoLocker                      4630                           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                              9                           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                      1                   0
##   montrealCryptoTorLocker2015               0                   0
##   montrealDMALocker                       133                   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 montrealGlobe
##   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                    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               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                    montrealJigSaw montrealNoobCrypt montrealRazy
##   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               227            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                 2            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                            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                     paduaCryptoWall paduaJigsaw paduaKeRanger
##   montrealAPT                                0           0             0
##   montrealComradeCircle                      0           0             0
##   montrealCryptConsole                       0           0             0
##   montrealCryptXXX                           0           0             0
##   montrealCryptoLocker                       8           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                    6226           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                    4609              5     0
##   princetonLocky                        1           3269     0
##   white                                 0              0     0
##
## Overall Statistics
##
##                Accuracy : 0.9983
##                  95% CI : (0.9976, 0.9988)
##     No Information Rate : 0.3065
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9978
##
##  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                 0.99343
## Specificity                                    1                 1.00000
## Pos Pred Value                                NA                 1.00000
## Neg Pred Value                                NA                 0.99958
## Prevalence                                     0                 0.05988
## Detection Rate                                 0                 0.05949
## Detection Prevalence                           0                 0.05949
## Balanced Accuracy                             NA                 0.99672
##                      Class: montrealCryptoLocker
## Sensitivity                               0.9981
## Specificity                               0.9992
## Pos Pred Value                            0.9972
## Neg Pred Value                            0.9994
## Prevalence                                0.2281
## Detection Rate                            0.2276
## Detection Prevalence                      0.2283
## Balanced Accuracy                         0.9986
##                      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                          0.992537                         NA
## Specificity                          0.999951                          1
## Pos Pred Value                       0.992537                         NA
## Neg Pred Value                       0.999951                         NA
## Prevalence                           0.006588                          0
## Detection Rate                       0.006539                          0
## Detection Prevalence                 0.006588                          0
## Balanced Accuracy                    0.996244                         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                              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: montrealNoobCrypt Class: montrealRazy
## Sensitivity                           0.99127                  NA
## Specificity                           1.00000                   1
## Pos Pred Value                        1.00000                  NA
## Neg Pred Value                        0.99990                  NA
## Prevalence                            0.01126                   0
## Detection Rate                        0.01116                   0
## Detection Prevalence                  0.01116                   0
## Balanced Accuracy                     0.99563                  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                         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: montrealXTPLocker Class: paduaCryptoWall
```

```
## Sensitivity                                  NA               0.9987
## Specificity                                   1               0.9991
## Pos Pred Value                               NA               0.9981
## Neg Pred Value                               NA               0.9994
## Prevalence                                    0               0.3065
## Detection Rate                                0               0.3061
## Detection Prevalence                          0               0.3067
## Balanced Accuracy                            NA               0.9989
##                      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                          0.9998                0.9982           NA
## Specificity                          0.9996                0.9999            1
## Pos Pred Value                       0.9985                0.9994           NA
## Neg Pred Value                       0.9999                0.9996           NA
## Prevalence                           0.2267                0.1610            0
## Detection Rate                       0.2266                0.1607            0
## Detection Prevalence                 0.2270                0.1608            0
## Balanced Accuracy                    0.9997                0.9990           NA
```