6.857 Project

Carson Smith, Berke Saat, Ignacio Parada

May 21, 2021

Contents

1 Abstract

This paper aims to use unsupervised machine learning to classify encrypted data into different categories. We chose encryption schemes that are normally applied by generating a random secret key for every text received. We want to prove that unless there is total randomness for each message in these encryption schemes, machine learning classifiers such as the one we implemented, can take advantage of that and recognize the patterns of the deterministic function on small subsets of the data. To test the success of our model on various encrypted data, the authors choose to use a combination of three distinct data sets; passwords, usernames, and product IDs. It proposes the grouping of these datasets through use of Self Organizing Maps (SOM) to gain information about the data while completely encrypted. While this paper's proposed algorithm does not attempt to decrypt any of the observed messages, it aims to gain information about the nature of the message itself. Through testing, the algorithm was found to classify the data extremely well on deterministic models, such as the basic Caesar cipher to a non-randomized version of ElGamal. However, as the approach uses frequency analysis as a main input to group, the more randomness and complexity that is added, the worse the approach performs, which shows the importance of randomness of each key while encrypting data.

2 Introduction

Since the onset of advanced Machine Learning models, they have been continually tested on encrypted data. Once successful, this allowed the privacy of the data, while still being able to create new functionality. One of the most popular examples of this practice is through federated machine learning by Google and Apple. Through this, these companies were able to take encrypted data from each phone in their system, run machine learning models on the encrypted data, and then send back the encrypted data to the phones which decrypt and use the results. The most common application for this is the suggestive keyboard, where phones can autocomplete words and sentences.

However, throughout all the applications of machine learning on encrypted data, almost

all of them are designed to provide further privacy to users while still increasing functionality, in a benign fashion. This allows us to still pose the question; since it has been proven that machine learning can be effective on encrypted data, is it possible to, with little to no information on the encryption methods, gain information on the data that could be harmful to the security of the system.

This paper aims to set up a series of assumptions and a real world scenario in which this method can be used to give an adversary the advantage. It will also describe the rational behind the main components, including the data and Machine Learning algorithm used.

2.1 Related Work

Extensive research has been done on how to cluster encrypted data.

Bost et al. [2] implemented three classification protocols that work without having to desencrypt data, these were: hyperplane decision, Naive Bayes and decision trees. On top of that, these protocols could be used in conjunction under the AdaBoost meta-algorithm. These protocols were constructed using a library that can be used to build other classifiers besides the already mentioned.

Hesamifard et al. [5] developed techniques that allow deep neural networks to run over encrypted data in order to classify information. They designed, trained and lastly implemented convolutional neural networks that work over encrypted data.

Li et a.l [6] propose a Naive Bayes classifier that works with multiple data sources and preserves data privacy. The scheme trains a classifier using a dataset that is provided by different data owners, without needing a trusted intermediary. The training will not break the privacy of each owner.

Baldimtsi et al. [1] propose a framework in which a user can upload his data to a server (for example for outsourcing) and later ask the server to transform and sort the result. In this case the privacy is obtained by using a secure coprocessor with a minimal amount of computational and memory resources. Both the server and coprocessor are assumed to be honest.

Wang et al. [9] propose a KNN which can search over encrypted data, mostly thinking in the use case of search in untrusted clouds.

Al-Rubaie et al. [8] do a review of the different ways in which machine learning systems work and collect and process data, with a special focus on privacy.

It is important to notice that most of these papers focus on keeping data private, but assume that all parties are well-behaved and no attacker is present. This is why the writers propose to use similar methods, but with malicious intent.

2.2 Adversary Scheme

To provide a real world scenario in which this approach would be beneficial, we define three actors. Alice, a benevolent defence contractor, Bob, a malicious defence contractor that is in direct competition with Alice, and Craig, a supplier who works directly with Alice and Bob to fill orders for their projects.

To start, we assume that there are three main categories between (Bob,Alice) and Craig; Order Requests, Messages, and Login Requests. Thus, we offer the following adversarial schemes. For a few months, Bob passively watches the encrypted communication from Alice to Craig, collecting all of the communications as a dataset. At the same time, Bob will send one communication, of each category type, to Craig and view the encrypted version of it.

Once Bob has collected the dataset, he will run all of the data through the unsupervised approach defined in this paper, resulting in a series of nodes with one type of encrypted messages belonging to each of them. However, given that this method is unsupervised, Bob does not know which node belongs to which type of communication as of yet. To determine this, Bob takes the labeled data that he has collected between himself and Craig and plots it alongside the other data. From this, Bob can determine with some level of certainty the nature of each node.

Moving forward, Bob can now passively observe the communication between Alice and Craig and, by plotting each encrypted method and discovering the closest node, can determine the nature of the communication between them. This can potentially give Bob and advantage as he can, for example, then determine when Alice is ordering a lot of supplies from Craig.

2.3 Data

Three different categories of data were used: usernames, passwords and product IDs, each of them with 500 items. In order to run an experiment that was comparable to an actual scenario the data sets used where as real as possible.

- **Products**: data set that contains real amazon product IDs from January 2020.[7]
- **Usernames**: data set that contains the username of any reddit account that left at least one comment. It is up to date until 2017. [3]
- **Passwords**: since a list of real passwords can not be used for obvious reasons, a subset from the rockyou list of common passwords found in Kali Linux was used.[4]

2.4 Unsupervised Approach

This paper revolves around the idea that an unsupervised machine learning approach can provide us information on encrypted data without knowing which encryption scheme is being used and without knowing a direct correlation between any encrypted message and plaintext message. Thus, for this to be accomplished, the unsupervised ML algorithm must be able to accomplish a series of things. Perform Frequency Analysis, look at both macro and micro patterns simultaneously, and disregard noise as best as possible.

Frequency Analysis: The model must be able to determine the most frequently used sequences within the encryption and, from there, isolate them to begin pattern recognition

Macro and Micro Patterns: Similar to frequency analysis, the text pre-processing, must convert the data into a series of vectors that simultaneously keeps the integrity of the encryption but also highlights minute elements of the encrypted text to be used in frequency analysis.

Disregard Noise: Given that a lot of the encrypted text will be about a specific username, password, product ID, etc, a model must be great at determining the important aspects of a given input and consider those much more than the other aspects.

3 Encryption Schemes

3.1 Caesar Cipher

We implemented a Caesar Cipher for the purpose of baseline testing. The characters consisted of lowercase letter, uppercase letters, and numbers, and therefore the cipher operated on mod 62. A new random shift value was generated for each text, and the shift was applied to every letter of the text.

The first time we ran our model was on the data encrypted with a Caesar Cipher. Once we got good results which we will elaborate on later in this paper, we decided to move forward with our model and run it on data encrypted with more complicated encryption schemes.

3.2 Vernam Cipher

Vernam Cipher also operated on mod 62. The process included generating a key of some large length, and a one time pad of the message with the generated key.

Every character had an order between $[0, 61]$. The key generation was sampling random values in that interval, finding its corresponding character, and eventually concatenating those characters to have one string as the key of our encryption.

Vernam is clearly more complex then Caesar. However, given the frequency of summing the same two letters over a large dataset, we expected there to be enough patterns for our model to recognize and thus work well on data encrypted via a Vernam Cipher.

3.3 ElGamal

In ElGamal, the main steps are key generation, encryption and decryption. In a real world scenario, there would be two sides of some communication, and they would have secret keys that the other does not know about. For example, if Alice and Bob are communicating, Alice would have access to a and Bob would have access to b. As the common secret key is g^{ab}, Bob receives the value $h = g^a$ and compute h^b to get that secret key, and Alice receives a public key $pk = g^b$ and computed $(pk)^a = g^{ab}$.

In order to imitate this behavior in our implementation, we followed the steps below:

Key Generation

- Choose a random "large" number q. For our implementation, we chose numbers between 10^{20} and 10^{40}.
- Choose a random number g in the interval $[2, q]$.
- Choose two different large numbers (a, b), so that $\text{GCD}(a, q) = \text{GCD}(b, q) = 1$. In a real life scenario, one side of the communication would only have access to a, and the other would only have access to b.
- Compute $h = g^a \mod q$.
- Output the list of values, $[g, a, b, q, h]$. The values g, q are used in both encryption and decryption functions, but b, h values are only used in encryption and the a value is only used in decryption.

Encryption

This function receives the inputs of (q, g, h, a) and the message.

- Compute the public key, $pk = g^b \mod q$.
- Compute the secret key, $sk = h^b \mod q = g^{ab} \mod q$.
- Compute the ciphertext by multiplying the order of each letter in the message by the secret key, g^{ab}.
- Output the ciphertext and the public key for decryption.
- Output the string concatenation of these numbers to be an input into our classifier. We are aiming to apply our model on textual data, as described in section 4. Thus this allows us to interpret these numbers in a way that we can run our model on them.

Decryption

This function receives the inputs of (q, g, b) and the ciphertext. It was implemented for the purpose of testing our own encryption scheme, and asserting that the input message to our encryption function is the output of our decryption function.

- Compute the secret key, $sk = (pk)^a = g^{ab}$.
- Divide each number in the ciphertext by the secret key, and convert the resulting number into a character based on the order.
- Concatenate all letters, output the text.

Randomness

As we tried to prove the importance of randomness in ElGamal's key generation, we added a new argument of randomness into our key generation function. The purpose of this was to control the x in the logic of **repeating the key x times**. We wanted to show that unless an encryption scheme operates on total randomness, and has some deterministic behavior among a subset of the data, a machine learning model such as a self organizing map could take advantage of that. As the function repeats the same set of (q, g, a, b) for multiples of data, we expected our model to recognize such patterns.

4 Self Organizing Maps (SOMs)

SOMs, or Self Organizing Maps, are the unsupervised machine learning approach this paper chooses to classify the encrypted data. SOMs are a clustering technique in which a matrix of nodes are used to topographically cover a dataset. Looking at image 4.1, we can see a brief representation of this process. The matrix of nodes, in black, is initialized in the left frame of the image. Throughout the training process, in the middle frame, a single point is analyzed, moving the nodes to cover the data. Finally, when training is complete, the nodes should complete a topographical map over the data where a point is identified by the node closest to it.

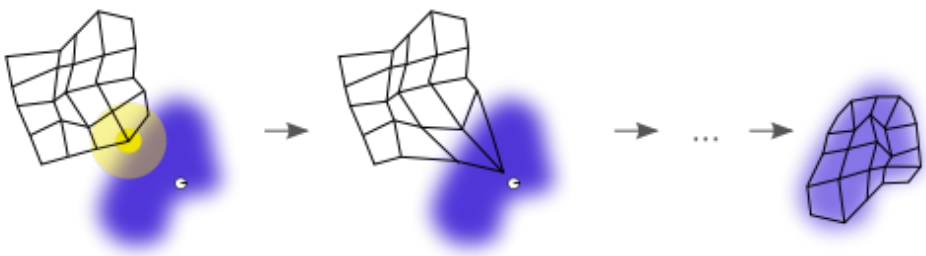

Figure 4.1: A visual demonstration behind the inner workings of SOMs.

This model was chosen because of its direct match to the specifications detailed in 2.3. Primarily, SOMs are great at ignoring both noise and outliers. In our example case with Alice, Bob, and Craig, both would pop up with great likelihood. Whether the noise is related to a specific product that Alice is ordered or outliers such as an entire communication category that Bob was not aware existed, SOMs do a great job at determining the main categories of data and determining which dimensions of a single entry are critical.

Additionally, SOMs are designed to topographically represent data or, in other words, perform dimension reduction to form a mapping. Thus, SOMs, in essence, use Macro patterns of the dataset to place nodes in their final positions and Micro patterns to influence the neighborhood of nodes around an individual point.

4.1 Pre-processing Text

To take full advantage of the SOMs, the data must be prepared in a way that is both an acceptable input format but still maintains integrity to the content of the text.

As defined above, SOMs take an input of a list of n-dimensional vectors. It is immediately apparent that the plain encrypted data can not be fed into the SOMs as is. Thus, this paper implements the following approach.

Step 1: For a specific encrypted message, c, split into $[c_1, c_2, c_3, ..., c_n]$ where c_i is a block of 5 characters of the encrypted text. (i.e. $c_1 = c[0:5]$)

Step 2: For all unique $c_i \in c \in C$, where C is the dataset of all encrypted data, assign c_i to a unique integer $a \in$(1,len(unique c_i)). Thus, creating a dictionary, map, that maps each unique 5-char text to an integer.

Step 3: Apply this mapping to each $c \in C$. Converting $[c_1, c_2, c_3, ..., c_n]$ to $s = [map[c_1], map[c_2], map[c_3], ..., map[c_n]]$, effectively turning c into a vector.

While this approach may not perfectly capture patterns of the data, it accomplishes the Macro and Micro requirement presented in 2.3. This allows SOMs to look at small blocks of the encrypted data while still keeping the overall flow of the cypher text whole.

5 Results

Plaintext

To begin the testing, the SOMs algorithm was run on the plaintext dataset to provide a baseline for performance on the encrypted data. As seen in figure 5.1 below, the SOMs had great success on this data and were able to find cluster placements that accurately split the data. After back calculation, similar to that described in 2.1, it became clear that node 0 was related to passwords, node 1 with usernames, and node 2 with product IDs. The SOMs were also able to perform with the following characteristics:

- Passwords: 91.6% accuracy, 100% precision
- Usernames: 91.2% accuracy, 100% precision
- Product IDs: 100% accuracy, 85.32% precision

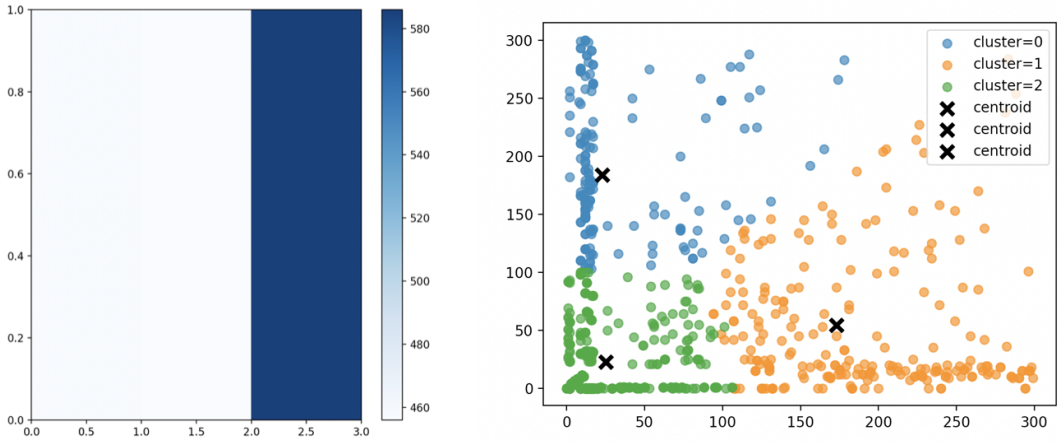

Figure 5.1: Data points per cluster, where each vertical third represents a cluster, (left) and Data points plotted with clusters (right) for plain text

Caesar Cipher

Next, this paper decided to try SOMs on a very simple encryption method, the Caesar Cipher. While this would make the message unreadable to the naked eye, SOMs would view the input as the exact same, as we are changing every single character by the same offset. Here we can see that the clusters have changed their respective category, although this is

expected given that the nodes are randomly placed at the start. Here, node 0 is related to Product IDs, node 1 with passwords, and node 3 with usernames. The characteristics were also very similar at:

- Passwords: 96.8% accuracy, 100% precision
- Usernames: 95% accuracy, 100% precision
- Product IDs: 100% accuracy, 92.42% precision

While there is a nominal increase in accuracy and precision across the different categories, this is the cause of the the inherent randomness in the model and, given enough training iterations would possess the same characteristics.

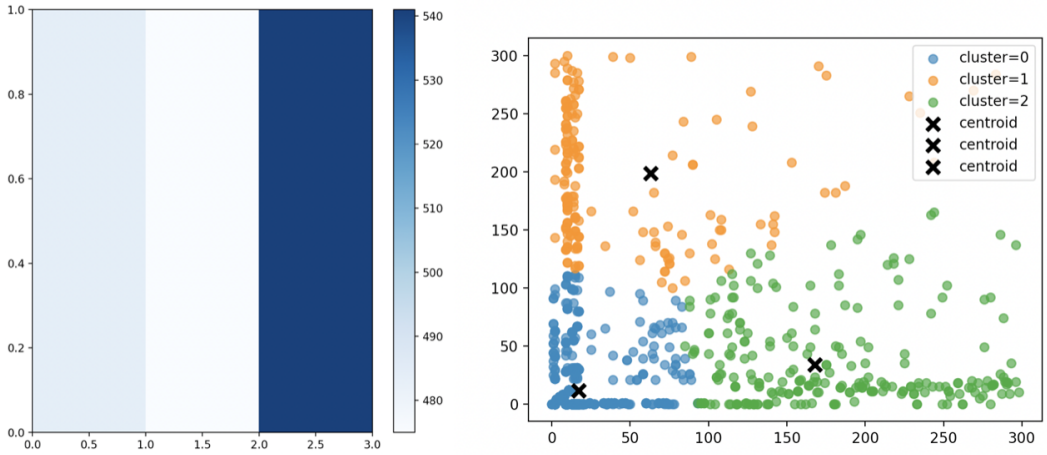

Figure 5.2: Data points per cluster, where each vertical third represents a cluster, (left) and Data points plotted with clusters (right) for Caesar Cipher

Vernam

With the Vernam Cipher with block size 7, we begin to see a slight difference in the data; however, this is most likely due to the fact that the block size is not divisible by 5, the size that we are breaking the text into, and thus can change the look of the data. However, SOMs was still able to successfully pick up on this new pattern, relating node 0 with usernames, node 1 with product ids, and node 2 with passwords. Thus, generating the following characteristics:

- Passwords: 96.8% accuracy, 100% precision

- Usernames: 94.6% accuracy, 100% precision

- Product IDs: 100% accuracy, 92.1% precision

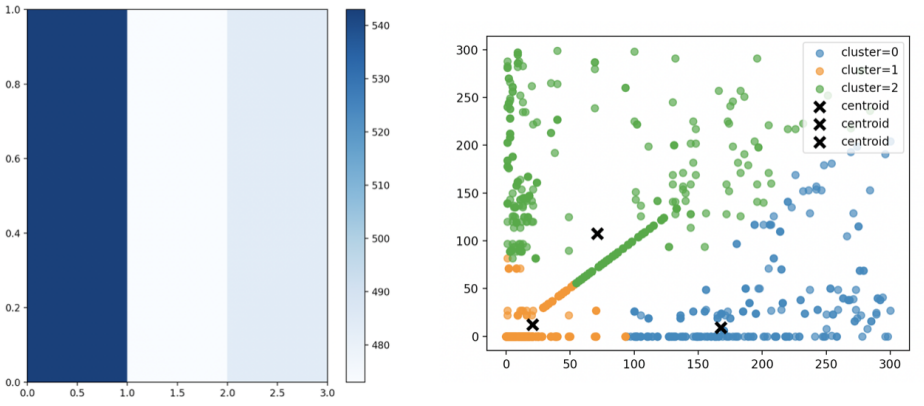

Figure 5.3: Data points per cluster, where each vertical third represents a cluster, (left) and Data points plotted with clusters (right) for Vernam Cipher with a block size of 7

The SOMs algorithm was also tested on a Vernam Cipher with a block size of 10 to determine if it would perform differently. However, as seen in 5.4 it did not. It performed with the following metrics, almost identical to the previous tests:

- Password: 94% accuracy, 100% precision

- Usernames: 98.8% accuracy, 100% precision

- Product IDs: 100% accuracy, 92.42% precision

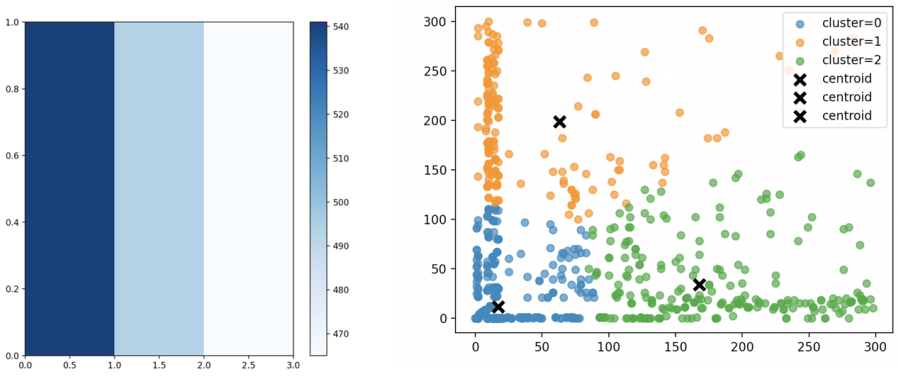

Figure 5.4: Data points per cluster, where each vertical third represents a cluster, (left) and Data points plotted with clusters (right) for Vernam Cipher with a block size of 10

ElGamal

Finally, we have the ElGamal results. As defined above, we are performing these experiments on 4 variations of ElGamal with increasing levels of randomness to determine how the randomness impacts this approaches success.

For the first version of ElGamal, with no randomness, we see the first ever perfect classification of the points in figure 5.5, where all three categories have 100% accuracy and 100% precision. This is most likely since each of the character in this encryption scheme are represented by a large integer. And thus, the differences between the data, and more-so the similarities between the same categories, can be discovered much easier.

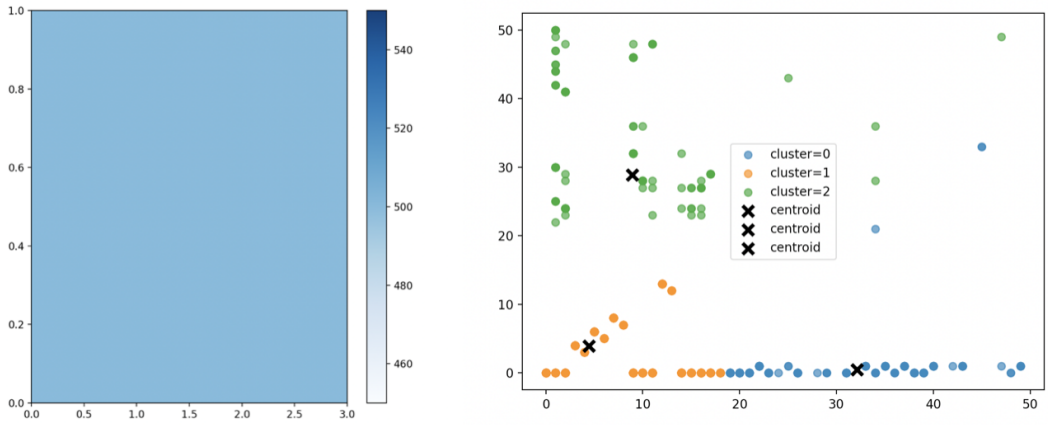

Figure 5.5: Data points per cluster, where each vertical third represents a cluster, (left) and Data points plotted with clusters (right) for non-randomized ElGamal

For the next level, randomizing every 250 entries, we see the data itself becomes a lot more visually random in the first two dimensions (in figure 5.6). This also becomes the first example where the algorithm is not able to strongly associate specific nodes with specific categories. The categories fall into nodes as below:

- Node 0: 500 Passwords, 5 Usernames
- Node 1: 250 Product IDs, 495 Usernames
- Node 2: 250 Product IDs

While these are not the results that have been expected in the previous cases, this paper now presents the base case, in relation to the real world example, to better compare these new cases. In the cases where bob simply sees the encryption and has to guess between the three categories, it is obvious to see that Bob will have, approximately, a 33% chance at guessing the category. However, extrapolating the relations above, we can see that, if an entry is in node 0, Bob will know, with 99% certainty that it is a password. If an entry is in node 1, he will know that it is a product ID with 34% probability and a Username with 66% probability. Finally, if it is in node 2, he knows it will be a product ID with 100% certainty. Thus averaging to a 99% accuracy for passwords and 66% accuracy for product IDs and Usernames, much better than that of the 33% chance he has purely guessing.

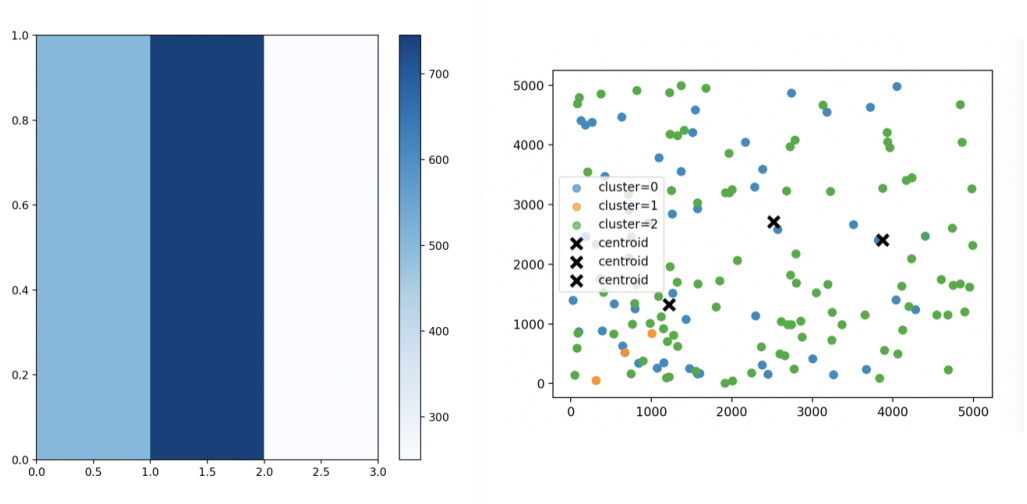

Figure 5.6: Data points per cluster, where each vertical third represents a cluster, (left) and Data points plotted with clusters (right) for ElGamal randomized every 250 entries

In the case where El Gamal is randomly generated every 10 entries, we see the same pattern from the above case repeated (in figure 5.7). This holds with the nodes being associated with the following number of entries:

- Node 0: 3 Passwords, 477 Product IDs, 15 Usernames
- Node 1: 1 Password, 22 Product IDs
- Node 2: 496 Passwords, 1 Product ID, 485 Usernames

Again, this averages to approximately 96% accuracy for Product IDs and 50% accuracy for both passwords and usernames. Which is, again, higher than the pure base case of Bob guessing and being correct 33% of the time.

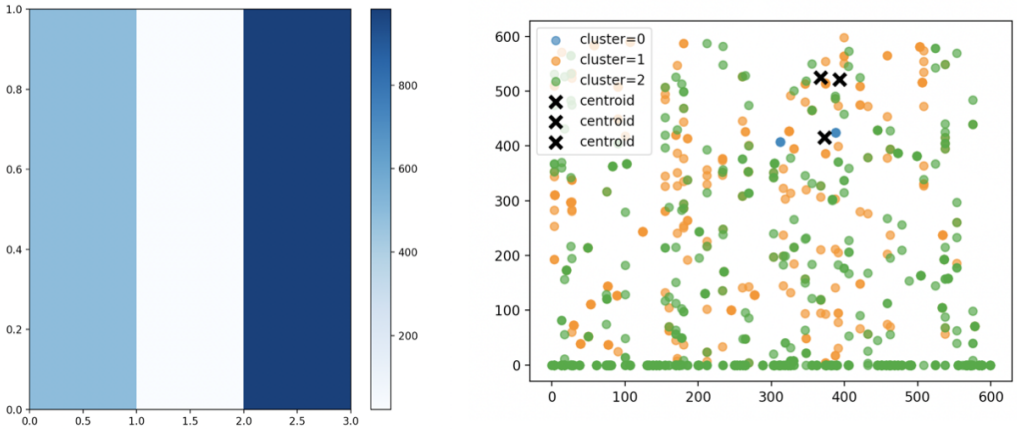

Figure 5.7: Data points per cluster, where each vertical third represents a cluster, (left) and Data points plotted with clusters (right) for ElGamal randomized every 10 entries

Finally, we look to an El Gamal version where the data is encrypted every time the encryption is run. From the image on the right, in Figure 5.8, it is obvious that this severely increases the randomness of the model and thus becomes harder for SOMs to notice any distinguishable pattern. However, very surprisingly, even with this much randomness, the model is able to determine some repeatable patterns from the data and the pattern from the previous cases is upheld. We have the following associations per node:

- Node 0: 192 Product IDs
- Node 1: 307 Product IDs, 2 Usernames
- Node 2: 500 Passwords, 1 Product ID, 498 Usernames

Here, even with this case, Bob would be able to expect an average of 99% accuracy for predicting when it is a product ID and 50% accuracy when predicting when it is a password or username.

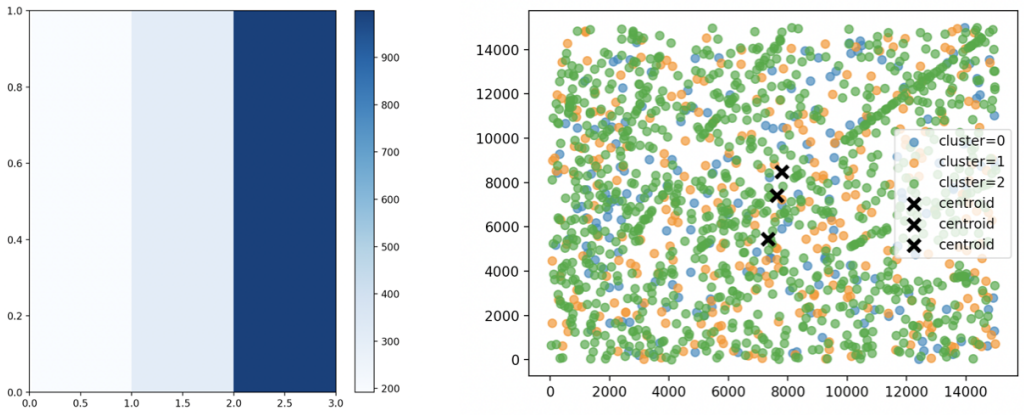

Figure 5.8: Data points per cluster, where each vertical third represents a cluster, (left) and Data points plotted with clusters (right) for ElGamal randomized every entry.

It is worth noting that, throughout most of the early results, we observe 100% precision in passwords and usernames but not for Product IDs. This is due to the extreme randomness that Product IDs have that is not as inherent in the other classes. Thus, if you have a really random password or username they will be grouped with the Product IDs; however, this will very rarely ever happen in reverse.

6 Conclusion

In this work we studied if information can be obtained from encrypted data using self organizing maps. By running the same ElGamal algorithm with varying levels of randomness, we showed the success rate of the classification on the encrypted data.

First, we showed our model work well with encryption schemes that are older, and less complex than the ones currently being used in modern systems. We can conclude here that this classifier works well on Caesar and Vernam ciphers, and gives the adversary a very good chance of guessing the category of the data being shared. The reason is that the frequency of an input letter outputting the same letter after encryption is fairly high in both cases. Therefore the model is able to follow such patterns with high frequencies, and classify them correctly.

However, with ElGamal, we see the model fail as the domain of the output, i.e. the interval of numbers that can be outputted is very large. The main point here is that total randomness is mandatory for a semantic encryption scheme like ElGamal. We imagined a scenario where to improve the performance, the randomness is decreased and the same set of keys is repeated for multiple messages. This may break the security of the overall system for various reasons, and in this paper, we have shown one concrete way of gaining information when the encryption scheme does not compute a brand new random key for every message.

As shown in the results section, the attack we have chosen have limitations. It will fail to gain information and break the security of a system as long as the system uses some pseudorandom function function at every encryption. Therefore, we do not see the attack we described in this paper have a large impact on the more modern encryption schemes which rely on pseudorandom functions, but we see this experiment as a way of proving the importance of using them in modern encryption schemes.

As further work, different deep learning models could be used in order to handle total randomness better, and maybe even show better classification even then which would break the security of the scheme.

References

[1] F Baldimtsi and O Ohrimenko. *Sorting and searching behind the curtain*. URL: https://link.springer.com/chapter/10.1007/978-3-662-47854-7_8.

[2] R Bost et al. *Machine learning classification over encrypted data*. URL: http://iot.stanford.edu/pubs/bost-learning-ndss15.pdf. NDSS, 2015 - iot.stanford.ed.

[3] ColinMorris. *Reddit Usernames*. URL: https://www.kaggle.com/colinmorris/reddit-usernames.

[4] *Common Password List (rockyou.txt)*. URL: https://www.kaggle.com/wjburns/common-password-list-rockyoutxt.

[5] E Hesamifard, H Takabi, and M Ghasemi. *Cryptodl: Deep neural networks over encrypted data*. URL: https://arxiv.org/abs/1711.05189.

[6] T Li et al. *Differentially private Naive Bayes learning over multiple data sources*. URL: https://www.sciencedirect.com/science/article/pii/S0020025518301415.

[7] PromptCloud. *Amazon Product Dataset 2020*. URL: https://www.kaggle.com/promptcloud/amazon-product-dataset-2020.

[8] M Al-Rubaie and JM Chang. *Privacy-preserving machine learning: Threats and solutions*. URL: https://ieeexplore.ieee.org/abstract/document/8677282/.

[9] B Wang, Y Hou, and M Li. *Practical and secure nearest neighbor search on encrypted large-scale data*. URL: https://ieeexplore.ieee.org/abstract/document/7524389/.