Department of Mathematics

Term Paper - Research Case Study

Self Organizing Maps for Fraud Detection

RCS Maths Group 2

Mustafa Saeed, Manoj Kumar, Manju Bala

Advisors

Prof. Dr. Volker Schulz

M.Sc Manuel Klar

March 15, 2022

Contents

1 Abstract

Purpose

Financial industries, such as banking, insurance, are often impacted by fraud, which is planned and intended activity aimed at achieving material gains. In this research case study, we have analyzed a bank's data set that consists of both fraudulent and non-fraudulent transactions to demonstrate a novel approach to profiling crime in the banking industry using self-organizing maps. We have made use of Cluster analysis using the self-organizing maps algorithm, with the support of Python programming language by using minisom library. While training self-organizing maps, a variety of factors were considered to ensure their accuracy.

Method

Self-organizing map is used for both training and testing.

Data

Data is collected by UCI Machine Learning Repository, it belongs to bank's customers, all attribute names and values have been masked to protect data privacy.

Results

The results of the analysis are estimates of fraudulent and non-fraud customers, and the accuracy of the model is also analyzed based on the effects of various training parameters and fluctuations in the values of the parameters during model training.

Conclusion

Having masked data impacted our understanding of the results that we got, but that's normal for any type of machine learning models. The amount of people who were predicted to be fraud is not small, which leaves extra room for improvements, still, the amount of frauds that were detected is relatively high.

2 Introduction

Almost all payment systems suffer from a high level of risk of fraud as a result of a large number of transactions, the complex relations between clients, and the increasing speed of data transmission. In order to manipulate risks, Payment systems need to develop and use mathematical models to determine suspicious/unstable situations, set up situations for their development, and consider the consequences of their realization.

Today, one of the most important and challenging issues for Payment Systems and its members is credit card fraud, the illegal use of credit cards by third parties. A fraudulent electronic transaction is already a serious problem, and it becomes even more important as the number of access points grows, especially if transactions on the Internet are fully electronic. Fraud detection and prevention methods are continually improving. But banks around the world lose millions of dollars each year. Visa International experts predict that certain frauds will grow up to 65 percent annually.

Credit card fraud is carried out in more than a few ways and is usually based on the withdrawal of fraudulent funds from the account of the cardholder who is the customer of the bank. Credit card fraud can be broadly divided into applications, "lost mail", stolen/lost cards, counterfeit cards, and "no cardholder" fraud. The number of different types of fraud is large enough, they are constantly changing, and new fraud methods are emerging as credit card protection improves. To date, banks that are members of PS have been able to prevent fraud through systematic measures such as limiting the number and volume of cardholder transactions, monitoring transactions in high-risk countries, and using various card verification methods. I was solving the problem. All banks need to take special steps to detect and prevent fraud in a timely manner. In international payment systems such as Visa International and MasterCard International, bankers need to take various measures to reduce the number of fraudulent operations on Payment Systems, We recommend switching from post-action type to proactive type to deal with fraudulent card operations.

In order for detection and prevention of fraud to be effective, banks should develop and use in their practice special fraud detection systems targeted to reveal among the flow of transactions and thus to prevent banks as well as their clients from fraudulent activities. Special rules should be developed for analysis, models, and methods that can describe fraudulent behavior, rules and methods of fraud prevention, and the generation of different decision alternatives in risky situations. Mathematical models and algorithms for classification and

pattern recognition problems could be considered as a basis for such systems.

In this paper, a model and algorithms for the detection of fraudulent operations in payment systems are proposed with the help of self-organizing maps which make use of cluster analysis to develop a model which can be used as a solution for detecting fraud.

3 Methods

3.1 Self Organizing maps

The Self Organizing Map is one of the foremost popular neural models. It belongs to the category of competitive learning. The SOM relies on unsupervised learning. We could, for instance, use the SOM for clustering membership of the input file. The SOM may be accustomed detect features inherent to the matter and thus has also been called Self Origination Feature Map(SOFM). The Self Organized Map was developed by professor Kohonen which is employed in many applications. The purpose of SOM is that it's providing an information visualization technique that helps to grasp high dimensional data by reducing the dimension of information to map. SOM also represents the clustering concept by grouping similar data.

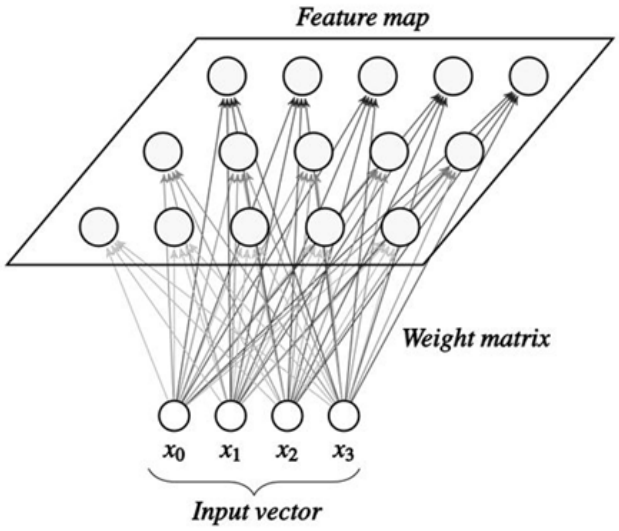

Figure 1: Self Organising Maps

3.1.1 Competitive Learning

In competitive learning, the weights related to prototypes compete with one another to find the Best Matching Unit(BMU) to the input vector. Only the weights related to the winning prototype are trained to be resembled in the SOM lattice. The weights of all other prototypes

change depending on how close they are to the BMU.

3.1.2 Training Algorithm

1. Estimate No. of Classes (No. of Output prototypes)
2. Set Weights randomly and normalize
3. Apply the normalized Input Vector X
4. Calculate Strength (i.e. Weighted Sum) of every prototype
5. Determine the prototype i with the best Response
6. Declare prototype i as the 'Winner' (i has the Weights most almost like X)
7. Train Weights of prototype i to make them, even more, the same as X

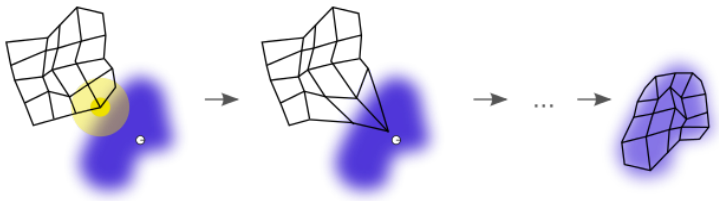

Figure 2: Competitive Learning to retain topology

During Training:

- Weight Vector of the Winner prototype is formed more up to Current Input Vector
- In other words, the Current Input Vector is Transferred to Winner

After Training:

- Winner prototype Carries the Input it Won (The weight vector of the winning prototype now retains the input pattern which it's been trained for)
- Any Successive Inputs almost like Previous select the prototype that has the smallest distance to the input vector as a winner to resemble it.

3.1.3 K-Mean Clustering Technique

K-Means clustering aims to partition n observations into k clusters within which each observation belongs to the cluster with the closest mean, serving as a prototype of the cluster.

k-means clustering tries to group similar forms of items in variety of clusters. It finds the similarity between the things and groups them into the clusters.

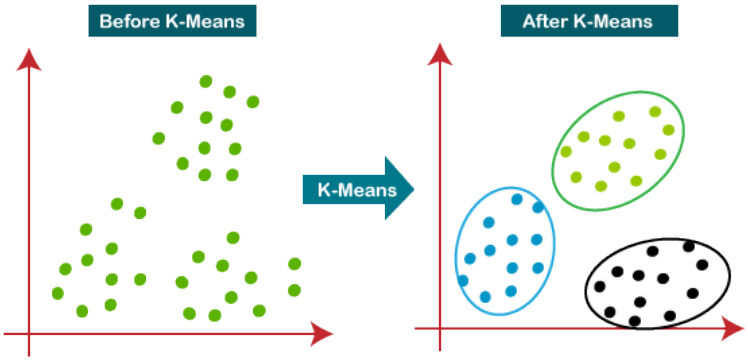

Figure 3: K-Means clustering

How k-Mean Cluster work

The k-Means clustering algorithm tries and split a given anonymous data set (a set of containing information on class identity into a set number (k) of the cluster. Initially, k number of the so-called centroid is chosen. A centroid may be a information (imaginary or real) at the middle of the cluster.

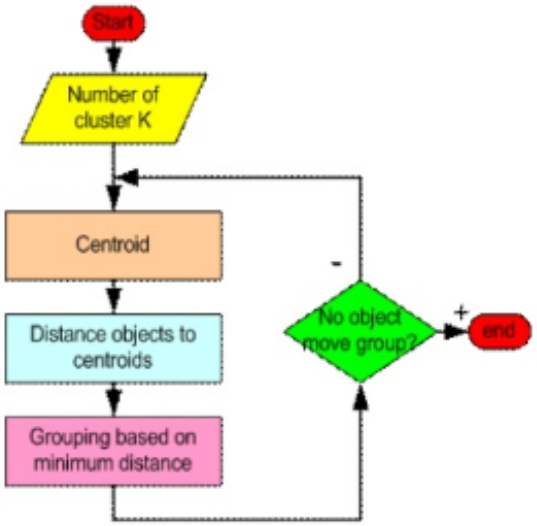

Figure 4: K-Means clustering workflow

3.2 Learning SOM Algorithm Overview

SOM uses an unsupervised learning technique where it doesn't need a target output to be specified unlike many other sorts of the network. Instead, where the prototype weights match the input vector, that area of the lattice is selectively optimized to more closely resemble the information for the category the input vector may be a member of. From an initial distribution of random weights, and over many iterations, the SOM eventually settles into a map of stable zones. Each zone is effectively a feature classifier, so you'll think about the graphical output as a kind of feature map of the input space.

Training occurs in several steps and over many iteration:

Step 1: Initializing the Weights

Each prototype's weights are initialized randomly with a value near zero but not zero. The word "weight" here carries an entirely other meaning than it did with standard neural networks. as an example, with artificial neural networks we multiply the input neuron value by the burden and, finally, applied an activation function. With SOMs, on the opposite hand, there's no activation function. Weights don't seem to separate from prototypes here. In SOM, the weights belong to the output prototype itself. Rather than being the results of adding up the weights, the output prototype in an exceedingly SOM contains the weights as its coordinates. Each of those output prototypes doesn't exactly become parts of the input space, but try and integrate into it nevertheless, developing imaginary places for themselves.

Step 2: Calculating the Best Matching Unit

Every prototype is examined to calculate which of them weights are most just like the input vector. The winning prototype is often called the Best Matching Unit (BMU). To determine the best matching unit, one method is to iterate through all the prototypes and calculate the Euclidean distance between each prototype's weight vector and the current input vector. The prototype with a weight vector closest to the input vector is tagged as the BMU.
The Euclidean distance is given as:

$$EuclideanDistance = \sqrt{(Observevalue - centroidvalue)^2 + (Observevalue - centroidvalue)^2} \tag{1}$$

$$\sqrt{\sum_{i=0}^{i=n}(X_i - W_i)^2} \tag{2}$$

Where X is the current input vector and W is the prototype's weight vector.

Step 3: Calculating the size of the neighborhood around the BMU

The radius of the neighborhood of the BMU is now calculated. This value often starts large, typically set to the 'radius' of the lattice, but diminishes each time step. Any prototypes found within this radius are deemed to be inside the BMU's neighborhood.

The size of the neighborhood around the BMU is decreasing with an exponential decay function. It shrinks on each iteration until reaching just the BMU.

$$\sigma(t) = \sigma exp(-\frac{t}{\lambda}) \tag{3}$$

Where t = 0, 1, 2, 3.... The figure below shows how the neighborhood decreases over time with each iteration

3.2.1 Neighborhood Function

Neighborhood function determines the rate of change of the neighborhood around the winner prototype. Neighborhood function influences the training result of the SOM procedure. Therefore, it is important to choose the proper neighborhood function with the data set.

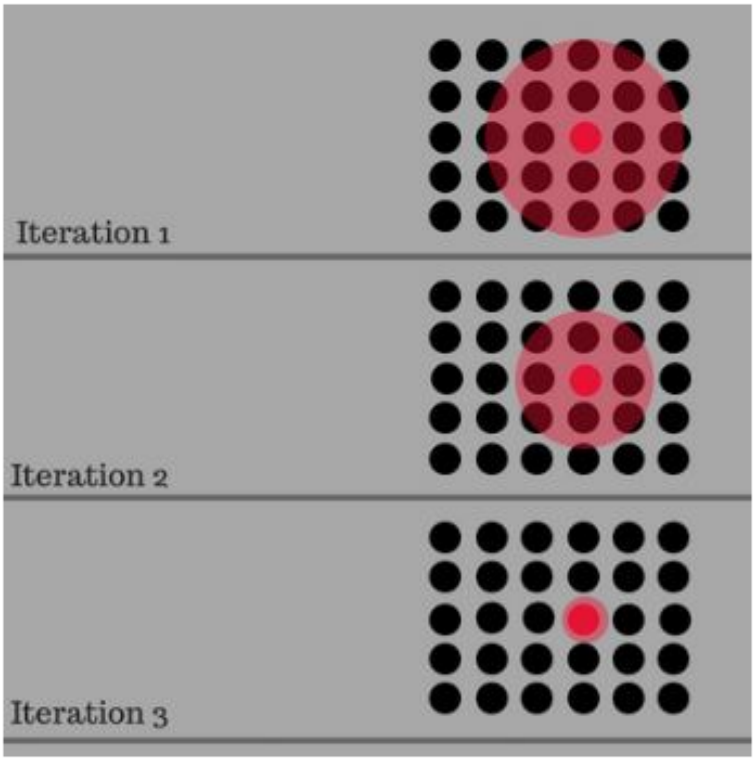

Figure 5: neighborhood decreases over time

There are various Neighborhood functions as follows:

- Gaussian
- Bubble
- Maxican Hat
- Triangle

The selection of neighborhood function depends upon σ and the training of SOM. We have used the Bubble function in this research case for better approximation and training of SOM as it is a constant function and change in weights modification occurs at a constant rate

while Gaussian function is decreasing function in the defined neighborhood of the winner prototype.

How to set the radius value in the self-organizing map?
It depends largely on the range and scale of your input data used for training SOM. As we are normalizing feature values to a range of [0, 1] then we try to train SOM with σ=1. Remember, we have to decrease the learning rate α and the size of the neighborhood function with increasing iterations, as none of the metrics stay constant throughout the iterations in SOM.

It also depends on how large your SOM is. σ is sometimes based on the Euclidean distance between the centroids of the first and second closest clusters.
Step 4: Adjusting the Weights
Every prototype within the BMU's neighborhood (including the BMU) has its weight vector adjusted according to the following equation:
New Weights = Old Weights + Learning Rate (Input Vector — Old Weights)
W (t+1) = W (t) + L (t) (V (t) — W (t))
Where t represents the time-step and L is a small variable called the learning rate, which decreases with time.

3.2.2 Learning Rate

The learning rate controls how quickly the model is adapted to the problem situation. Smaller learning rates require more training epochs given the smaller changes made to the weights of each update, whereas larger learning rates result in rapid changes and require fewer training epochs.

The decay of the learning rate is calculated for every iteration using the following equation:

$$L(t) = L_0 exp(-\frac{t}{\lambda}) \tag{4}$$

Where λ is the decay function As training goes on, the neighborhood gradually shrinks. At the end of the training, the neighborhood has shrunk to size zero. Neighborhood function is given as:

$$exp(-\frac{dist^2}{2\sigma^2(t)}) \tag{5}$$

3.3 SOM Mathematically

1. Given K, the number of prototypes, and a low dimensional lattice $W = [w_1, , , w_k]$, equipped with a neighbourhood structure, or topology. Set T_{Max} to be the maximum number of iterations.

2. **Initialize :** Define the initial prototypes, $w_l, 1 \leq l \leq k, w_j \neq w_k for j \neq k$. Set t = 0 . Denote the initial prototypes by $w_l(0)$.

3. **Iteration:** for $t < T_{max}$

   (a) Draw a data vector $x^{(t)}$.

   (b) **Find the best matching unit(BMU)** $w_{(t)}(t)$ among the current prototypes $w_l(t), 1 \leq l \leq k$
   $||w_{j(t)}\ (t) - x^{(t)}|| = min\{||w_{j(t)}(t) - x^{(t)}|| \Big| 1 \leq l \leq k\}$.

   (c) **Compute the current learning rate** $\alpha(t)$ and the j(t)th of the neighborhood matrix $h_{i,j(t)}$, and update all prototype vectors.
   $w_l(t+1) = w_l(t) + \alpha(t) h_{l,j(t)}(t) \Big(x^{(t)} - w_{(l)}(t) \Big)$.

   (d) Advance the counter by one, $t \rightarrow t + 1$

Code Walk Through:

Step 1: Reading Dataset

Step 2: Exploring Dataset

```
dataset = pd.read_csv('Credit_Card_Applications.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Figure 6: Reading data from csv

The dataset contains 14 features A1 to A14 and a class label. we'll use these features to predict Credit Cards frauds. Class 1 represents whose applications got approved and class 0 represents the frauds.

Step 3: Min Max Scaling

	CustomerID	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	Class
0	15776156	1	22.08	11.460	2	4	4	1.585	0	0	0	1	2	100	1213	0
1	15739548	0	22.67	7.000	2	8	4	0.165	0	0	0	0	2	160	1	0
2	15662854	0	29.58	1.750	1	4	4	1.250	0	0	0	1	2	280	1	0
3	15687688	0	21.67	11.500	1	5	3	0.000	1	1	11	1	2	0	1	1
4	15715750	1	20.17	8.170	2	6	4	1.960	1	1	14	0	2	60	159	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
685	15808223	1	31.57	10.500	2	14	4	6.500	1	0	0	0	2	0	1	1
686	15769980	1	20.67	0.415	2	8	4	0.125	0	0	0	0	2	0	45	0
687	15675450	0	18.83	9.540	2	6	4	0.085	1	0	0	0	2	100	1	1

Figure 7: Source Data CSV view

In this approach, the data is scaled to a fixed range — more often than to not 1. We'll conclusion up with littler standard deviations, which might smother the impact of exceptions.

Step 4: Training SOM

```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
X = sc.fit_transform(X)
```

Figure 8: Min Max Scaling

Object of minisom class is made and initialized with random weights using $som.random_weights_init()$ function.

```
from minisom import MiniSom
som = MiniSom(x = 10, y = 10, input_len = 16, sigma = 1.0, learning_rate = 0.1, neighborhood_function = 'bubble')
som.random_weights_init(X)
som.train_random(data = X, num_iteration = 100)
```

Figure 9: SOM Initialisation

x and y are the dimensions of the SOM (it shouldn't be small so outliers may be detected). x*y gives the entire number of maximum classes/prototypes present within the SOM grid, into which the input patterns/records from the data set are classified.

input_len are the number of features of X(customer id included to search out the id of faulters).

sigma is the radius and *learning_rate* is that the rate at which SOM learn .

$som.train_random()$ function trains the SOM for the provided input vectors X . Randomly, a record is chosen from the given patterns and it's then fed to the network. The winner prototype (the prototype whose euclidean distance to the given input vector is minimum) is that the predicted class for the given input vector/record. Then, the change in weights (or features) of the winner prototype in SOM is finished therefore the euclidean distance between the winner prototype and thus the input vector decreases. Also, weights of neighbors of the winner prototypes are updated within the identical sense to decrease the euclidean distance specified the weights of neighboring prototypes which are closer to the winner prototype are updated over those that lie far-flung from the winner prototype.

After the training process, all the input vectors/records of the data set will be matched to the prototypes in the SOM grid. The prototypes in the grid with similar properties will lie closer and those with different properties will lie at a longer distance to other prototypes (these prototypes will be outliers in the SOM grid). The features given in the data set v1 to v14 include parameters like the place of transaction, time taken for a transaction, how different is the amount of transaction from the usual amount of transactions of the user, etc. . .

Transactions (records/patterns) that do not seem to be fraud will match to the prototypes within the SOM grid which are closer to each other as they possess similar properties/ features and transactions which are able to match to the prototypes which lie as outliers

within the SOM grid as they possess unusual properties/features like time taken for the transaction is different, amount of transaction is solely an excessive amount of bigger than the quality amount of transaction of the user, etc.

So, we will determine the frauds by trying to find the records/transactions which were matched to the outlier prototypes within the SOM grid.

It's not necessary the records predicted as frauds are to be frauds. they'll be predicted as frauds because of the weird behavior of the user. If this can be often the case, we may apply additional security measures to verify the user.

4 Results

4.1 SOM Analysis

4.1.1 Efficiency v/s Learning rate

Training SOM model for learning rate with different sigma values to finalize the learning rate with highest efficiency

Iteration 1 Learning Rate = 0.1			
S No	Learning Rate	Sigma	Efficency in Percentage
1	0.1	1	76
2		2	63
3		3	62
4		4	57
5		5	58
6		6	37
7		7	15
8		8	39
9		9	19
Average Efficiency for Learning rate 0.1			47.33333333

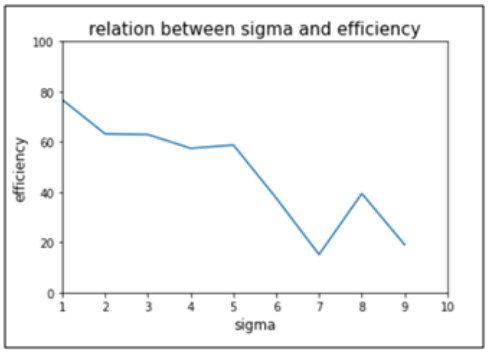

Figure 10: Analysis of efficiency with constant LR and varying sigma for LR 0.1

Iteration 2 Learning Rate = 0.2			
S No	Learning Rate	Sigma	Efficency in Percentage
1	0.2	1	68
2		2	62
3		3	58
4		4	42
5		5	24
6		6	7
7		7	15
8		8	13
9		9	12
Average Efficiency for Learning rate 0.2			33.44444444

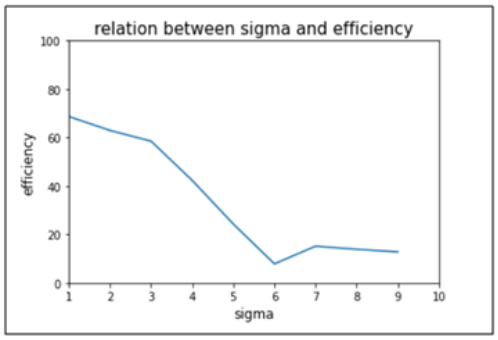

Figure 11: Analysis of efficiency with constant LR and varying sigma for LR 0.2

Iteration 3 Learning Rate = 0.3			
S No	Learning Rate	Sigma	Efficency in Percentage
1	0.3	1	81
2		2	44
3		3	72
4		4	26
5		5	15
6		6	20
7		7	24
8		8	9
9		9	15
Average Efficiency for Learning rate 0.3			34

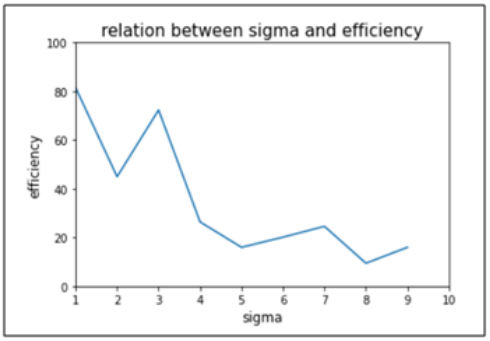

Figure 12: Analysis of efficiency with constant LR and varying sigma for LR 0.3

Iteration 4 Learning Rate = 0.4			
S No	Learning Rate	Sigma	Efficency in Percentage
1	0.4	1	76
2		2	89
3		3	31
4		4	33
5		5	23
6		6	22
7		7	17
8		8	10
9		9	36
Average Efficiency for Learning rate 0.4			37.44444444

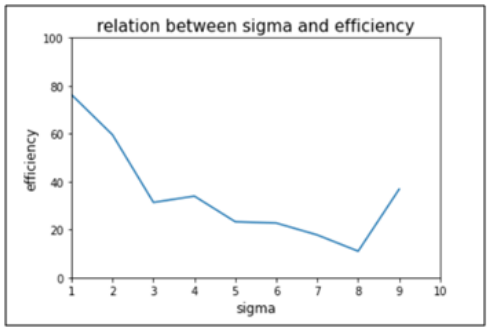

Figure 13: Analysis of efficiency with constant LR and varying sigma for LR 0.4

Iteration 5 Learning Rate = 0.5			
S No	Learning Rate	Sigma	Efficency in Percentage
1	0.5	1	70
2		2	54
3		3	31
4		4	21
5		5	9
6		6	45
7		7	23
8		8	25
9		9	7
Average Efficiency for Learning rate 0.5			31.66666667

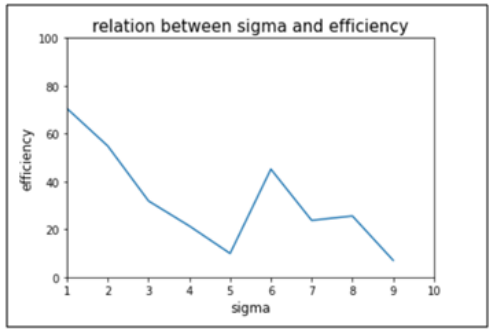

Figure 14: Analysis of efficiency with constant LR and varying sigma for LR 0.5

Iteration 6 Learning Rate = 0.6			
S No	Learning Rate	Sigma	Efficency in Percentage
1	0.6	1	80
2		2	49
3		3	21
4		4	43
5		5	29
6		6	31
7		7	22
8		8	30
9		9	15
Average Efficiency for Learning rate 0.6			35.55555556

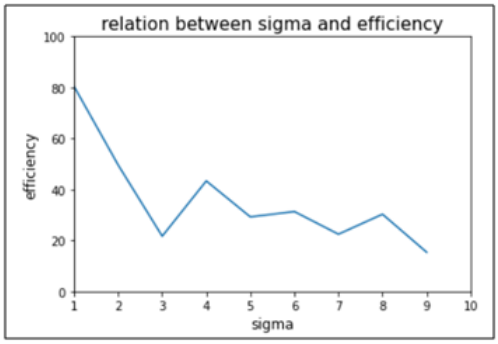

Figure 15: Analysis of efficiency with constant LR and varying sigma for LR 0.6

Iteration 7 Learning Rate = 0.7			
S No	Learning Rate	Sigma	Efficency in Percentage
1	0.7	1	50
2		2	48
3		3	46
4		4	31
5		5	11
6		6	13
7		7	24
8		8	5
9		9	28
Average Efficiency for Learning rate 0.7			28.44444444

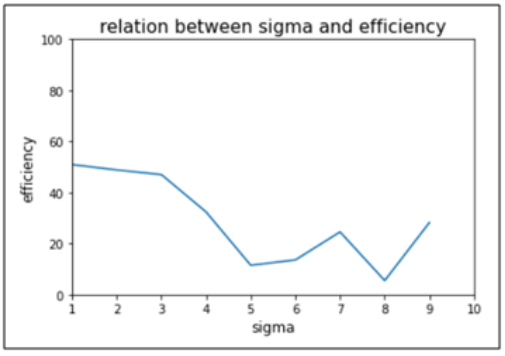

Figure 16: Analysis of efficiency with constant LR and varying sigma for LR 0.7

Iteration 8 Learning Rate = 0.8			
S No	Learning Rate	Sigma	Efficency in Percentage
1	0.8	1	70
2		2	79
3		3	56
4		4	26
5		5	62
6		6	36
7		7	28
8		8	31
9		9	42
Average Efficiency for Learning rate 0.8			47.77777778

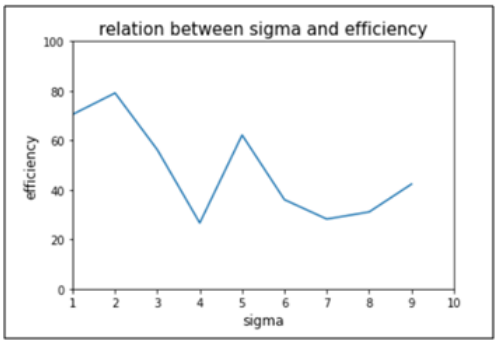

Figure 17: Analysis of efficiency with constant LR and varying sigma for LR 0.8

Iteration 9 Learning Rate = 0.9			
S No	Learning Rate	Sigma	Efficency in Percentage
1	0.9	1	78
2		2	30
3		3	20
4		4	44
5		5	34
6		6	13
7		7	55
8		8	21
9		9	28
Average Efficiency for Learning rate 0.9			35.88888889

Figure 18: Analysis of efficiency with constant LR and varying sigma for LR 0.9

Observations from Analysis for Learning Rate: We could see that we have got maximum efficiency for Learning rates 0.1 and 0.2, Hence we are taking the learning rate parameter to 0.1 for better approximation for SOM training. Since we are getting maximum efficiency with this value we can make this parameter constant in SOM.

4.1.2 Efficiency v/s Sigma rate

Iteration 1 Sigma = 1			
S No	Sigma	Learning rate	Efficency in Percentage
1	1	0.1	71
2		0.2	60
3		0.3	71
4		0.4	65
5		0.5	52
6		0.6	66
7		0.7	64
8		0.8	58
9		0.9	51
Average Efficiency for Sigma =1			62

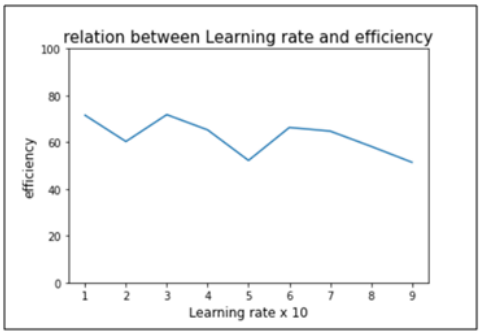

Figure 19: Analysis of efficiency with constant Sigma and varying LR for Sigma 1

Iteration 1 Sigma = 2			
S No	Sigma	Learning rate	Efficency in Percentage
1	2	0.1	69
2		0.2	65
3		0.3	45
4		0.4	62
5		0.5	53
6		0.6	26
7		0.7	36
8		0.8	51
9		0.9	72
Average Efficiency for Sigma =2			53.22222222

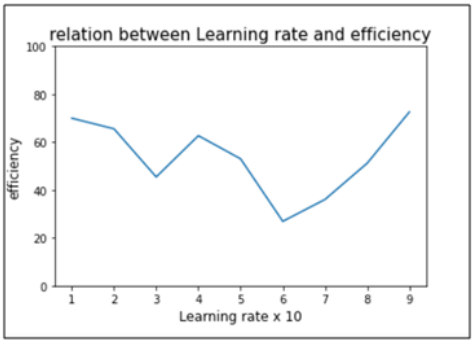

Figure 20: Analysis of efficiency with constant Sigma and varying LR for Sigma 2

Iteration 1 Sigma = 3			
S No	Sigma	Learning rate	Efficency in Percentage
1	3	0.1	73
2		0.2	38
3		0.3	31
4		0.4	27
5		0.5	20
6		0.6	44
7		0.7	37
8		0.8	35
9		0.9	25
Average Efficiency for Sigma =3			36.66666667

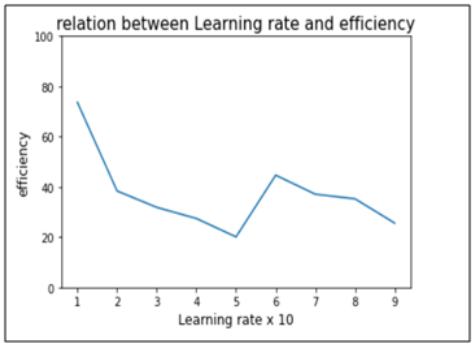

Figure 21: Analysis of efficiency with constant Sigma and varying LR for Sigma 3

Iteration 1 Sigma = 4			
S No	Sigma	Learning rate	Efficency in Percentage
1	4	0.1	51
2		0.2	50
3		0.3	42
4		0.4	38
5		0.5	18
6		0.6	23
7		0.7	33
8		0.8	39
9		0.9	34
Average Efficiency for Sigma =4			36.44444444

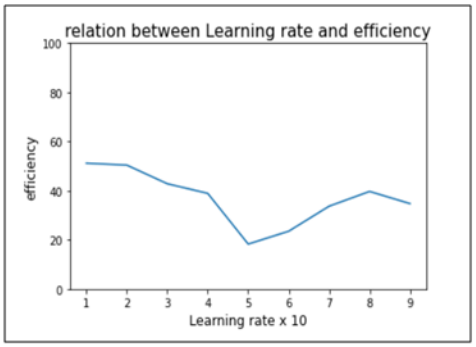

Figure 22: Analysis of efficiency with constant Sigma and varying LR for Sigma 4

Iteration 1 Sigma = 5			
S No	Sigma	Learning rate	Efficency in Percentage
1	5	0.1	65
2		0.2	60
3		0.3	26
4		0.4	42
5		0.5	10
6		0.6	19
7		0.7	21
8		0.8	30
9		0.9	45
Average Efficiency for Sigma =5			35.33333333

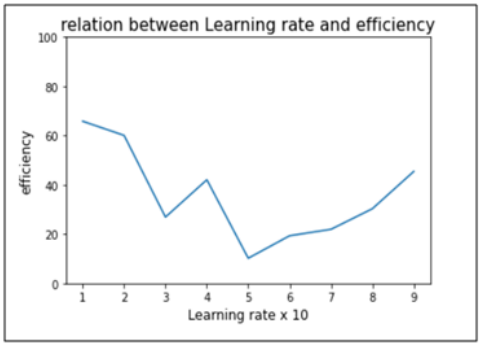

Figure 23: Analysis of efficiency with constant Sigma and varying LR for Sigma 5

Iteration 1 Sigma = 6			
S No	Sigma	Learning rate	Efficency in Percentage
1	6	0.1	49
2		0.2	29
3		0.3	57
4		0.4	14
5		0.5	25
6		0.6	30
7		0.7	7
8		0.8	37
9		0.9	35
Average Efficiency for Sigma =6			31.44444444

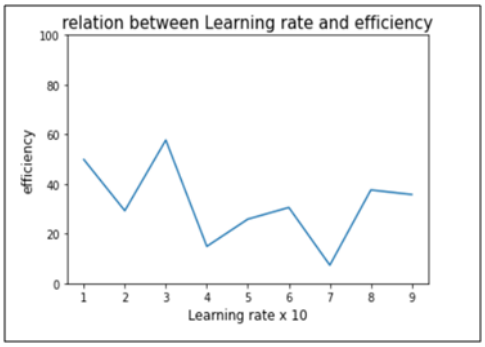

Figure 24: Analysis of efficiency with constant Sigma and varying LR for Sigma 6

Iteration 1 Sigma = 7			
S No	Sigma	Learning rate	Efficency in Percentage
1	7	0.1	44
2		0.2	20
3		0.3	16
4		0.4	22
5		0.5	13
6		0.6	12
7		0.7	21
8		0.8	5
9		0.9	26
Average Efficiency for Sigma =7			19.88888889

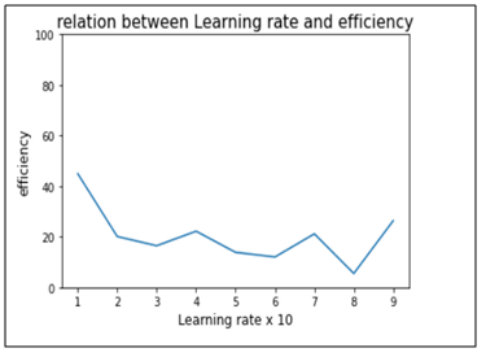

Figure 25: Analysis of efficiency with constant Sigma and varying LR for Sigma 7

Iteration 1 Sigma = 8			
S No	Sigma	Learning rate	Efficency in Percentage
1	8	0.1	47
2		0.2	6
3		0.3	12
4		0.4	21
5		0.5	9
6		0.6	8
7		0.7	16
8		0.8	25
9		0.9	22
Average Efficiency for Sigma =8			18.44444444

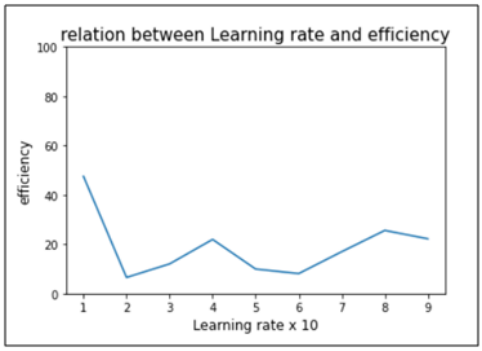

Figure 26: Analysis of efficiency with constant Sigma and varying LR for Sigma 8

Iteration 1 Sigma = 9			
S No	Sigma	Learning rate	Efficency in Percentage
1	9	0.1	27
2		0.2	12
3		0.3	18
4		0.4	17
5		0.5	32
6		0.6	17
7		0.7	40
8		0.8	28
9		0.9	26
Average Efficiency for Sigma =9			24.11111111

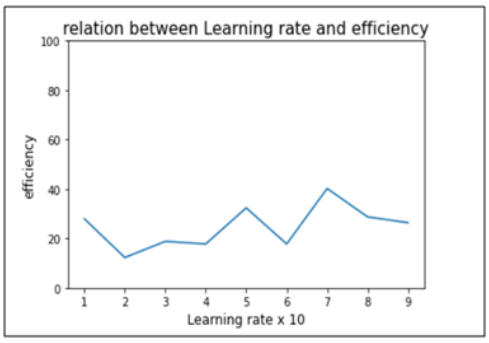

Figure 27: Analysis of efficiency with constant Sigma and varying LR for Sigma 9

Observations from Analysis for Sigma: We could see that we have received maximum efficiency for Sigma 1 and 2, Hence we are taking Sigma parameter to 1 for better approximation for SOM training. Since we are getting maximum efficiency with this value we can make this parameter constant in SOM.

4.1.3 Running SOM model with learned Sigma Learning rate analyzed

```
Right Prediction:57.51633986928104
Percentage of frauds detected out of total actual frauds:91.90600522193212
Percentage of fraud customers in population as claimed by prediction model:51.01449275362319
Actual precentage of frauds out of total population:55.507246376811594
Affected population(percentage of custormers who were not fraud but were predicted as fraud)):37.68115942028986
```

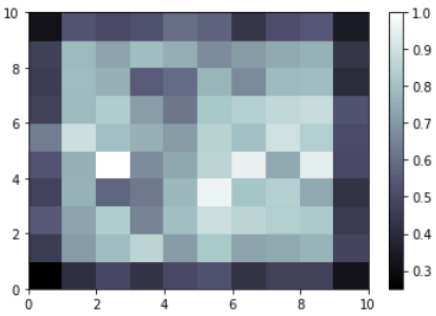

Figure 28: SOM model with LR = 0.1 and Sigma = 0.1 and Neighbourhood function = "bubble"

We could see that with Learning Rate = 0.1 and Sigma = 1 we have achieved the most efficient model with efficiency = 91.9 percent.

Visualising the Grid: The distance between the SOM grid's prototypes is calculated. Those prototypes with the smallest inter-prototype distance will form a cluster, whereas those with the largest inter-prototype distance will form outliers. The patterns or data that

match these anomalies are almost certainly fraudulent. The white boxes in the distance map above correspond to prototypes with a considerable mean inter-prototype distance. The smallest mean inter-prototype distance is black, and the greatest mean inter-prototype distance is white.

Final Model efficiency overview

Our fraud detection algorithm had an accuracy of 91.9 percent, while the affected population (those who were suspected of fraud but were not) was found to be 37.6 percent. Because initialization of the weights of the prototypes of the SOM grid is done by randomly selecting records from the input space.

5 Future work

5.1 Fraud Detection Application

- Lowering the affected population which was considered fraud while they aren't by using time series to enhance the results as we had the data for the time for each input.

- trying to find a better initialization to the SOM model than the random initialization.

5.2 Other applications

as we got a hand on SOM as a model for unsupervised learning, we are also interested in using it in other applications such as:

- Natural language processing.

- Anomaly Detection.

References

D. Calvetti and E. Somersalo.Mathematics of Data Science, A Computational Approach to Clustering and Classification. SIAM Philadelphia, 2021

https://medium.com/@yashwant2451/
credit-card-fraud-detection-using-self-organizing-featuremaps-f6e8bca707bd

https://medium.com/@yashwant2451/
credit-card-fraud-detection-using-self-organizing-featuremaps-f6e8bca707bd

https://medium.com/machine-learning-researcher/self-organizing-map-som-c296561e2117

https:
//www.academia.edu/45494309/Credit_Card_Fraud_Detection_Using_Self_Organizing_Maps

https://arxiv.org/ftp/arxiv/papers/1611/1611.06439.pdf

https://web.stanford.edu/group/pdplab/pdphandbook/handbookch7.html

http://www.sci.utah.edu/~arpaiva/classes/UF_eel6814/clustering_and_SOM.pdf

https://www.researchgate.net/publication/327442770_Efficient_Neighborhood_Function_
and_Learning_Rate_of_Self-Organizing_Map_SOM_for_Cell_Towers_Traffic_Clustering

http://www.ijmo.org/vol6/504-M08.pdf

https:
//www.researchgate.net/publication/317339061_Brief_Review_of_Self-Organizing_Maps

Statutory Declaration

Hiermit versichere ich, dass diese Arbeit von mir persönlich verfasst ist und dass ich keinerlei fremde Hilfe in Anspruch genommen habe. Ebenso versichere ich, dass diese Arbeit oder Teile daraus weder von mir selbst noch von anderen als Leistungsnachweise andernorts eingereicht wurden. Wörtliche oder sinngemäße Übernahmen aus anderen Schriften und Veröffentlichungen in gedruckter oder elektronischer Form sind gekennzeichnet. Sämtliche Sekundärliteratur und sonstige Quellen sind nachgewiesen und in der Bibliographie aufgeführt. Das Gleiche gilt für graphische Darstellungen und Bilder sowie für alle Internet-Quellen. Ich bin ferner damit einverstanden, dass meine Arbeit zum Zwecke eines Plagiatsabgleichs in elektronischer Form anonymisiert versendet und gespeichert werden kann. Mir ist bekannt, dass von der Korrektur der Arbeit abgesehen und die Prüfungsleistung mit nicht ausreichend bewertet werden kann, wenn die Erklärung nicht erteilt wird.
I hereby declare that the paper presented is my own work and that I have not called upon the help of a third party. In addition, I affirm that neither I nor anybody else has submitted this paper or parts of it to obtain credits elsewhere before. I have clearly marked and acknowledged all quotations or references that have been taken from the works of others. All secondary literature and other sources are marked and listed in the bibliography. The same applies to all charts, diagrams and illustrations as well as to all Internet resources. Moreover, I consent to my paper being electronically stored and sent anonymously in order to be checked for plagiarism. I am aware that the paper cannot be evaluated and may be graded "failed" ("nicht ausreichend") if the declaration is not made.

_____

Signature

_____

Place, Date