# MACHINE INTELLIGENCE AND EXPERT SYSTEMS

## TERM PROJECT

## REPORT

**GROUP NUMBER: 8**

MEMBERS:

1. Ravi Ghadia 17EC10045
2. Shubham Maheshwari 17EC10055
3. Arpit Dwivedi 17EC35005
4. Pankaj Mishra 17EC35034
5. Wayal Rushikesh 17EC35044

## PROBLEM STATEMENT :

Given the data for Mouse dynamics of various people, design an algorithm using **Self Organizing Maps** to differentiate an imposter from an authentic user. The purpose of such a system could be to authenticate a user based on his/her mouse usage and restrict usage if any suspicious activity is detected, and therefore prevent a security breach.

## INTRODUCTION :

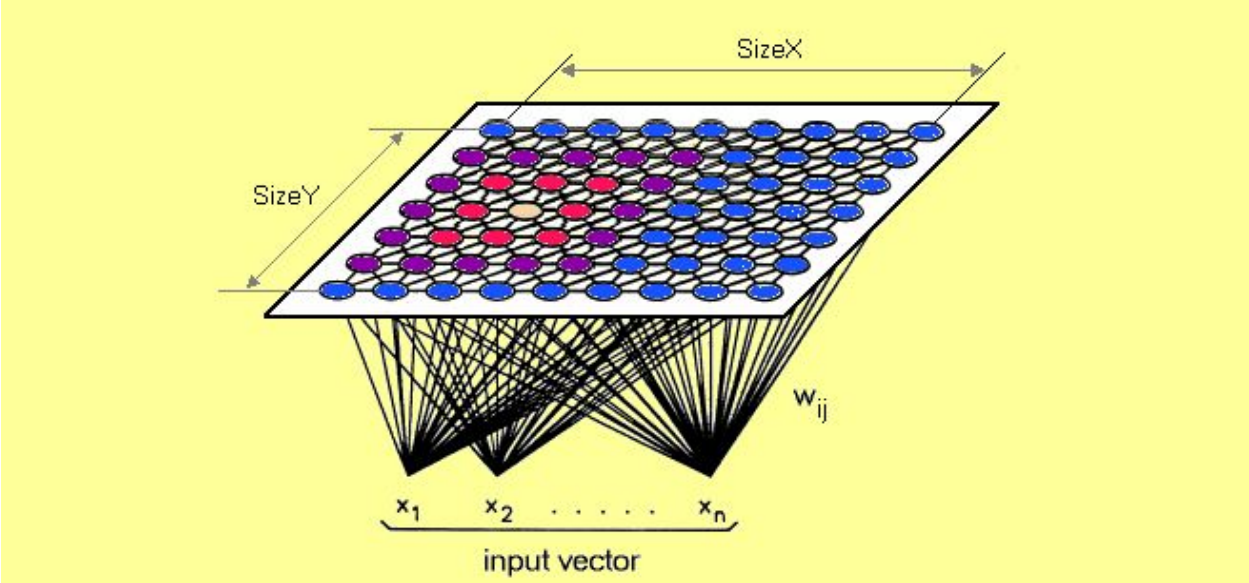
In today's data driven world, it is as much necessary to protect the sensitive data as it is to collect and store it. Security breaches on some public systems are not uncommon and could lead to some serious issues at times. Thus, there needs to be some security systems to keep a watch on whether the person using the system is authenticated to do so. Some common solutions for this are keeping the systems or files password protected, but they may unfortunately fail if somehow the passwords have been cracked or leaked. An assuring solution in such cases might be to judge the user based on his/her behavior, something which he/she can not change at an instant.

In this report, we try to build an authentication system based on the behavioral pattern of Mouse usage by a user. We have collected data from 10 people to train our preliminary model. Although the results established in this report do not directly guarantee a fully functional Mouse dynamics based authentication system, but it surely forms a baseline for such systems and gives a broad idea on how to achieve it. For our purpose, we have used Self Organizing Maps to build the model.

**Understanding Self-Organizing Maps :**

Self Organizing Neural Maps are a type of Neural Networks with a typical difference in the way in which they are trained. Instead of classical methods of updating the weights of a Neural Network using SGD or some other gradient descent, the weights of SOM are updated using competitive learning which can be explained with the following three step process :-

1. Competition
2. Cooperation
3. Adaptation




(Courtesy: Achraf KHAZRI Self Organizing Maps www.medium.com)

During training, it first tries to assign a winner neuron/node to the input data point. Then it tries to find the neighbours of this winner neuron. And finally it updates weights of the winner as well as the neighbour neurons. The learning is Adaptive in the sense that the learning rate keeps on decreasing with increase in training time and asymptotically approaches zero.

Broadly speaking, what SOM basically does is dimensionality reduction. Let's say we have a d-dimensional (d>>2) dataset of N points and we train a SOM model of size AxB neurons. Then, one of these AB neurons will be the winner neuron for each data point and thus our dataset of N points is now being described by one of the AB neurons (winner neuron) thereby reducing the dimensionality from d to 2.

**Learning rate decay:**

$$\alpha(t) = \alpha_0 \cdot \left(\frac{\alpha_{\text{end}}}{\alpha_0}\right)^{t/t_{\max}}$$

**Neighbourhood decay function**(initial value of neigh. function=$\sigma_0$)**:**

$$\sigma(t) = \sigma_0 \cdot \left(1 - \frac{t}{t_{\max}}\right)$$

**Neighbourhood distance weight function:**

$$h_{c,i}(t) = \exp\left(-\frac{d^2}{2 \cdot \sigma(t)^2}\right)$$

d=distance between BMU c and any other node i

**Weight update rule:**

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \alpha(t) \cdot h_{c,i}(t) \cdot (\boldsymbol{x}(t) - \boldsymbol{w}_i(t))$$

## METHODOLOGY :

### DATA PREPROCESSING:

We try to model the user behaviour using certain Mouse dynamics. The data regarding the Mouse dynamics that we have is:

| Notation | Meaning |
|---|---|
| MC, n, t: | *Mouse Clicked, Click count, Relative time* |
| MP, n, t: | *Mouse Pressed, Button ID, Relative time* |
| MR, n, t: | *Mouse Released, Button ID, Relative time* |
| MM, x, y, t: | *Mouse Moved, x-coordinate, y-coordinate, Relative time* |
| MD, x, y, t: | *Mouse Dragged, x-coordinate, y-coordinate, Relative time* |
| MWM, x, y, w, a, s, t: | *Mouse Wheel Moved, x-coordinate, y-coordinate, Wheel rotation sense, Amount of scrolling, Scroll type, Relative time* |

### FEATURES EXTRACTED :

The data from the raw dataset files can not be used to trin and thus we need to extract features from them for our purpose. We mainly try to focus on Mouse Movements and Mouse Clicks as those were the two kinds of dynamics occurring most frequently in every dataset:

1. **Click time:** Time the user takes to click the button.
2. **Pause time:** Time elapsed between pointing at an object and clicking it.
3. **Linear Movement:** Movement of the cursor(both along x and y axis)
4. **Velocity:** The velocity vector of the cursor(both along x and y axis)
5. **Acceleration:** Acceleration vector of the cursor(both along x and y axis)
6. **Angular Movement:** Movement of cursor along the angular axis.
7. **Angular Speed:** Speed along the angular axis.
8. **Curvature:** Rate of change of angular movement w.r.t linear movement.

For the above vectors, statistics like mean, standard deviation, maximum, minimum, range etc. about these quantities were stored as one datapoint. Several other temporal information such as no. of pauses, pause time, pause time ratio that act critical in depicting behaviour were also mined.

| SPATIAL INFORMATION | TEMPORAL INFORMATION |
|---|---|
| 1. Horizontal coordinates<br>2. Vertical coordinates<br>3. Path distance from the origin<br>4. Angle of the path with respect to X axis<br>5. Curvature of the path<br>6. Derivative of the curvature of the path | 1. Input x values<br>2. Input y values<br>3. Input t values<br>4. Horizontal velocity<br>5. Vertical velocity<br>6. Tangential velocity<br>7. Tangential acceleration<br>8. Tangential jerk<br>9. Angular velocity |

**BUILDING THE CLASSIFIER :**

For the classifier, we have used a Custom SOM Library: SuSi package. This package offers several SOM based models for clustering, regression, and classification. We have used the SuSi classifier for our purpose. The SuSi package SOM classifier makes use of two layers of SOM model, that are trained

in a different manner: One using **unsupervised** training and the other using **supervised** training.

The Unsupervised SOM is similar to the Classical SOM model that we described in the previous Introduction section, ie, directly trained on the unlabelled data and the weight vectors are the same size as the input vector dimension. Thus the unsupervised SOM focuses on assigning BMUs (Best Matching Units aka winner nodes) to each datapoint.

The purpose of the supervised SOM is to map each neuron in the unsupervised SOM to a target label. The weights of the supervised SOM have their dimension equal to the output dimension (in this case 1) and take discrete values depending on the number of classes (in this case 2). The weights are updated using a class change probability matrix.

Thus given an input datapoint, the unsupervised layer will first assign a BMU to it and then the supervised SOM assigns a target label to the given datapoint.

**Note that to train a SuSi classifier, it first trains the unsupervised layer and then uses this trained unsupervised layer to train the supervised layer.** The learning rate and the neighbourhood spread is decreased with iterations for better convergence.

## Training of Supervised SOM:

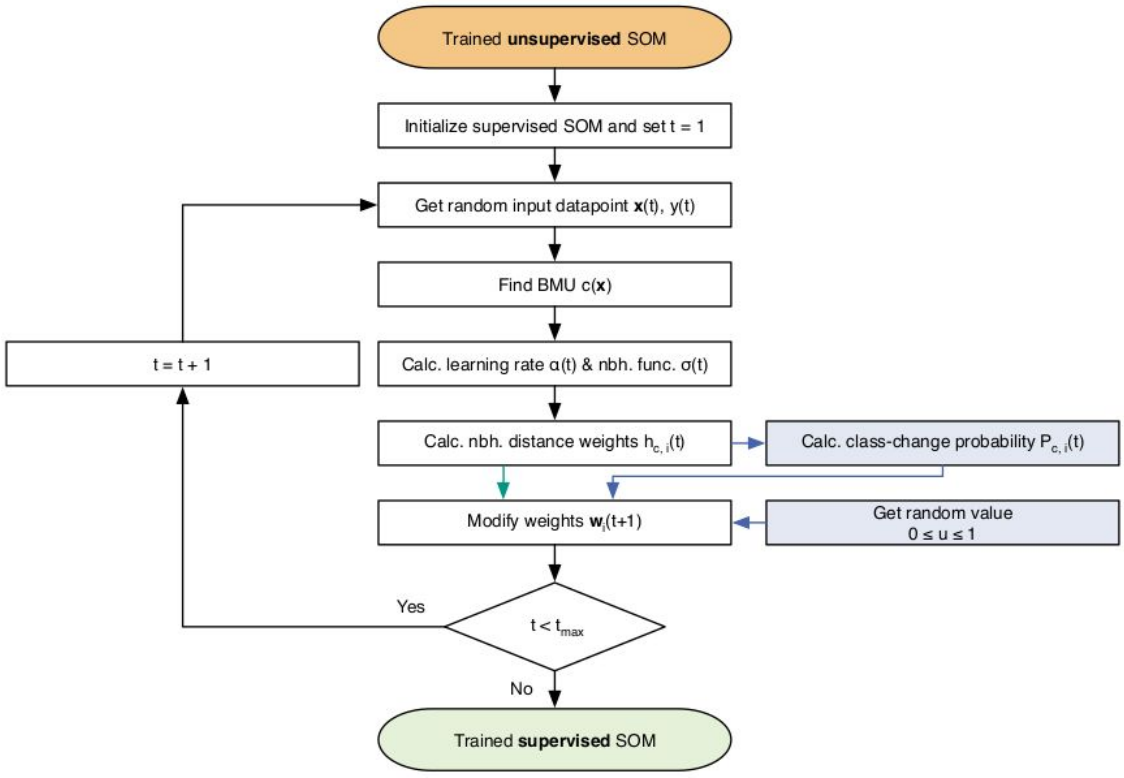



Figure depicting flow chart for SuSi classifier

## Update rule for class change:

$$w_{\text{class}(j)} = \begin{cases} N/(n_{\text{classes}} \cdot N_j), & \text{if class weighting,} \\ 1, & \text{otherwise,} \end{cases}$$

$$P_{c,i}(t) = w_{u(t)} \cdot \alpha(t) \cdot h_{c,i}(t),$$

$$w_i(t+1) = \begin{cases} y(t), & \text{if } u_i(t) < P_{c,i}(t), \\ w_i(t), & \text{otherwise,} \end{cases}$$

where h(t): neighbourhood function , ***α***(t):learning rate function , and u(t): random no. in [0,1]

## RESULTS :

**Cross Validation Results (5 Fold) :**

**Accuracy : 82.54%**

**F1 Score : 82.21%**

The results are obtained by performing 5 fold cross validation. The results are obtained by considering one of the team members as an authenticated user while considering all others as an imposter. Using the following hyperparameters we achieve a classification accuracy of nearly 82.5%.

N_rows = 100, N_columns=100, N_iterations = 200,000.

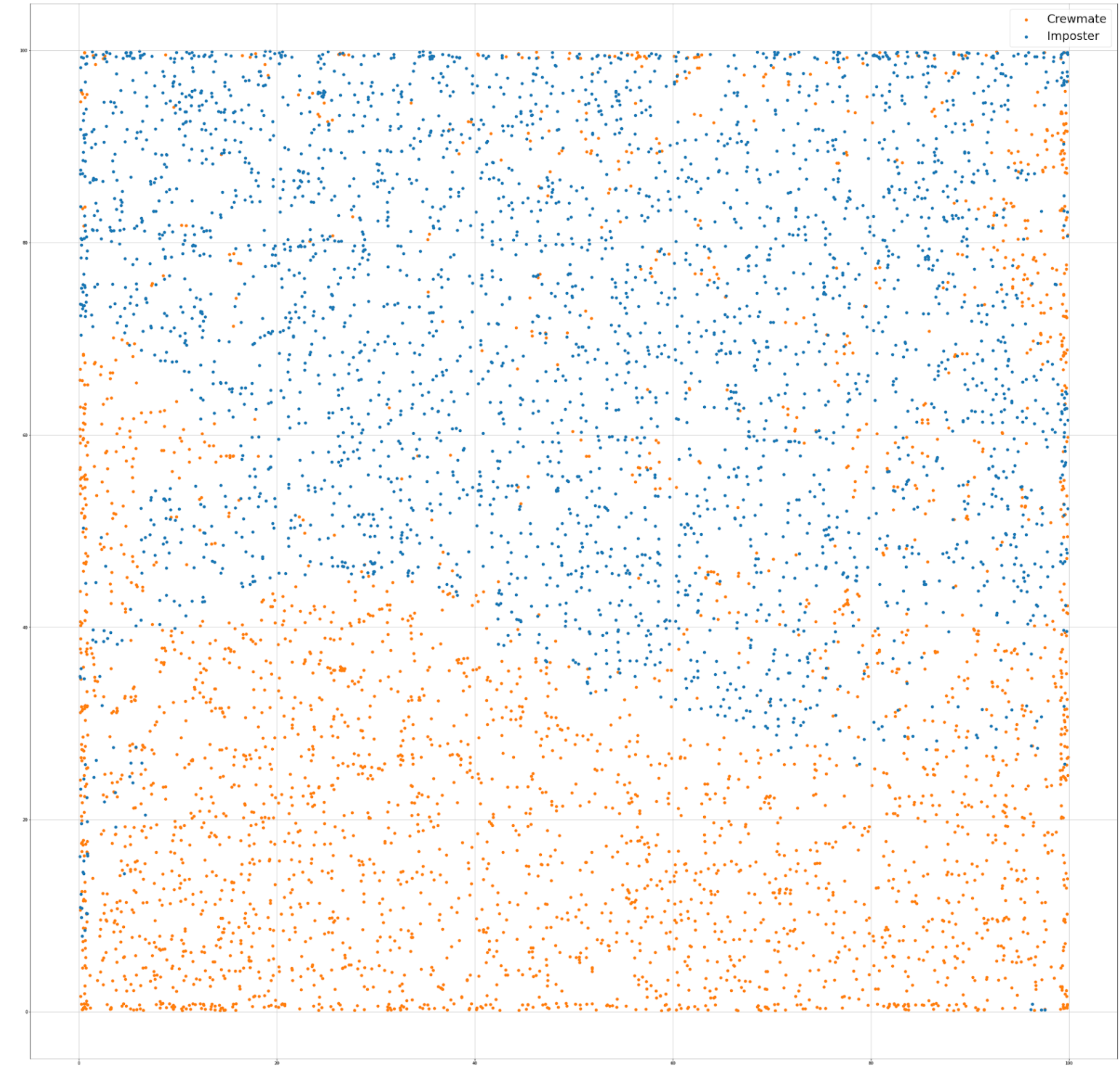
We performed hyperparameter tuning using grid search that trains the model using all listed combinations of hyperparameters and returns the hyperparameters for which the cross validation accuracy is maximum. Some of the grid search results are shown.

Columns :: iters & index :: no of rows

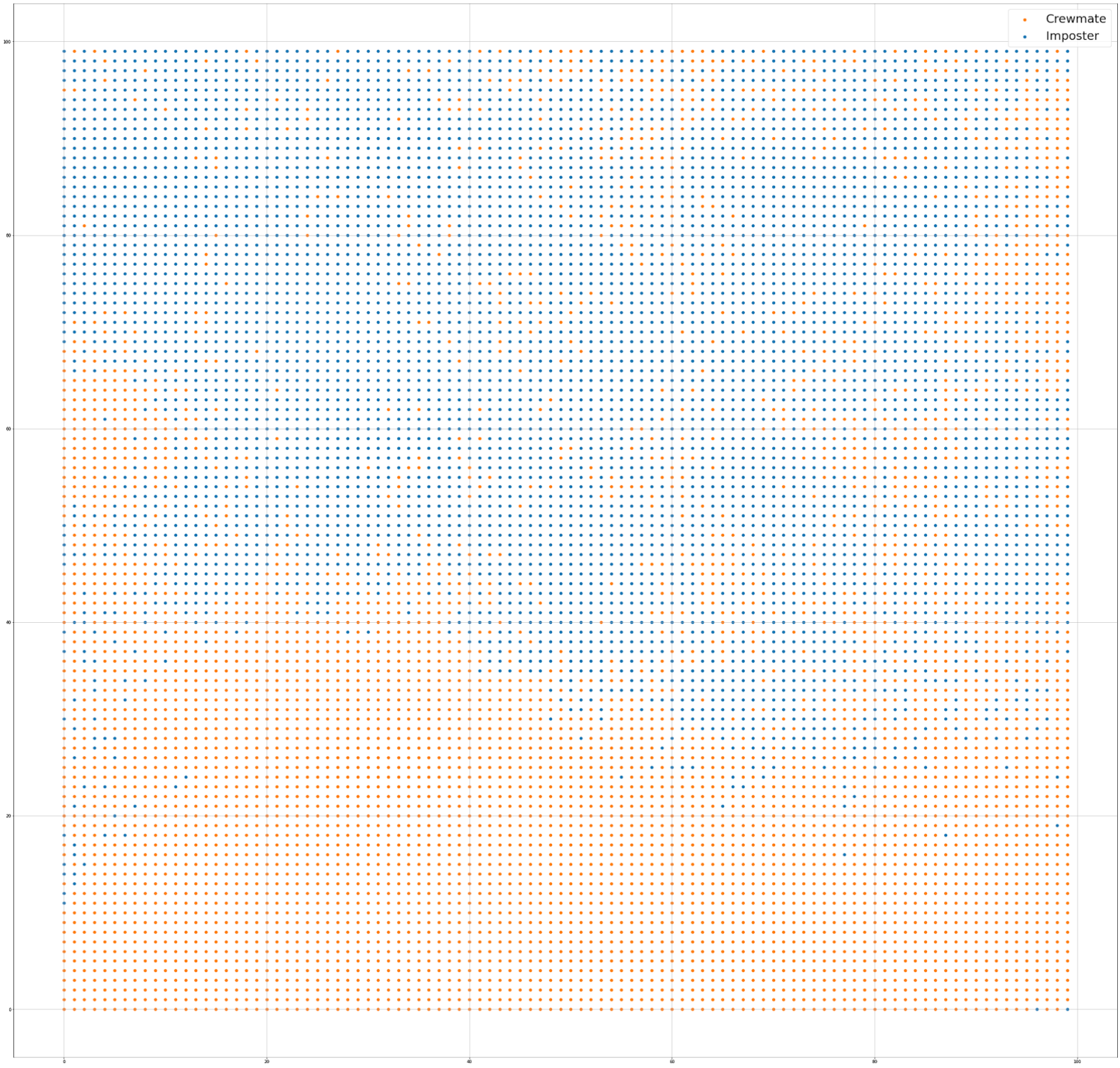
| | 10000 | 25000 | 50000 | 75000 |
|---|---|---|---|---|
| **10** | 0.7305 | 0.6785 | 0.7142 | 0.7280 |
| **25** | 0.7788 | 0.7487 | 0.7919 | 0.7894 |
| **50** | 0.7543 | 0.7863 | 0.8001 | 0.8107 |
| **75** | 0.7838 | 0.7926 | 0.8038 | 0.8145 |
| **100** | 0.7625 | 0.7919 | 0.8107 | 0.8201 |

F1 score is equal to the hypergeometric average of recall and precision and a good F1 score implies good precision and recall. For security breach detection it is extremely important to have good recall (true positives / actual positives) so that all the actual imposters are detected correctly. Using the above hyperparameters we achieved a recall of 80.89%.

**Distribution of the best matching units for the training data (with noise added for better visualization )**



## Distribution of the output labels corresponding to each neuron



## ADDITIONAL WORK DONE

The SOM classifier has two layers one for projection of data and other for classification. If we replaced the classification SOM with a tree based boosting (ensemble) model-XGBOOST and train it in a supervised manner, the accuracy and F1 score were both observed to jump by ~8-10% (> 90% ), owing to the high supervised classification accuracy achieved by tree based models.

This was done as :

1. Train a SOM in an unsupervised way to perform dimensionality reduction.
2. Extract the features (x and y coordinates of the best matching units) for training and validation data.
3. Train the tree based classification model using the extracted features and specified labels.
4. Make predictions on the validation dataset and report the accuracy and F1 score.

## CONCLUSION

Thus we can observe that self organizing maps are a very effective tool and can help distinguish between an authenticated user and an imposter with ease provided a good amount of training data is available. They have various applications in unsupervised machine learning and if used along with good supervised algorithms, they can perform extremely well for classification and regression tasks.

## REFERENCE

F. M. Riese, S. Keller and S. Hinz, "Supervised and Semi-Supervised Self-Organizing Maps for Regression and Classification Focusing on Hyperspectral Data", Remote Sensing, vol. 12, no. 1, 2020.