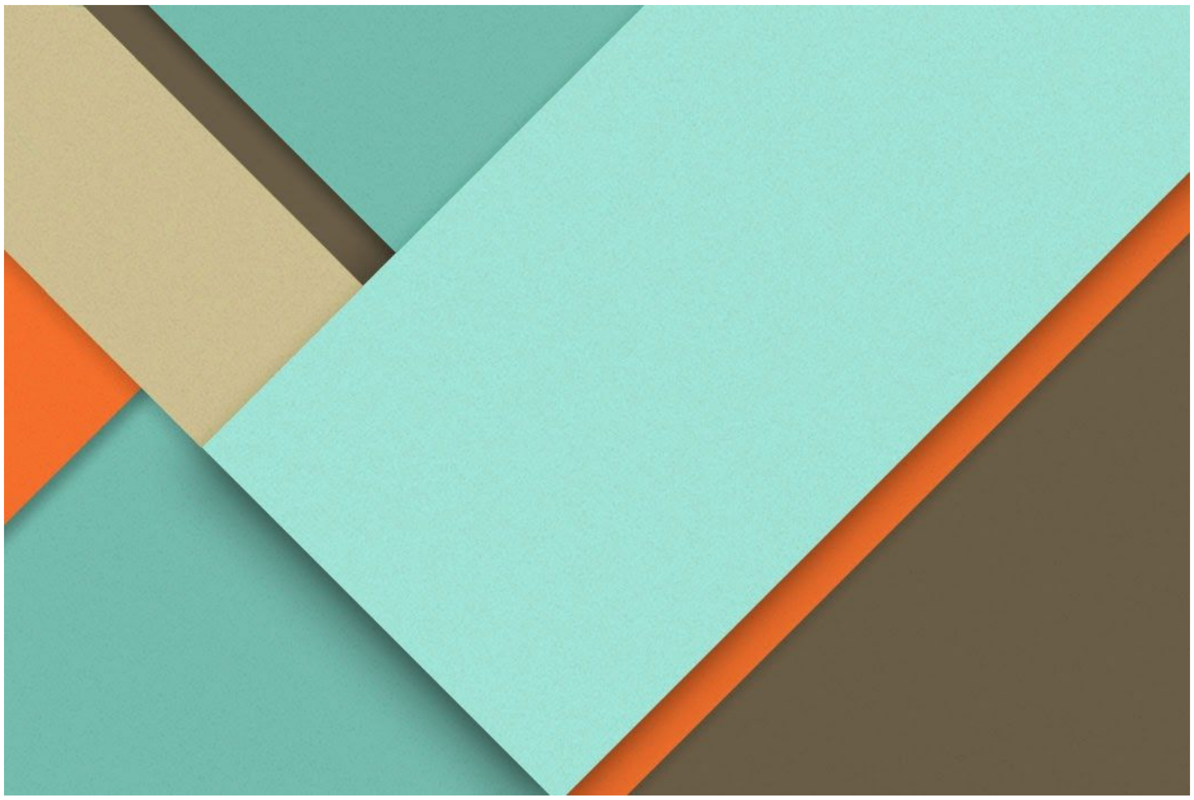


User Authentication based on Key-stroke Logging using Self Organizing maps

Group-4 13/11/2020

Pranit Chawla	(17EC35017)
Pravartya Dewangan	(17EC35041)
Subhadeep Paul	(17EC10064)
Sumegh Roychowdhury	(17EC35033)
Vandit Sharma	(17EC10060)

Problem Statement

To authenticate the user based on the keystroke logging features like hold time and latency time using self-organizing maps.

Introduction

The project aims to detect unauthorized people to prohibit access to the particular system and private data of users. Keystroke logging is an act of tracking and recording every keystroke entry made on a computer. A “keystroke” is just any interaction you make with a button on your keyboard. The two keylogger features for a particular user are almost unique and hence it is used to uniquely identify the user. Hold time basically tells you the length of keypress and the latency time tells the time to switch from one particular key to the next. These two features put the surveillance on the user and simply decline if the trained model declares him as an unauthorized user. Having these two keylogging features along with the mouse dynamics can accurately authenticate the user but we are only using keylogging features in this project.

Self-Organizing Map (SOM)

Self Organizing Map is a type of Artificial Neural Network (ANN) trained using unsupervised learning. The key difference that it has with classical ANNs is that it does not learn weights by backpropagation with gradient descent, rather, it uses competitive learning to adjust neuron weights.

A SOM reduces the input dimensionality in order to represent its distribution as a map. Therefore, SOM forms a map where similar samples are mapped closely together.

SOMs typically have only 2 layers: the input layer and the output layer also called feature map. The activation function used is the identity operator.

Competitive learning is based on three processes:

1) **Competition**:

   Each neuron is assigned a weight vector with the same dimension as the input space (for one training example). The distance between each neuron of the output layer and the input data is calculated. The neuron with lowest distance is denoted as the “winner” of this competition. The Euclidean metric is commonly used to calculate the distance.

2) **Cooperation**

   Certain neighbours of the “winning” neuron are selected whose weights will finally be updated (along with the winning neuron). To do so, a neighborhood kernel function is used, which depends on two factors:

   a) Time
   b) Distance from the winning neuron

3) **Adaptation**

   The selected neuron weights are adapted according to the following formula:

$$\mathbf{w_k = w_k + \eta(t)*h_{ik}(t)*(x^{(n)} - w_k)}$$

Where,

w_k = weight of the ‘k’th neuron

$\eta(t)$ = A learning rate decay rule, such as

$$\mathbf{\eta(t) = \eta_0*e^{-t/T}}$$

$h_{ik}(t)$ = The neighborhood kernel function

For example, if a gaussian kernel is chosen, then

$$h_{ik}(t) = \exp(-d_{ik}^2/2\sigma^2(t))$$

where d_{ik} is the distance between the 'k'th and the 'i'th neuron. In this case, the 'i'th neuron is the "winning" neuron that was selected in the competition stage.

Choices of neighborhood function

We require certain properties of the neighborhood function, h_{ik} , such as:

1. h_{ik} should attain its maximum when $d_{ik} = 0$, and should be symmetric about the winner ('i'th) node.
2. As d_{ik} increases, h_{ik} should decrease.

Any function that meets these requirements is a valid neighborhood kernel function. Some examples follow:

1. Gaussian: $h_{ik}(t) = \exp(-d_{ik}^2/2\sigma^2(t))$
2. Step: $h_{ik}(t) = \{1 \text{ if } d_{ik} \le \sigma(t) \text{ ; } 0 \text{ otherwise}\}$
3. Triangle: $h_{ik}(t) = \{1 - d_{ik}/\sigma(t) \text{ if } d_{ik} \le \sigma(t) \text{ ; } 0 \text{ otherwise}\}$
4. Gaussian with cutoff: $h_{ik}(t) = \{\exp(-d_{ik}^2/2\sigma^2(t)) \text{ if } d_{ik} \le \sigma(t) \text{ ; } 0 \text{ otherwise}\}$

Data Preprocessing

The data fed to a SOM includes all the information that a network gets. If erroneous data is fed to the SOM, the result is also erroneous or of bad quality. Thus, the Self-Organizing Map follows the "garbage in - garbage out"' principle, which is why preprocessing of data is required. A few ways to pre-process data are as follows:

1. Focus on a particular subset of data
2. Removing erroneous data
3. Scaling data to have unit variance
4. Data encoding

Intuition behind SOM

- Start by having an NxN grid.
- Each node has weights for input variables in the data
- Based on weights and data points, we find the best matching unit(BMU) in the grid for each data point
- Iteratively update the weights of BMU and the neighborhood nodes
- The size of the neighborhood around the BMU is decreasing exponentially. At the end of the training, the neighborhood size shrinks to zero size.
- Similar data points will naturally be in the same nodes.

Variable Notation

s is the current iteration

t is the index of the target input data vector in the input data set D

D(t) is the target input vector

W(s) is the weight of a particular node at s-th iteration.

L(s) is the learning rate given by $L(t) = L \exp(-s/\lambda)$

Ɵ(s) is the influence rate.

Algorithm

1. Randomize the node weight vectors in a map between 0 and 1.
2. Choosing a random input vector D(t) from the training dataset.
3. Traverse each input vector in the input data set
    - Traverse each node in the map:
        - Each node is examined to find the one whose weights are most similar to the input vector. Here we use Euclidean distance for finding similarity. This unit is called as Best Matching Unit(BMU)

    - Calculate the size of the neighborhood around BMU and update the weights of all the neighborhood nodes(including BMU)
        - $W(s+1) = W(s) + \Theta(s)L(s) (D(t) - W(s))$

4. Increase s and repeat from step 2 while $s < \lambda$.

Methodology

We use the continuous data acquired by Group 4 and Group 5 for training purposes. We got data of **9 students** and we assigned each student as one class. This task can be thought of as a **one of** classification task in which given the key stroke and latencies, we have to classify it as one of the classes on which it model has been trained. We use certain data pre-processing techniques followed by training of our model using Self Organizing Maps. We report the classification accuracy of this task through a K-fold mechanism where K = 5, which implies that we divide the entire dataset into 5 sections, we hold out one section for testing and the other 4 are used for training. The final accuracy is reported as the average over all hold out sets.

Libraries Used:

1) ***json*** (to save, load, get pre preprocessed data)
2) ***numpy*** (for vector operation)
3) ***sklearn*** (for k-fold split, evaluation metrics)
4) ***susi*** (for Self Organizing Map)
5) ***matplotlib*** (For Plots)

Feature Extraction

We run the code provided to us for feature extraction, given the continuous data collected and the extracted features are as follows:

1) We get 26 hold time values for each of the letters from A-Z.
2) We get 26*26 latency values from (AA-ZZ)

We take these 26 + 26*26 = 702 values as a feature vector for one class and use it to classify from several classes. We take the average of all hold time and latency values for one particular key or one particular pair of keys as this helps in smoothing the values and returns an accurate representation of the several values.

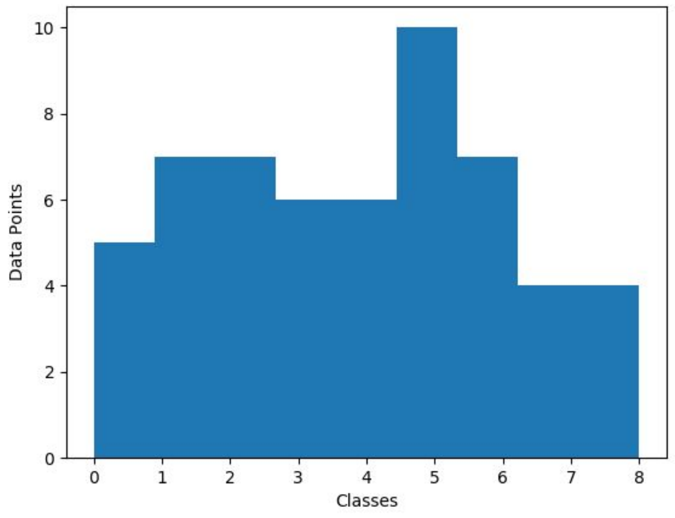
Hold time: It tells us the length of the keypress or the time difference between the press time and the release time. It signifies the impulsive nature of the user.

Latency: It tells us the time to switch from one particular key to the other. It signifies the switching and typing speed of the user.

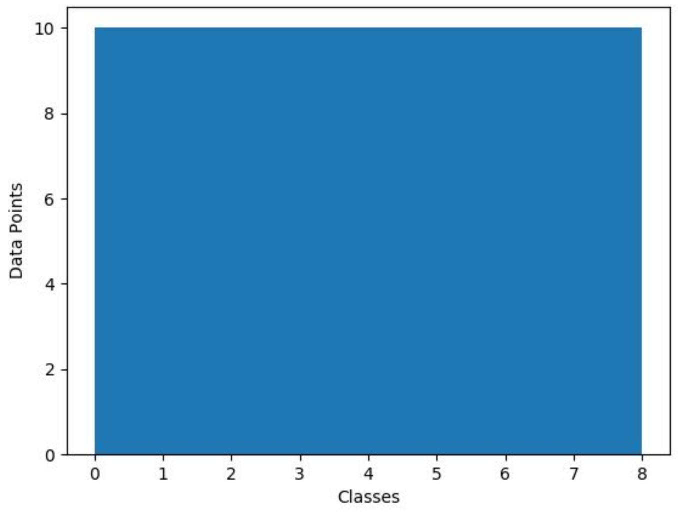
These two features would put surveillance on the user and uniquely identify the user.

Imbalanced Classes

After extracting the features for the various classes, we had 56 data points in total, which is quite less to obtain significant results. Moreover, the data was quite imbalanced with an unequal amount of data points available for each class. We first show results on this unbalanced class dataset and then we use techniques to solve this class imbalance problem by obtaining more data for the poorly represented classes. We also generate some additional data points by taking the moving average of existing data points for the poorly represented classes.



Imbalanced Classes (lesser number for class 7-8)



After balancing classes

Results

We vary the following hyperparameters of the Self Organizing map to try and find the optimal parameters which work best on the test set. For all results, we return the 5-fold accuracy on the test set.

1) N_rows
2) N_columns
3) N_iterations_supervised
4) N_iterations_unsupervised

Higher number of N_rows and N_cols improves expressibility of the function. But for too high values, we require a substantial amount of data, which is not present in our case. The number of iterations also varies with the amount of data available, in our case we found 10000 iterations to be sufficient to learn the features well.

N_rows	N_cols	N_iter_super	N_iter_unsuper	K-Fold Accuracy	At random
50	50	1000	500	39.24	11.11
50	50	10000	5000	55	11.11
50	50	100	50	22.8	11.11
40	**40**	**10000**	**5000**	**62.72**	**11.11**
5	5	10000	5000	28.90	11.11
10	10	10000	5000	39.20	11.11

Varying Hyper-parameters on Imbalanced dataset

N_rows	N_cols	N_iter_super	N_iter_unsuper	K-Fold Accuracy	At random
50	50	1000	500	60	11.11
50	**50**	**10000**	**5000**	**83.33**	**11.11**

50	50	100	50	42.22	11.11
40	40	10000	5000	81.01	11.11
5	5	10000	5000	47.77	11.11
10	10	10000	5000	77.77	11.11

Varying Hyper-parameters on balanced dataset

The results for the optimal parameters which we found on the Imbalanced class dataset are the following:

Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
50	81	63	55	63	**62.72**

The results for the optimal parameters which we found on the balanced class dataset are the following:

Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
83	77	88	77	83	**82**

User Authentication and Imposter Detection

Once we have trained this Self Organizing Map, it can be used to detect imposters and authenticate corresponding users. This is done by collecting the continuous stream of data for a user and passing it through our trained model. Then we predict the corresponding class for this user and if it is equal to the actual class, we authenticate this user else we label it as an imposter.

We took our group data as the user and the other group as the imposter, and tested our model on the imbalanced test set:

Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
0.83	0.91	0.82	1	0.63	**0.84**

Analysis of Results and Conclusion:

1) We observe that if we try to select a class at random, the probability that we select the correct class is only 11.11% (1 out of 9, 1/9). However, we improve this accuracy to **59%** in the case of the imbalanced dataset and **83%** in case of the balanced dataset.
2) Imbalanced classes are a major source of poor performance of classification models and they can be solved using oversampling (the class with lesser number of points), undersampling(the class with more number of points) and generation of new data points from existing ones. We employ the latter.
3) The hold time and latencies of a user act as a signature and this 702 dimensional vector is able to uniquely identify the user with a decent accuracy.
4) One major drawback while training this model was the lack of data. In supervised machine learning, the amount of data greatly affects the performance of our model. Thus while our model was not able to perform very well on the Imbalanced dataset, the performance can be greatly enhanced by collecting a good amount of data for each user.
5) We initially had 10 classes but had to drop one class due to the lack of data in it.

References:

SOM Algorithm : Tonny J. Oyana, Luke E. K. Achenie, Joon Heo, "The New and Computationally Efficient MIL-SOM Algorithm: Potential Benefits for Visualization and Analysis of a Large-Scale High-Dimensional Clinically Acquired Geographic Data", Computational and Mathematical Methods in Medicine, vol. 2012, Article ID 683265, 14 pages, 2012. Link

SOM Algorithm Article : Wikipedia Link

Data preprocessing : Jaakko Hollmen Fri Mar 8 1996 Link

Hyperparameter tuning : The commonly used hyperparameter settings are taken from [RieseEtAl2020]. Link

Imbalanced Classes: Blog on Towards data science to deal with imbalanced classes Link