

Keystroke Dynamics for Authentication

Data X 1 Project Lab Report

Team: Karthika, Mahathi, Mayank, Susmeet



Problem Statement

Two factor authentication is recommended by most major consumer facing applications. However, the extra step, often entering a One-Time-Password, is cumbersome as it adds delay and involves the use of an extra application, usually text message/email, in order to log-in to the original application. This also imposes a lot of cost on the companies. We propose a solution that can replace the existing means of second factor authentication, bring down the costs involved, eliminate external dependencies like network access and with no extra hardware needed.

Proposed Solution

One way to tackle this problem is to incorporate the second factor of authentication into the typing of the password itself. One approach is to use the rhythm of the typing of the password itself as a unique signature of an individual and use this as the second factor of authentication. This rhythm or pattern of typing is known as keystroke dynamics.

After trying various standard approaches, the details of which you shall find in this document, **we have come up with a novel approach that uses Siamese Networks to learn the keystroke dynamics of individuals** and classify a person's typing pattern as having matched or not-matched the target individual's keystroke dynamics.

About the Dataset

We used a dataset from researchers at Carnegie Mellon University¹. We later expanded on this by generating data ourselves. The initial dataset contains responses from 51 individual users (typists) each typing a password (*.tie5Roanl*) 400 times. We then added 100 observations to this dataset by having 5 of our fellow mates entering the password 10 times each. We observed the rates were quite different for the same person because of the inconsistency in their typing so, now we are still collecting data by making them type for at least 20 times. After which we will use it on our model that is pre trained on the online CMU dataset mentioned above.

The dataset has 34 columns of which 3 contain identifying information and 31 contain relevant data. The identifying columns contain subject, session in which they recorded data and repetition number within each session. The three columns together uniquely identify each observation.

When the data was collected, both by CMU and later by us, key press and key release times were recorded. The variables described below were then computed. Variables containing timing information:

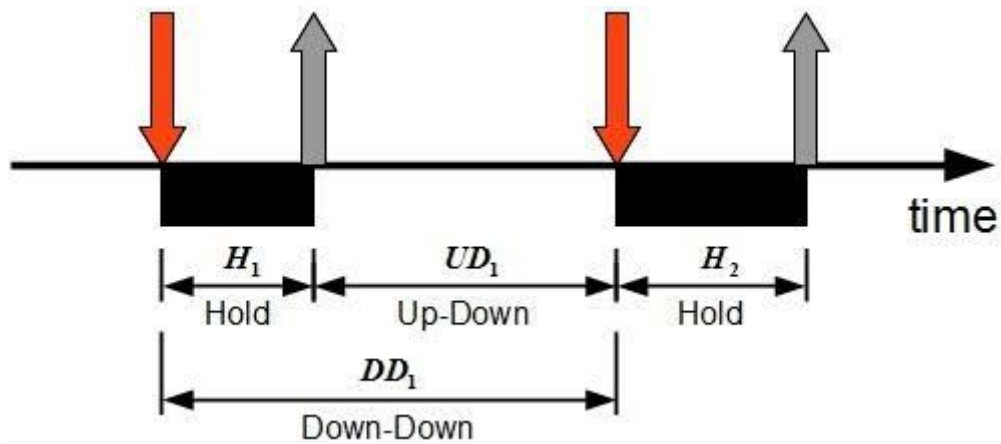
H.key - These variables denotes the hold-time for a *key*. This is the time between first pressing the key and realizing it.

DD.key1.key2 - This records the time elapsed between the instance when *key1* was pressed and the instance when *key2* was pressed.

UD.key1.key2 - This records the time elapsed between the instance when *key1* was released and *key2* was pressed.

Note: H time and UD times add up to DD times.

¹ Ali, M.L., Monaco, J.V., Tappert, C.C., Qiu, M.. Keystroke biometric systems for user authentication. *Journal of Signal Processing Systems* 2017;86(2-3):175–190.



Credits: <https://appliedmachinelearning.blog/2017/07/26/user-verification-based-on-keystroke-dynamics-python-code/>

Data Analysis / EDA

Types of features

We are collecting 3 distinct types of features from a user's keystroke patterns.

- The features starting with a **H** contain the hold time for different keys.
- The features with initials **D.D** contain the time gap b/w pressing two keys.
- The features with initials **U.D** contain the time gap between lifting one key and pressing another.

Let us explore each category of features one by one.

Hold times

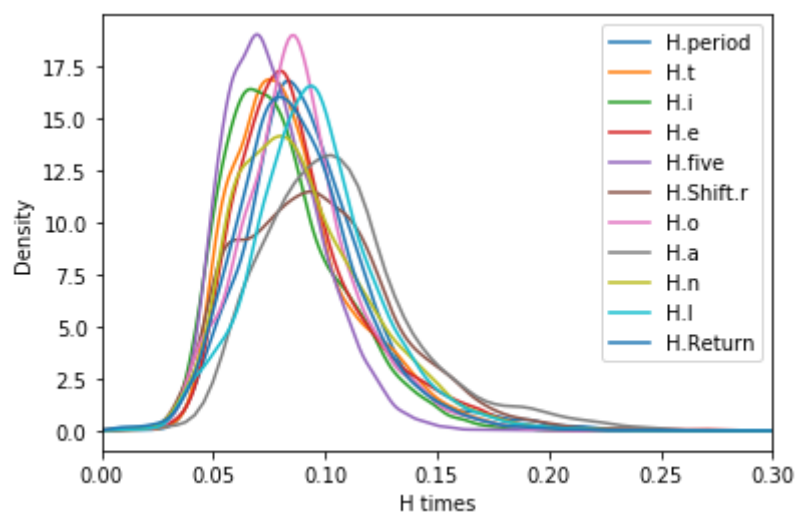


Figure: Plot of all the different Hold times with the number of user samples

So the distribution of press times for all the keys is more or less similar across different users across different conditions, which is not surprising. This alone can't be used to detect keystrokes as the user distribution is also very similar.

Down-Down(DD) times and Up-Down(UD) times

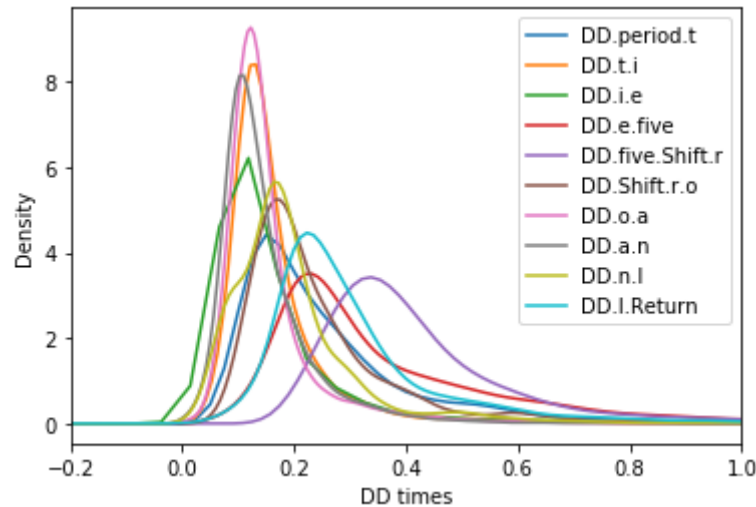


Figure: Plot of all the different DD times with the number of user samples

There is a lot of visible variation across down-down times.

- Most significant right skewness can be observed when the users have to type the shift key for a capital R after pressing 'five'. As you would expect.
- Individual distributions for different down down times by themselves have a lot of variations, which shall make them a great differentiating feature for authentication.

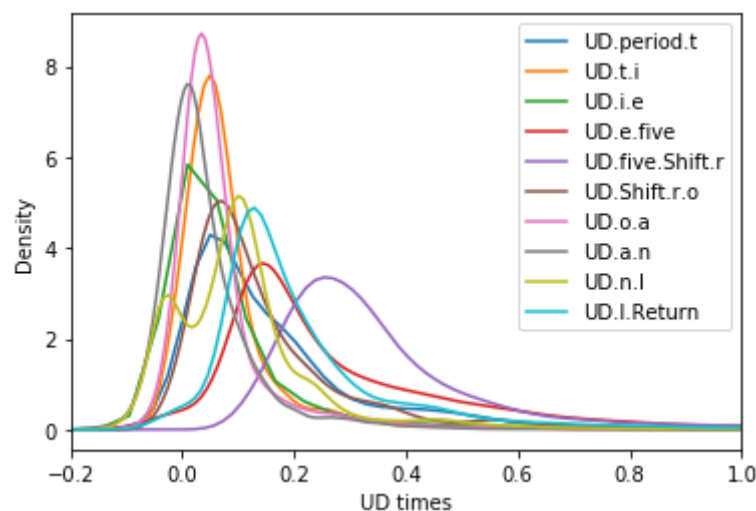


Figure: Plot of all the different UD times with the number of user samples

The variations in Up-Down times is surprisingly similar to down-down times. With one major difference being - a lot more negative values. This is extremely valuable. Some people perhaps have a typing style where they press the next key even before lifting up their fingers from the previous key. These kinds of subtle patterns are what are classifiers are going to pick up on!

Negative UD Times		
	Number	Percentage
UD.period.t	903	4.43%
UD.t.i	1943	9.52%
UD.i.e	4088	20.04%
UD.e.five	559	2.74%
UD.five.Shift.r	0	0.00%
UD.Shift.r.o	832	4.08%
UD.o.a	2644	12.96%
UD.a.n	6487	31.80%
UD.n.l	4190	20.54%
UD.l.Return	472	2.31%

Figure: Distribution of different negative values in the UD times in the data

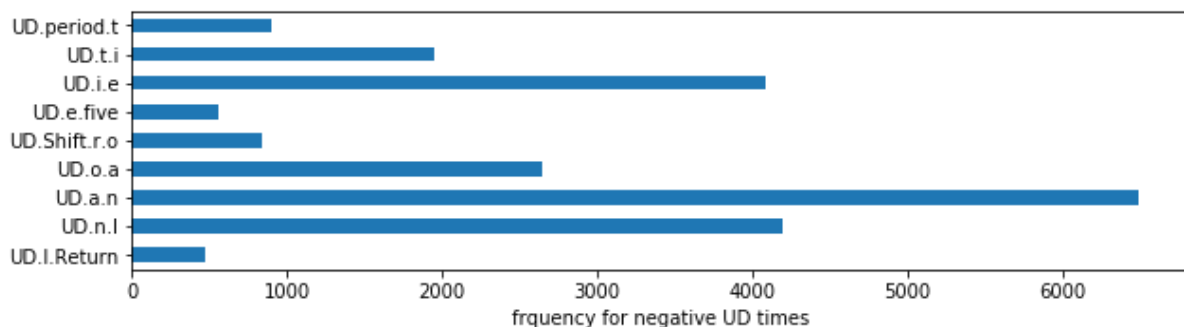


Figure: The above Table represented in a graph

The bar plot and the table above capture the number of times UD times were negative. The plot is in order in which the keys are typed for the password. There is a lot of variation across these frequencies between different pairs of keys. Surprisingly, a lot of users do not lift their fingers from 'a' before they press 'n'. This could be a result of typing different keys from your left and right hands. This shall be an interesting exploration for a later point in time.

Baseline Model

We have implemented logistic regression, one-class SVM, and Neural Network as our baseline models. The testing and training data was prepared ensuring equal samples of all the classes and these models were tested on the same data. Given below is the description and performance of those models.

Model 0 - Logistic Regression

As logistic regression is one of the simplest models for multi-class classification, the data was tested on this model. The following is the network model and the performance of it.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 51)	1632
Total params: 1,632		
Trainable params: 1,632		
Non-trainable params: 0		

Figure: Network Architecture of Logistic regression

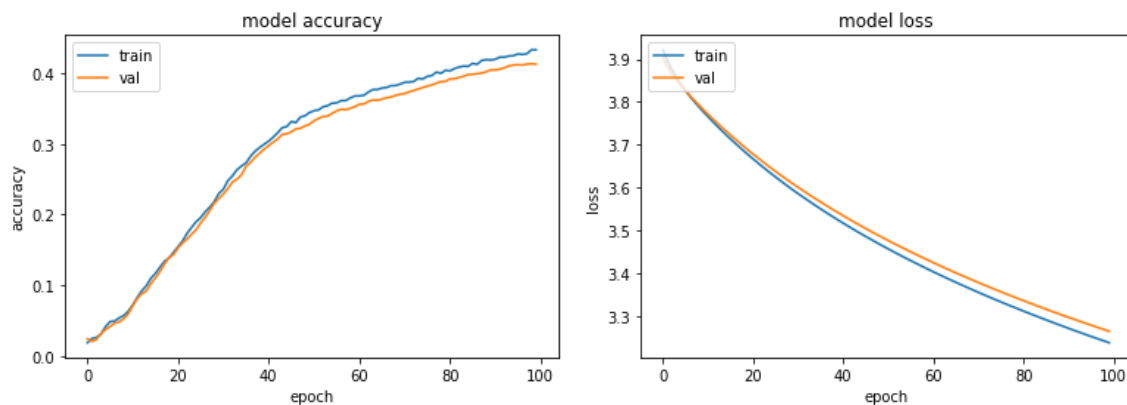


Figure: Performance of the Model

The model gave an accuracy of **42.03%** on the training data and hence was rejected immediately as a baseline model.

Model 1 - One-class SVM

A method for this task, which gained much popularity in the last two decades, is the One-Class Support Vector Machine.

It basically separates all the data points from the origin and maximizes the distance from this hyperplane to the origin. This results in a binary function which captures regions in the input space where the probability density of the data lives. Thus the function returns +1 in a “small” region (capturing the training data points) and -1 elsewhere.

We used a one-class SVM model as our baseline as this model is often used in anomaly detection use cases.

Equal error rate (EER) is a biometric security system algorithm used to predetermines the threshold values for its false acceptance rate and its false rejection rate. When the rates are equal, the common value is referred to as the equal error rate. The value indicates that the proportion of false acceptances is equal to the proportion of false rejections. The lower the equal error rate value, the higher the accuracy of the biometric system.

The EER we found for this model is 17.56% and hence the accuracy is **82.43%**.

Model 2 - Neural Network

We used a neural network next.

Different combinations of number of layers and number of neurons were tested to get the best results. The final model used has the following architecture:

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 30)	960
batch_normalization (Batch Normalization)	(None, 30)	120
activation (Activation)	(None, 30)	0
dense_1 (Dense)	(None, 25)	775
batch_normalization_1 (Batch Normalization)	(None, 25)	100
activation_1 (Activation)	(None, 25)	0
dense_2 (Dense)	(None, 51)	1326
Total params: 3,281		
Trainable params: 3,171		
Non-trainable params: 110		

Figure: Network Architecture of Neural Network used

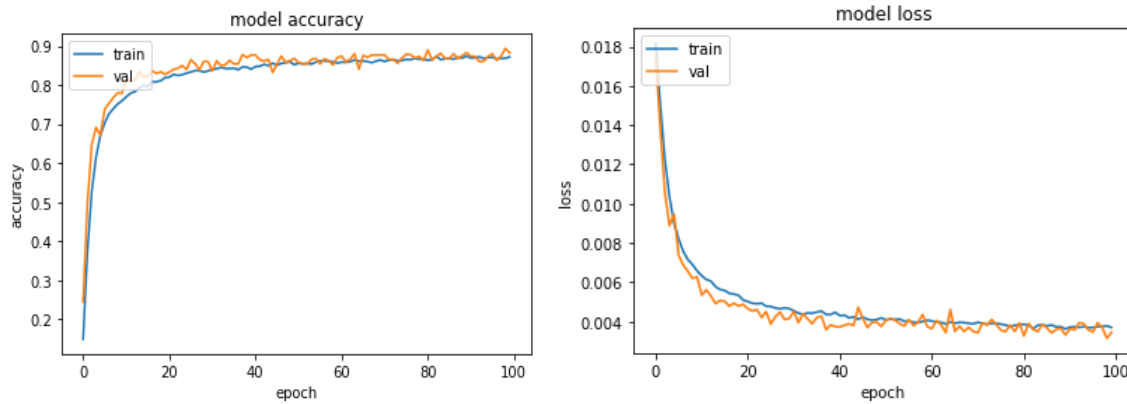


Figure: Performance of the Model

The model produced a testing accuracy of **88.35%**. This was an upgrade compared to our previous model and hence became the new baseline.

Current Solution

Overview

Our current system is developed using Python. The user inputs user-name and user-password in a simple pop-up window. In the backend, keystroke dynamics are recorded using the 'pyxhook' keylogger python library for linux while the user types the password. This keystroke is compared against the keystroke present in the database against the username. If it is the keystroke of the same person the model prediction is 1 showing it is the genuine user and if not the output is 0 implying an imposter. This 0 and 1 is finally converted to a simple display stating Access Granted/Denied. The GUI uses tkinter library for our prototype however we are also exploring pyqt as it provides more flexibility. Further iterations will involve a more sophisticated UI/UX.

Model

Siamese Neural Network model is the final decided upon model to be used because of its significance and also better performance in user identification.

Siamese Network is a Model Architecture used alongside a Distance-based Loss. It is like learning in one shot. It learns what makes 2 pair of inputs the same (e.g. dog-dog, eel-eel). In Comparison, Classification learns what makes an input a dog/ cat/ eel etc. It contains two or more identical subnetworks. identical here means they have the same configuration with the same parameters and weights. Parameter updating is mirrored across both subnetworks. Siamese NNs are popular among tasks that involve finding similarity or a relationship between two comparable things. Sharing weights across subnetworks means fewer parameters to train for, which in turn means less data required and less tendency to overfit.

Simply putting, Siamese network compares two keystrokes and tells if it is of the same user or not. This is significant because in real-life application if in an authentication system where keystrokes of all the users are saved, for any other model it will be compared with each of the users to find the right one whereas in Siamese network it will be just comparing with one particular keystroke to give out the prediction.

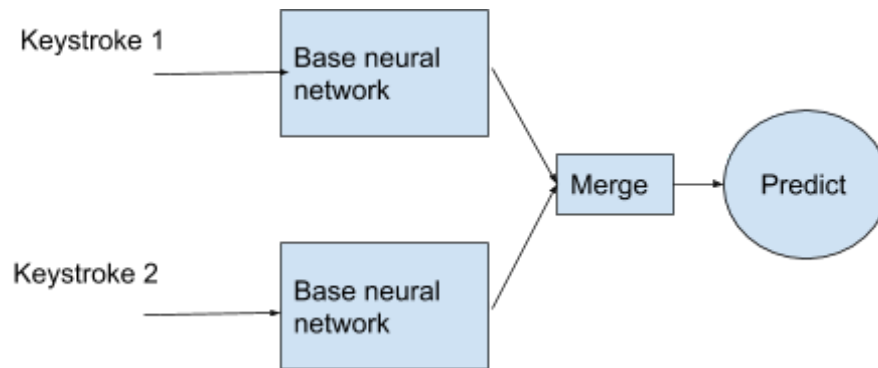


Figure: Basic Siamese Model

Dataset: From the existing database, 12000 genuine user pairs labelled as 1 and 12000 imposter user pairs labelled as 0 were prepared for training. We basically kept on increasing the training data in order to get a testing accuracy above 90% which is desirable for an authentication system.

Improving model performance:

- The initial architecture as shown above used loss function as contrastive loss, this compared to adding 2 more dense layers ahead of the merged layered that calculated the distance between the two feature vectors with a binary_crossentropy loss performed better.

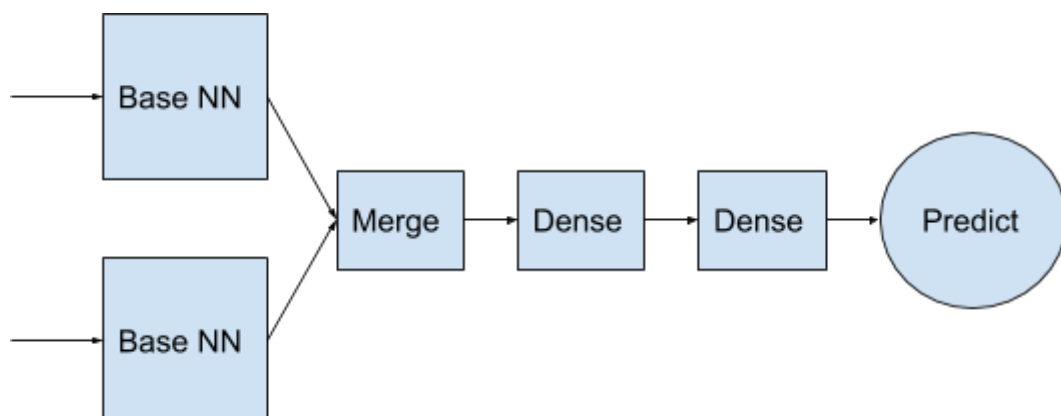


Figure: Modified Siamese Network

- We also tried 3 different distance functions for the Lambda layer Cosine_distance, absolute difference and euclidean distance. Euclidean distance seemed to perform better in terms of accuracy and faster convergence. Although other 2 also performed almost equally well. This can be further explored and quantified.

```
# 3 different distance function that can be tried for the Lambda Layer

def cosine_distance(vests):
    x, y = vests
    x = K.l2_normalize(x, axis=-1)
    y = K.l2_normalize(y, axis=-1)
    return -K.mean(x * y, axis=-1, keepdims=True)

def abs_diff(x): return K.abs(x[0] - x[1])

def euclidean_distance(x):
    return K.sqrt(K.sum(K.square(x[0] - x[1]), axis=1, keepdims=True))

diff = Lambda(function=euclidean_distance,output_shape=lambda x: x[0])([feat_vecs_a, feat_vecs_b])
```

Figure: Code Snippet of the distances used in the Siamese Model

- Adding batch normalisation layer also improved the accuracy fairly from 80 to 90%. So far, our model has achieved around 92% accuracy on testing data and our architecture looked like below:

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 31)]	0	
input_2 (InputLayer)	[(None, 31)]	0	
sequential (Sequential)	(None, 25)	9825	input_1[0][0] input_2[0][0]
lambda (Lambda)	(None, 1)	0	sequential[1][0] sequential[2][0]
dense_4 (Dense)	(None, 10)	20	lambda[0][0]
batch_normalization_2 (BatchNor	(None, 10)	40	dense_4[0][0]
activation_2 (Activation)	(None, 10)	0	batch_normalization_2[0][0]
dense_5 (Dense)	(None, 1)	11	activation_2[0][0]
Total params: 9,896			
Trainable params: 9,726			
Non-trainable params: 170			

Figure: Network Architecture of the Siamese Model used

UI workflow

We would more clearly like to describe how we intend our system to work in this section.

Two factor authentication with
Keystroke Dynamics

Username :

Password :

The above image is how our UI would look like.

Two factor authentication with
Keystroke Dynamics

Username :


Password :

Authenticated!

Mahathi is a user who uses our authentication system, through the multistage process described in the following section, her keystroke dynamics are tested positive and she gets authenticated.

Two factor authentication with Keystroke Dynamics

Username :

Password : 

Authentication Error : We do not think it's Mahathi typing

Mahathi's friend Karthika is able to guess her (obvious) password and she tries to break in using her credentials. Our system detects that her dynamics do not match and does not let her in.

Python GUI as the screenshots display above were developed using python GUI framework. Still we are working on developing this. At the backend the challenge we faced was to invoke the keystroke capturing script, because of which the UI is not completely made. Hence we have the UI, data collection script and the model script working independently now, and the integration of which will be done in the next 2-3 days that we hopefully can demo.

Performance Evaluation

We are satisfied in terms of using a novel approach of Siamese Neural Network. Before which we also tried several other models. The accuracy given out by Siamese Neural Network satisfied our hypothesis of that performing better than the rest of the models. This performance can be improved further as depicted by the loss vs epoch and accuracy vs epoch graph.

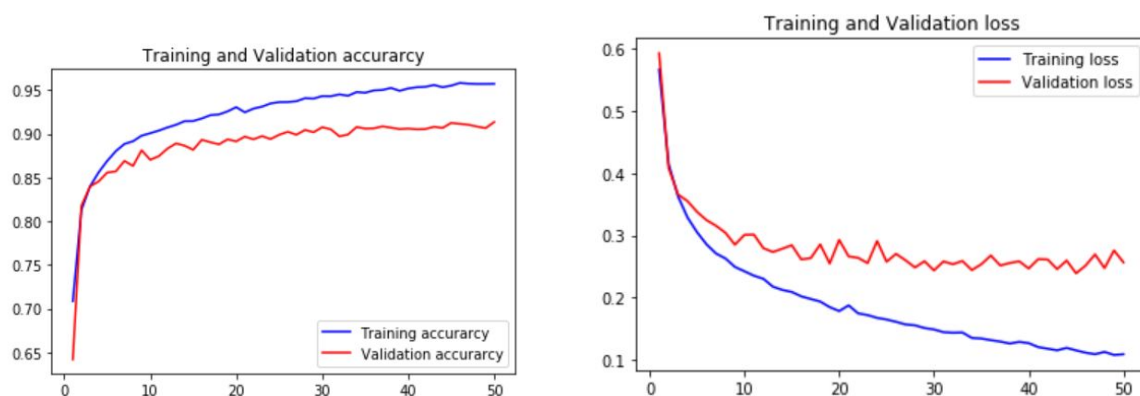


Figure: Performance of Model

We realised for this system to work efficiently, False Positive predictions needs to be penalised more for which we will have to create our own asymmetric loss function. Temporary work around for this was to overfit the data upto some degree. The reason is when we try to over fit the data with the classifier the classification bound tends to wrap around the clusters very tightly and with that model we can sometimes miss classify positive classes as negative(due to high variance) but there are comparatively less situations where you end up miss classifying negative classes as positive.

We are yet to be evaluating our model on our own collected data samples which again we hope to achieve soon.

System Architecture Overview

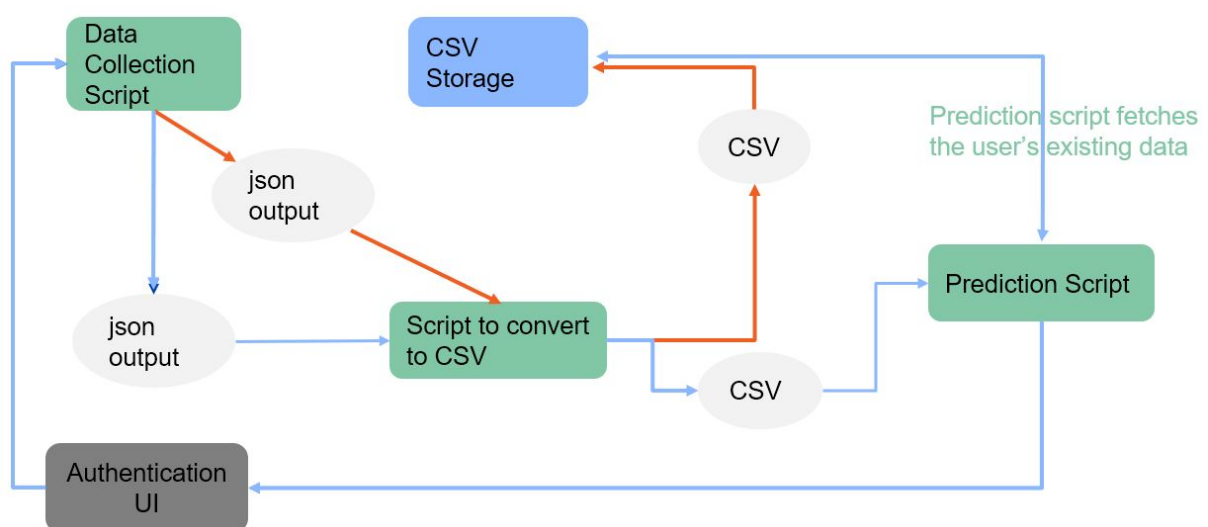


Figure: Architecture Overview

In the architecture, you would find two separate pipelines, the orange arrows correspond to our data collection pipeline, and the blue arrows form the authentication UI framework.

Data Collection pipeline

We have a fully functional but very primitive data collection pipeline as of now. We run the data collection script from the terminal, it records the keystroke dynamics of a user entering the password, gives out a Json output that gets consumed by our script to convert to Csv, which appends the data in csv format to our main csv file.

Authentication Framework

The user inputs the username and password into our UI. As the user inputs this, our data collection script is activated, it generates the user's keystroke dynamics and outputs it in a json file once the user is done. This Json file gets consumed by our script to convert to csv that then sends this csv data to our prediction script. The prediction script identifies the

username from this data and fetches existing data for this user from our csv storage. Then these two data points are fed into the Siamese model which predicts if these belong to the same user. This result is then outputted back to our UI in the form of authentication being granted or rejected. All of this works in a sequential manner except for the keystroke capture part that gets invoked as and when the user types a key using multi threading.

Key Findings

Visualizing dynamics for different users using t-SNE

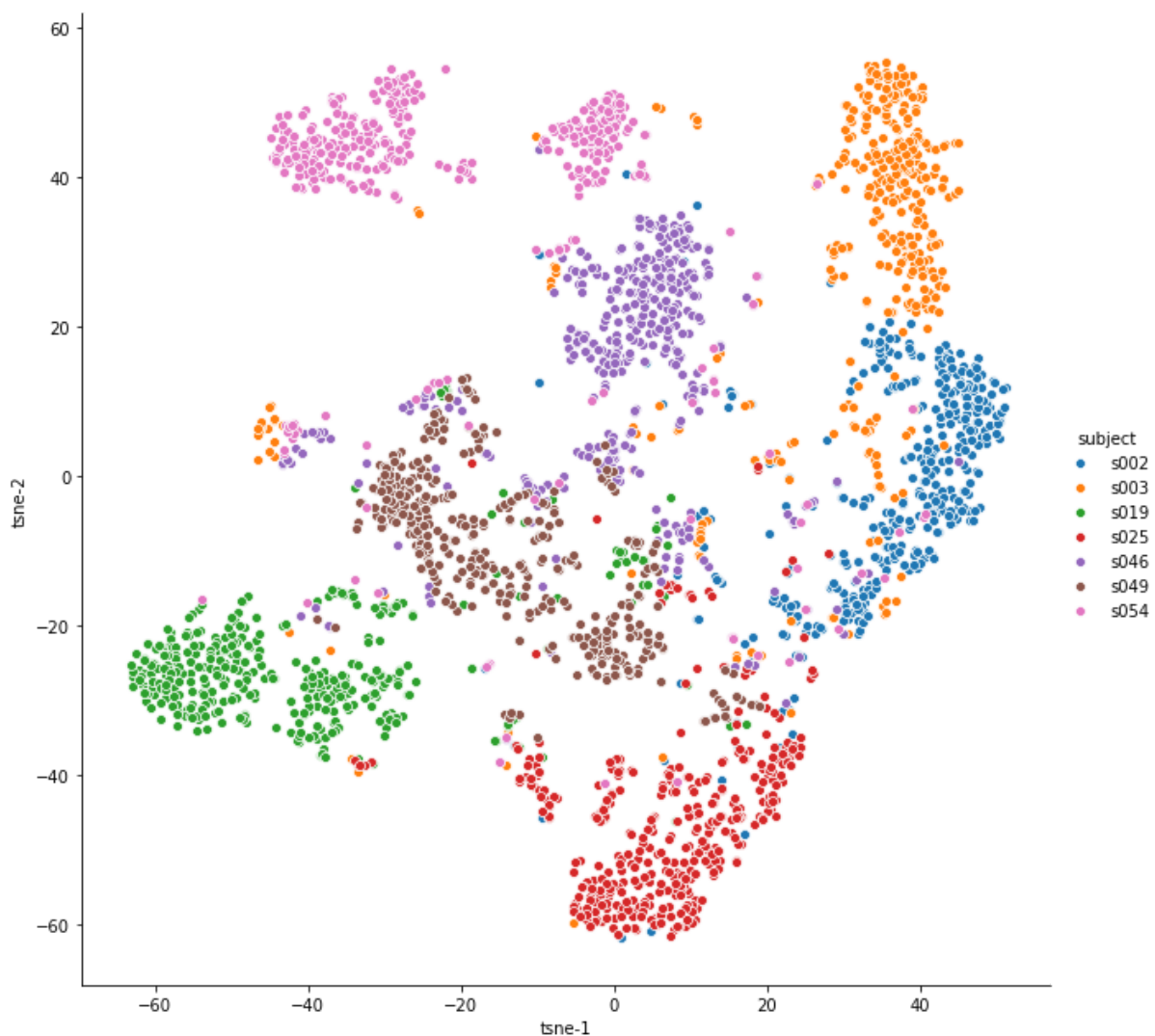


Figure: t-SNE plot for some random subjects

Motivation for t-SNE

On doing some simple error analysis, we found that our model gets confused between two subjects, s002 and s003. This motivated us to visualize the data using t-SNE. And indeed we

found that these two users have very similar typing styles. One of our key findings is this, if two users have very similar typing styles than it is (as of now) difficult for our network to work efficiently. One possible solution to this problem could be to use a more complex/deeper network.

```
# through EDA observation from t-SNE plot we find s002 and s003 subjects forming clusters
# nearby and quite overlapping, telling us their patterns being quite similar, and this model
# finds it actually difficult to differentiate both giving us prediction of 1 that is both same

x1 = np.array(data.loc[data.subject == 's003', 'H.period': 'H.Return'][1:2])
x2 = np.array(data.loc[data.subject == 's002', 'H.period': 'H.Return'][1:2])
pred = loaded_model.predict([x1,x2])
np rint(pred)

array([[1.]], dtype=float32)
```

Another interesting find

The subject s054 seems to practice two very distinct typing styles as seen in the visualization above. And on further analysis we did find that our classifier finds it difficult to distinguish between these two typing styles.

```
# another case as seen from t-SNE plot done in EDA shows s054 subject has 2 very different styles of typing
# hence x1 and x2 give 0 that is different while x1 and x3 give same that is 1

x1 = np.array(data.loc[data.subject == 's054', 'H.period': 'H.Return'][1:2])
x2 = np.array(data.loc[data.subject == 's054', 'H.period': 'H.Return'][25:26])
x3 = np.array(data.loc[data.subject == 's054', 'H.period': 'H.Return'][250:251])
pred = loaded_model.predict([x1,x2])
print('x1 and x2: ', np rint(pred))
pred = loaded_model.predict([x1,x3])
print('x1 and x3: ', np rint(pred))

x1 and x2:  [[0.]]
x1 and x3:  [[1.]]
```

Model selection

1. Baseline Models:

While trying out different models, playing with different optimizers and loss functions, we found that ‘adam’ as an optimizer and ‘mean_squared_error’ as loss function gave better results.

Also, using BatchNormalization helped improve our results.

2. Siamese with adding dense layers after the distance based lambda layer gave better performance and Euclidean distance seemed to work marginally better than cosine distance and absolute difference.
3. When tested on so far collected data for example Mahathi’s keystroke, that we matched against some random sample from the corpus which is not her keystroke the model predicted 0 which means it will not be authenticating but when we compared with another keystroke of Mahathi’s itself, that also predicted 0 that provoked us to

explore our collected dataset. We realised no one really typed the password in a set rhythm because it was a new password to type for them. So we are still collecting data but now collecting more samples for each.

Future Possibilities

While working on our project, we realised the potential keystroke dynamics hold

- This could be used to identify people with similar key typing patterns. It could be interesting to explore if these similarities correlate with something real and then perhaps, could be used to connect similar types of people with each other for personal/ professional purpose.
- We can further analyse keystrokes to determine the blood group of a person, age of a person and even something like lie detection and fatigue detection.
- One of our team members, Mahathi is really interested in using these dynamics to understand the overall well being of a user. The way a person is using their devices might be indicative of their mental well being.

Bottlenecks/Challenges

Making UI/UX talk to Backend

We lacked an individual experienced in UI and frontend. This combined with our hampered motivation (which we describe in a section below) lead to delayed work, and we still do not have professional quality UI and an overall connected system. We have individual pieces that work independently but those pieces are not properly connected.

Time Management

With many competing priorities during the course of the term, we struggled to allocate time evenly across the term. The work often took place in spurts with tasks being assigned individually. However, we rarely made progress till we met collectively. Hence, a team management practice we employed towards the end was to find time to sit together and work on individual tasks to ensure work gets done.

Lack of Alignment

Our team formed based on an intention to work on behaviour change for social impact. We reached out to the Center for Social and Behaviour Change at Ashoka University to collaborate on projects that were being supported by NITI Aayog - India's apex think tank. After a three hour long meeting with the Director of the center, and concrete next steps for knowledge exchange through a series of three workshops, we left motivated and decided to form the team.

However, after multiple attempts to reach them again over phone calls, mutual connections, emails and text messages, we received no response from them. After three weeks of pursuing this, we decided to abandon the idea. In the meanwhile, one of our team mates also left the team for another.

For our low-tech demo, we rallied to generate an idea that resided at a weak intersection of our interests, and presented the report within two days. However, after the demo, the team was not sold on the project idea. To build alignment, we split up to come up with individual ideas and opened ourselves to all possible domains. However, we were unable to align everyone and the rest of the project saw two members of the team, who were friends outside the project, drove most of the project.

The key learning from this experience was that forming teams based on singular ideas with no other prior affinity can be challenging. This inhibits flexibility and alignment down the line in case a pivot away from the initial idea becomes a necessity.

GitHub Link

https://github.com/SusmeetJain/keystroke_auth