A Project Report on

# KEYSTROKE DYNAMICS BASED PASSWORD AUTHENTICATION

*Submitted in partial fulfilment of the*

*requirements for the award of the degree*

*of*

**BACHELOR OF TECHNOLOGY**

*in*

**ELECTRONICS AND COMMUNICATION ENGINEERING**

Submitted by:

| **Anshul Dutt Sharma** | **Ayush Shrivastava** |
|---|---|
| 18116016 | 18116020 |

Under the guidance of:

**Professor Dharmendra Singh**

Department of Electronics and Communication Engineering

Indian Institute of Technology Roorkee

May 2022

# DECLARATION

We hereby declare that the work which is being presented in this report entitled **"KEYSTROKE DYNAMICS BASED PASSWORD AUTHENTICATION"** in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in **Electronics and Communication Engineering** to the Department of Electronics and Communication Engineering, Indian Institute of Technology Roorkee is an authentic record of our own work carried out under the supervision of Dr. Dharmendra Singh, Department of Electronics and Communication Engineering, Indian Institute of Technology Roorkee. This work has been done during August, 2021 to May 2022. We have not submitted the matter embodied in this report for the award of any other degree or diploma.

Date: 1st May 2022

Place: Indian Institute of Technology Roorkee

**Anshul Dutt Sharma** 18116016

**Ayush Shrivastava** 18116020

# CERTIFICATE

This is to certify that the project report entitled **"KEYSTROKE DYNAMICS BASED PASSWORK AUTHENTICATION"** submitted by *Anshul Dutt Sharma* and *Ayush Shrivastava* to the Department of Electronics and Communication Engineering, Indian Institute of Technology Roorkee toward partial fulfilment of the requirements for the degree of **BACHELOR OF TECHNOLOGY** in **Electronics and Communication Engineering** is a record of bonafide work carried out under my supervision and guidance.

Date: 1st May 2022

Place: Indian Institute of Technology Roorkee

**Professor Dharmendra Singh,**

Professor,

Department of Electronics and Communication Engineering,

Indian Institute of Technology Roorkee

# ACKNOWLEDGEMENT

We would like to express our very great appreciation to Professor Dharmendra Singh for his constructive feedback during the execution of this research work. His willingness to give his time so generously has been very much appreciated. His guidance and constant motivation even in the toughest of times were main reasons for our completion of the project. We would like to thank our friends who helped us constantly through our project for Keystroke data collection and support. We would like to express our gratitude to this college IIT Roorkee for providing us the appropriate platform to execute our research and lastly, we would like to express our thanks to our parents who constantly supported us even in the most difficult times.

# ABSTRACT

Keystroke dynamics is a secured data processing technique that analyses the detailed timing information on when and how a key is pressed when the user is typing on the keyboard. It is based on the idea that each user has a unique typing rhythm just like his fingerprint. We have created a fully-fledged working model that takes in inputs as the timing information of the users passwords. We then train the model based on the user's keystroke registration data and predict whether the user is genuine or not when he logs into the system next time. In this thesis, we provide a detailed elaboration beginning from how we designed the system followed by the workflow of the user interface and the description and detailed analysis of the Deep Learning models used for training the keystroke data. We conclude by providing an analysis of our system and how it could further be improved and adapted into real time scenarios.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# Chapter 1
# Introduction

## 1.1. Objective

Our work primarily deals with analyzing behavioral uniqueness of the user and creating a fully-fledged working interface (Command Line Interface) based on it. This model is incorporated alongside password-based authentication to enhance the security of the system. The hacker will now not only need to know the exact password but also the exact manner as to how it was typed. At the same time, for every successful login attempt a user makes, more keystroke information becomes available and the authentication model becomes more efficient exponentially. Dynamic datasets are introduced to help prevent the dataset from getting stale, and to make sure we have latest typing habits of the user stored in the system. If the model is unable to provide a very accurate score, we move on to other 2 factor authentication methods like OTP verification which is covered in detail later.

For designing the model, we have used a Deep Learning based approach which is mostly used in computer vision and NLP related work. We create 2 models XGBoost and MLP that are used to provide a correlation score that denotes the correlation between the user's registration keystroke sample and the timing pattern that is stored while logging into the system. The correlation must be high for the user to login using Keystroke Dynamics model or else we use OTP verification.

## 1.2. Motivation and Background

The demand for security has been increasing on a daily basis. With the current technology, it's not very difficult to compromise a username and password of a person and get his private credentials. This can be disastrous when only a password/PIN is required to hack into a user's system for example ATMs, bank vaults, etc. Hence, there is a need for a more robust mechanism to verify the user's identity that goes beyond the usual password-based authentication. To counter this problem, two-factor authentication systems have recently been introduced in many areas. Applications where the user's authenticity and security are of top priority such as net banking and Emi websites are actively depending on two-factor authentication. However, entering a one-time password every time the user tries to login is a strenuous task. We need to find answers to questions like, when exactly is two-factor authentication necessary to identify the user, and is two-factor authentication system secure enough? This is where Keystroke Dynamics comes into picture. The core concept used in the analysis of Keystroke dynamics is that each individual has a certain rhythm and manner with which he types characters on a keyboard which is unique and can be defined as the characteristic of the user. Similar to fingerprints and retina scans, this unique behavioral information of the user can be used for authentication purposes. The advantages of using a keystroke dynamics-based approach is manifold, firstly no complicated hardware is required to record keystroke data as opposed to fingerprints and eye scanning softwares which requires a hardware setup to be installed wherever put to use. The Keystroke data can be gathered using an API interface which is cost efficient and easy to implement. Processing this keystroke data, we can provide a probabilistic score which can directly help us recognize if the user is authentic and if there is a need for two-factor authentication.

## 1.3. Introduction to the user model interface

We have developed a basic prototype in the form of a command line interface where a certain user A registers to the system by providing his username/email and password to the system, while the keystroke pattern data is also collected, and can login to the system later. The registration behavioral data of all the users registered in the system is recorded and used for training purposes. The detailed analysis of how the dataset is used for training is described in Chapter 3.

From the training data provided to us, we have developed 2 models for testing: XGBoost and MLP. We use both models to yield scores that depict how much in fraction a user's login keystroke pattern matches his registration keystroke pattern.

It must be noted that we have worked with a less detailed sequence of timing information, namely: we'll only have access to the timing information, without knowing exactly which key was being pressed/released while typing the password. Hence the user's password will always be hidden to the model, just the timing information will be stored.

We have organized the paper in the following manner; Chapter 2 provides detailed information about the user workflow and the user CLI developed. Chapter 3 provides an in-detail depth on the Deep Learning models (XGBoost and MLP) used for training and also gives the graphical analysis of the performance of both these models. Finally, we end this paper by providing a proper conclusion and make room for adaptability and future scopes of this project.

# Chapter 2

# Basic workflow of the model and the CLI developed

This chapter explains in detail about the basic workflow of the model and how the user will interact with it. We have created a sample command-line interface to showcase how the model works and what are the various ways one can integrate the system into their application. The journey of a user is tested from the stages of login/signup in the model to two-factor authentication and authentication based on keystroke dynamics.

## 2.1. Workflow of model

We have designed a flowchart to show the various features and steps integrated in the model in a simplistic way. The flowchart attached below describes the journey of a new user as they pass through various steps and stages of the model. After initializing the model, the user if firstly directed to a login/signup page for password-based authentication. The user is asked to register in the model if visiting for the first time, and user information like **'username'**, **'email-id'**, and **'password'** are collected and stored. While at the same time the Keystroke timing data for the user is also recorded and stored for future use. We have stored the given information and hashed passwords in a CSV text file marked as **'user_info.csv'**, which can be easily edited and managed with the help of python libraries and built-in features like, ***Pandas*** and ***csv***. At the time of registration the user is asked to enter their password 5 times, this helps collect ample amount of keystroke data unique to this new user, and is stored to create an initial keystroke timing dataset, which can be used for authentication in next stages.

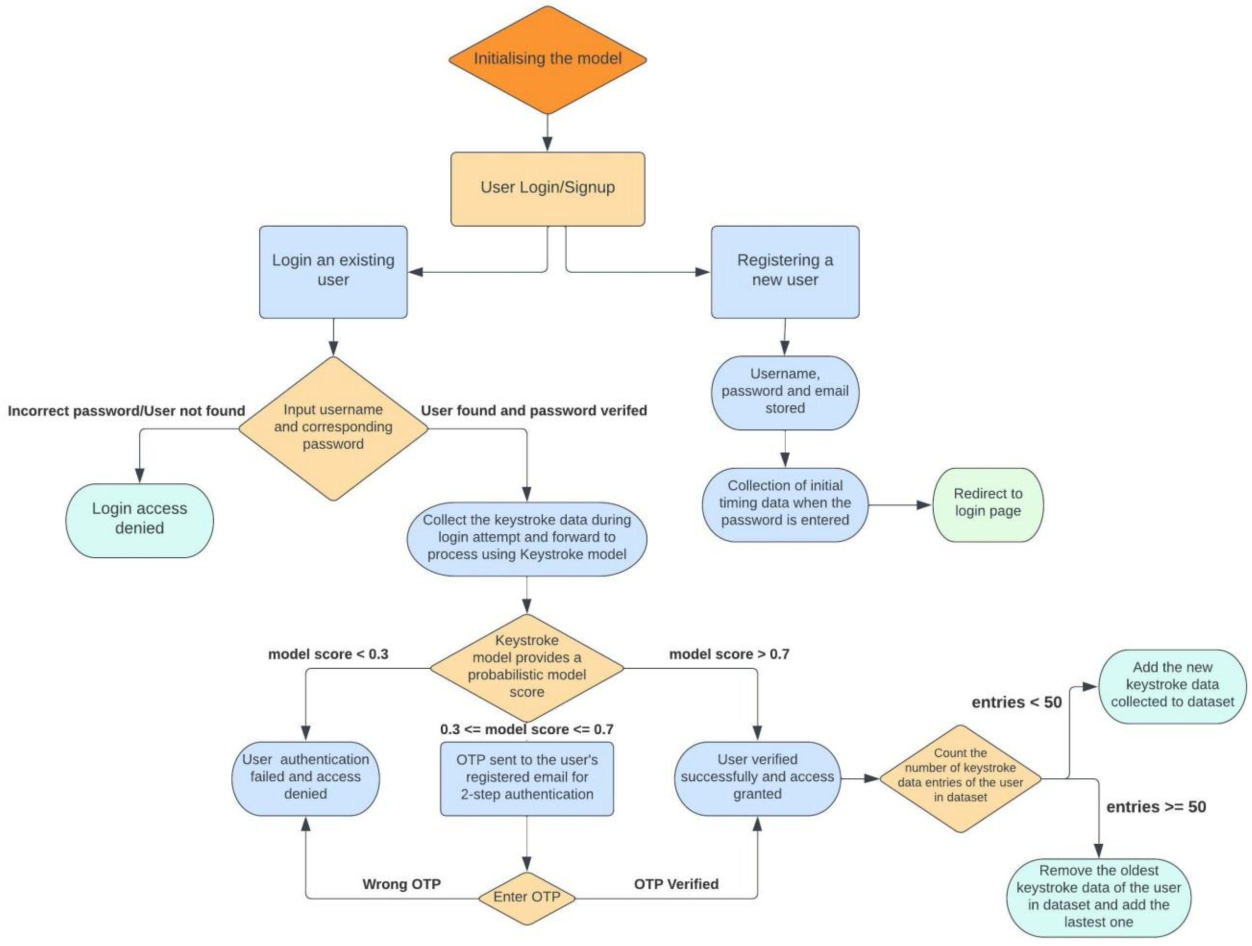

Initialising the model
User Login/Signup
Login an existing user
Registering a new user
Input username and corresponding password
Incorrect password/User not found
User found and password verifed
Login access denied
Username, password and email stored
Collection of initial timing data when the password is entered
Redirect to login page
Collect the keystroke data during login attempt and forward to process using Keystroke model
Keystroke model provides a probabilistic model score
model score < 0.3
model score > 0.7
0.3 <= model score <= 0.7
User authentication failed and access denied
OTP sent to the user's registered email for 2-step authentication
User verified successfully and access granted
Count the number of keystroke data entries of the user in dataset
entries < 50
entries >= 50
Add the new keystroke data collected to dataset
Remove the oldest keystroke data of the user in dataset and add the lastest one
Enter OTP
Wrong OTP
OTP Verified

*Fig 1. Flowchart representing the workflow of the model*

Details on how the timing information on keystroke presses is collected is explained in the next sections. Once the user has successfully registered and created an account, they are redirected to the login window, where the actual work of our Keystroke based authentication model begins. Here, the model asks the user to enter his username and password, and checks for the stored details to verify if the entered details are valid (password-based authentication model). While at the same time, the Keystroke timing data for this login attempt is stored temporarily. If the password is wrong or if the username is not found, there is no reason to move forward and the program terminates. However, in a better scenario, the username is found and the password entered is verified to be correct. After the password is verified as correct, the next stage is to decide if the user can be marked as genuine (authentic) or if further verification is needed to proceed. Our

Keystroke-dynamics based authentication model makes this decision on the basis of a probabilistic score that marks the chances of the user being authentic (login attempt is valid). The temporarily stored keystroke timing data at the time of login attempt is sent to a ***XGBoost/MLP*** model we designed, which takes some keystroke timing information as input, compare it with a pre-defined dataset of timing information of a particular user and predict if both belong to the same person. The model at last returns a probabilistic value marking the probability of user being authentic. The details on how the AI models work and what are the corresponding features is given in Chapter 3.

After sampling the login attempt, the keystroke model returns a probabilistic score or ***'model score'*** as discussed above. Based on the value of this model score, the system decides if the user can be classified as truly genuine/authentic or if the model still has doubts and further verification is needed. A two-factor authentication request is generated in the case of doubt and a OTP is sent to the registered email of the user. Entering the correct OTP, the user can then finally be verified as authentic and access will be granted. At the end, if the user logs in successfully, and is marked as genuine, the temporarily stored Keystroke timing data is then sent to the main timing dataset and is stored, to further improve the efficiency of the AI model.

## 2.2. Two-factor Authentication (OTP-Based)

In today's time, simple password based authentication systems can no longer be trusted, as they are vulnerable to breaches. To counter this problem, many application and organizations concerned with top-notch security have integrated the concept of ***two-factor authentication*** to further verify the user's identity. However, every time the user tries to log into the system, entering a OTP can prove to be an uncomfortable and tiring experience.

However, our keystroke-dynamics based authentication model solves this problem by reducing the amount of redundant two-factor authentication requests. As, the user enters their username and password, and their keystroke timing data is analyzed by the AI model, we receive a probabilistic ***model score*** determining the chances of the user being authentic. Based on the value of this model score a division of model is made. If the value of the model score received is ***more than 70% or 0.7*** then the model is sure that the user is genuine and the login access is granted, there is no need for two-factor authentication in this case any further. On the other hand if the score is ***below 30% or 0.3*** the model surely believes that the user is not the same person as the owner of the account and the access is denied unconditionally. At last, if the model score lies somewhere ***in between 0.3 and 0.7***, the model is still doubtful about the scenario and hence asks for another layer of security. A request for a OTP based two-factor authentication is generated and a randomly generated One-Time Password is sent to the registered email id of the user. On entering, the correct OTP the access is then granted to the system.

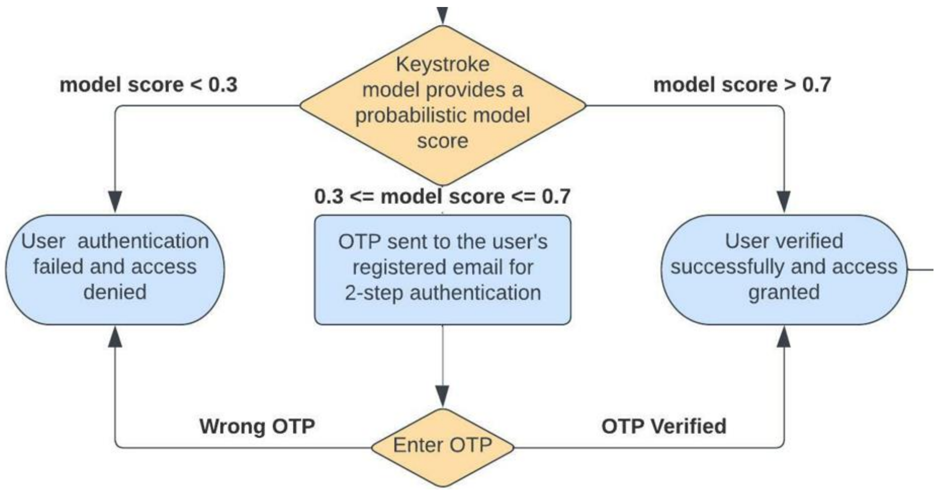

Keystroke model provides a probabilistic model score
model score < 0.3
model score > 0.7
0.3 <= model score <= 0.7
User authentication failed and access denied
OTP sent to the user's registered email for 2-step authentication
User verified successfully and access granted
Wrong OTP
Enter OTP
OTP Verified

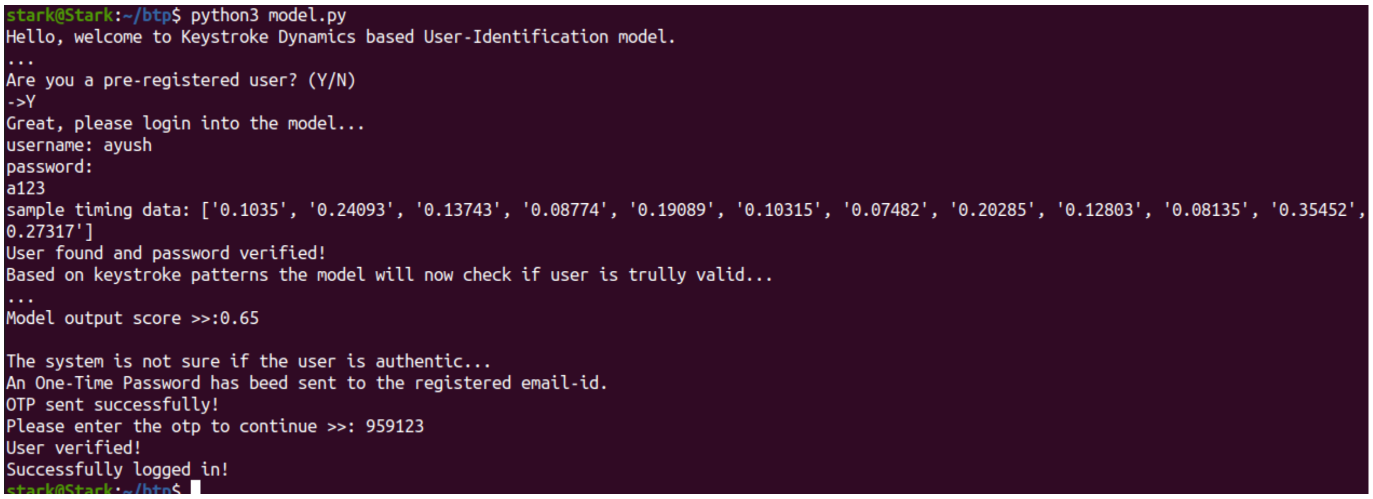
*Fig 2. Decision-making for two-factor authentication*

## 2.3. Command-line Interface for the model

We have developed a Command-line Interface to integrate the above-described model, where the user logs in and registers into the model for a password-based authentication system. While at the same time, keystroke dynamics data is also analyzed and stored. The final decision to check if the user is authentic for every login attempt is then made based on:

- The username and password entered are correct
- The keystroke-dynamics based AI model decides the user as authentic

An instance of the CLI model working in action is shown below. We can see, after the model asks for username and password to enter, a list of keystroke timing data for the password entered is stored. This timing data is then sent to the model for user verification and a model score is returned (***0.65 in this case***). And finally, according to the value of model score, if two-factor authentication is needed (***model score = 0.65 < 0.7***), an OTP is sent to the user and one can log into the system after entering the correct OTP.



```
stark@Stark:~/btp$ python3 model.py
Hello, welcome to Keystroke Dynamics based User-Identification model.
...
Are you a pre-registered user? (Y/N)
->Y
Great, please login into the model...
username: ayush
password:
a123
sample timing data: ['0.1035', '0.24093', '0.13743', '0.08774', '0.19089', '0.10315', '0.07482', '0.20285', '0.12803', '0.08135', '0.35452',
0.27317']
User found and password verified!
Based on keystroke patterns the model will now check if user is trully valid...
...
Model output score >>:0.65

The system is not sure if the user is authentic...
An One-Time Password has beed sent to the registered email-id.
OTP sent successfully!
Please enter the otp to continue >>: 959123
User verified!
Successfully logged in!
stark@Stark:~/btp$
```

*Fig 3. An instance of the CLI model working in action*

## 2.4. Dynamic data entries

The behavioral typing patterns of a user are expected to change with time. Thus, to keep a record for these changing habits, the dataset containing the Keystroke timing data for users must be continuously updated. We need to ensure that the data never gets stale and the model works efficiently with the user's current typing pattern. Thus, to implement the same, on every successful login attempt the user makes, the keystroke timing data for this session is stored into the main dataset. Our Keystroke-dynamics based authentication model needs to have around 50 keystroke timing data entries for every user, to make the decision efficiently. Hence, every time we have a new timing data to be stored, one of the two decisions is made,

- If the total number of keystroke timing data entries of this particular user in the main dataset is less than 50, we simply add the new timing data to the main dataset.
- If the total number of keystroke timing data entries for this user is greater than or equal to 50, we remove the oldest timing data of this user from the main dataset and add a new entry with latest data.

# Chapter 3

# Deep Learning and Machine Learning model analysis

## 3.1. Background

Chapter 2 describes in detail the user workflow and the process by which a particular user first registers himself using his keystroke registration pattern and how this pattern for all the registered users is used for training purposes. Based on the training, a model score is passed as output to the user interface which determines whether the user needs to undergo OTP verification or whether the user is considered genuine. This chapter will deal with the AI part of the user interface which trains the model using the user registration dataset and determines a score of a particular login keystroke pattern determining if the particular user is false or genuine. We developed 2 models for training, both of which can be used to determine the score of the login user. We describe in detail the dataset used for training followed by the training algorithm.

## 3.2. Methods and Approaches

### *A. Extreme Gradient Boosting (XGBoost)*

XGBoost is a boosting tree ensemble method that uses weak learners trees. XGBoost is an extension of gradient boosting trees. The algorithmic implementation of XGBoost has an addition of regularization function to the loss function that enables the algorithm to avoid overfitting. In XGBoost, besides the regularized objective, shrinkage and column (feature) subsampling are two additional techniques that are used to reduce overfitting.  XGBoost also has out-of-core, cache aware along with code parallelization features that makes XGBoost very fast and very efficient.

The XGBoost library offers three different types of hyperparameters for tuning: General parameters, parameters for the chosen booster, and learning parameters. Booster parameters concern the tree structure itself. There are several parameters and the following are the most common to tune:

**Max depth**: Max depth limits on the depth of the tree.

**Min child weight**: Min child weight sets a limit on the weight of a tree node for partition to occur.
**Alpha**: Alpha represents the L1 regularization term.

**Subsample**: Subsample sets the share of data selected for training.

**Learning rate**: learning rate adjusts the weights after each boosting step.

### *B. Multi-Layer Perceptron MLP Model*

The MLP model consists of four fully connected layers, in which the number of neurons is **512, 256, 144,** and **51**, respectively. The last layer's output is fed into the softmax function to determine the corresponding probability of each user. A rectified linear unit (relu) activation function and a batch normalization layer are used in the first and second dense layers. We use the cross-entropy loss function for this model to calculate the error.

## 3.3. Dataset and feature selection

To train the keystroke model, we first collect the Keystroke dynamics data through regular QWERTY keyboards, which have receivers that register time-stamps of the keystrokes. The detailed description of how this receiver is implemented is provided in chapter 2. Here, we directly use a dataset that already has 1000s of keystroke data and is used in past research for training. The use of a common dataset enables research to be directly compared. The following research uses the CMU keystroke dataset for training our model (https://www.cs.cmu.edu/~keystroke/#sec2). The dataset is a supplement to the paper "**Comparing Anomaly-Detection Algorithms for Keystroke Dynamics**," by Kevin Killourhy and Roy Maxion, published in the proceedings of the DSN 2009 conference. There are now many studies that use this same dataset and outperform a baseline result that was achieved in the above paper. The data consists of keystroke-timing information from 51 subjects (typists), each typing a password (**.tie5Roanl**) 400 times that consists of 50 repetitions over 8 sessions, thus giving a total of 20400 entries. This password can be seen as a generic password of 10 characters as it contains a special symbol. The ENTER key is pressed after typing the password thereby making a total of 11 keystrokes. Each keystroke pair gives rise to 3 features that includes:

The duration for which a key is pressed (**hold time**),

The duration of time from one key press to another (**down-down time**) and

The duration of time between the release of 1 key and the press of another key (**seek time**).

Hence a total of 31 features are obtained which are used in designing the Neural network models.

## 3.4. The XGBoost model

### 3.4.1. Cross Validation

The first step after preprocessing and selecting the features from the dataset is splitting the training dataset into a training and a validation dataset. Cross-validation splits the training sets into folds, where each set is used for validation and the remaining are used for training as shown in the below figure. This helps in avoiding data leakage and under-fitting. The model evaluates which split scored the highest accuracy and uses that model in the testing phase on the original test data set.

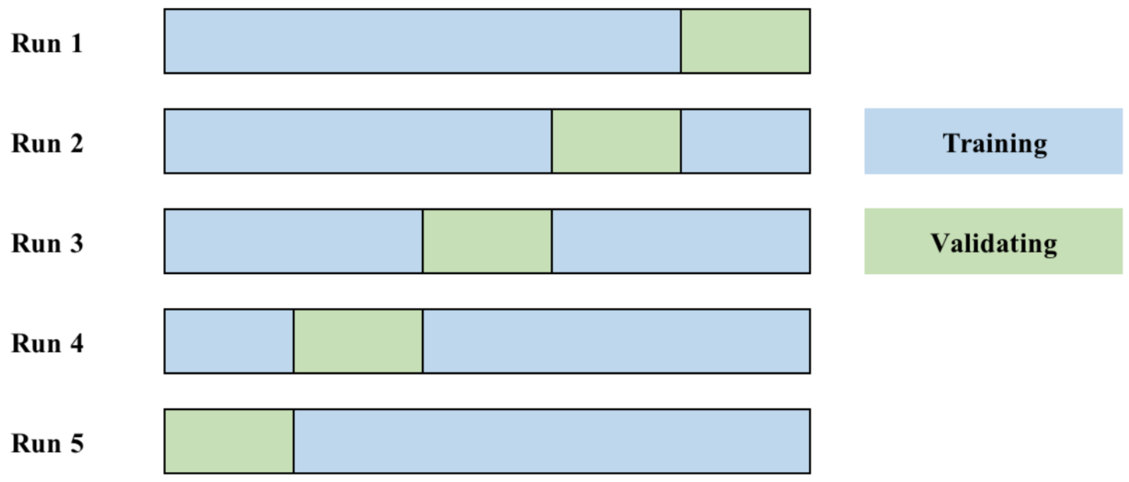


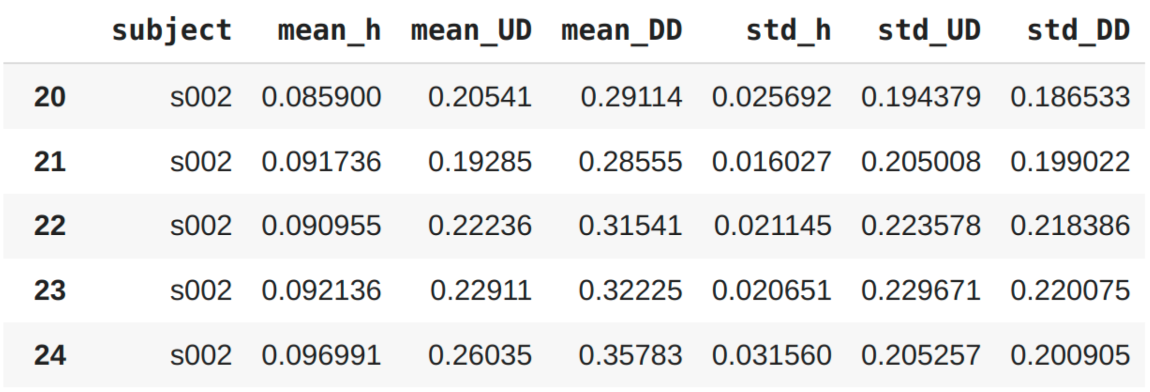
*Fig 4. Illustration depicting cross validation*

The next step is to split the usernames of the dataset into usernames whose data will only be used for training and validation, and usernames whose data will only be used for testing. Here, out of 50 usernames, 40 are used for training and validation while the remaining 10 are used for testing. For each username there are 400 entries in the dataset. Capturing 400 entries for a particular user is extremely tiresome and rarely achievable if adapted on real world systems.

Hence, we select the first **20** keystroke entries of a certain user as the **registration** dataset. For training the model we only require the use of registration dataset for each user as we will describe below.

### 3.4.2. Feature extraction

It must be noted that we have extended our model to take in input keystroke data with variable password lengths. There is no condition required for the password length to be the same for every user. After acquiring the dataset, features are extracted. As the passwords are all different from each other, direct comparison of time-stamp data key to key is impossible. The features selected instead are the mean and the standard deviation of different types of flight times along with their usernames as shown in the figure below.



| | subject | mean_h | mean_UD | mean_DD | std_h | std_UD | std_DD |
|---|---|---|---|---|---|---|---|
| **20** | s002 | 0.085900 | 0.20541 | 0.29114 | 0.025692 | 0.194379 | 0.186533 |
| **21** | s002 | 0.091736 | 0.19285 | 0.28555 | 0.016027 | 0.205008 | 0.199022 |
| **22** | s002 | 0.090955 | 0.22236 | 0.31541 | 0.021145 | 0.223578 | 0.218386 |
| **23** | s002 | 0.092136 | 0.22911 | 0.32225 | 0.020651 | 0.229671 | 0.220075 |
| **24** | s002 | 0.096991 | 0.26035 | 0.35783 | 0.031560 | 0.205257 | 0.200905 |

*Table 1. Sample of users with timing sequence features*

Here,

**subject**: Name of the user (here entered as a number to maintain anonymity)

**mean_h**: Average of the hold time values corresponding to each letter of the password.

**mean_UD**: Average of the up-down time values corresponding to each consecutive pair of letters in the password including the return key.

**mean_DD**: Average of the down-down time values corresponding to each consecutive pair of letters in the password including the return key.

**std_h**: Standard deviation of the hold time values corresponding to each letter of the password.

**std_UD**: Standard deviation of the up-down time values corresponding to each consecutive pair of letters in the password including the return key.

**std_DD**: Standard deviation of the down-down time values corresponding to each consecutive pair of letters in the password including the return key.

The next step is to split the training set into folds for cross-validation. Within the folds, the binary classification algorithm is executed: All feature sequences are paired with each other and labeled according to whether they belong to the same user or different user, and the feature difference table is extracted that is simply the difference between the sequences per feature. This is shown in the below figures.

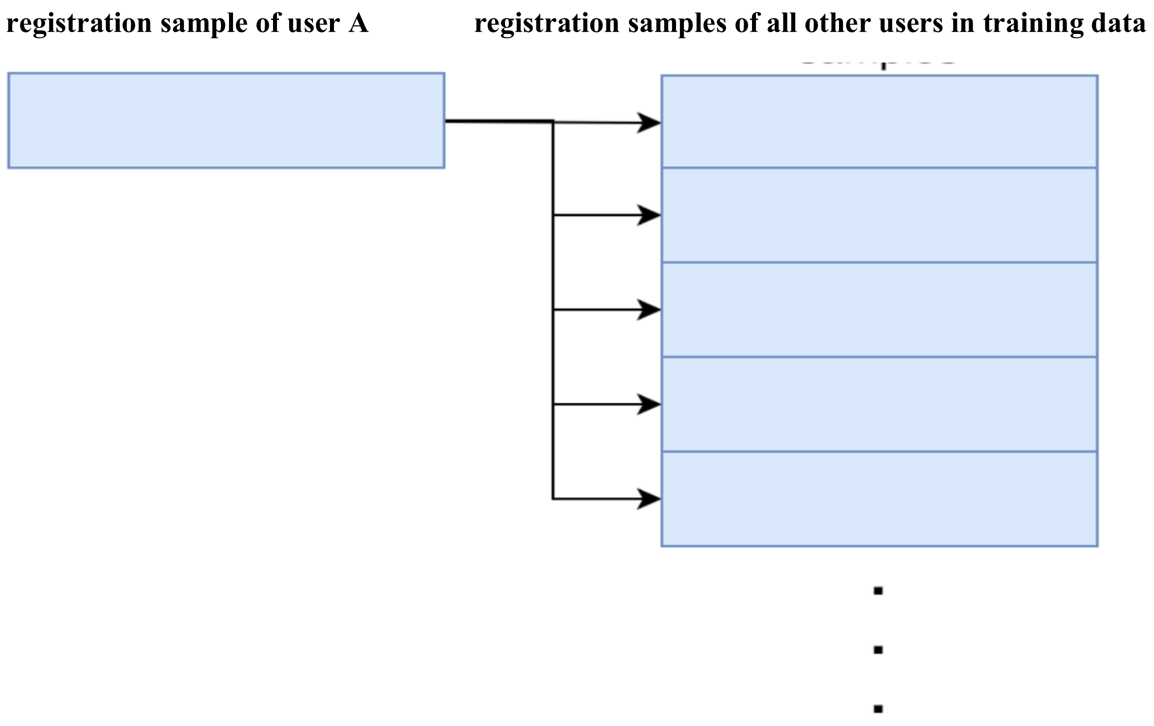
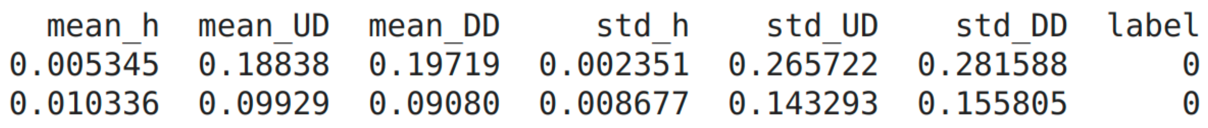

registration sample of user A
registration samples of all other users in training data

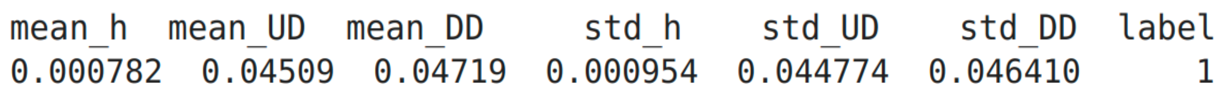
*Fig 5. Extracting features by pairing sequences*

This pairing of all the keystroke features of a certain user with the keystroke features of all other users in the registration dataset creates new feature matrix that we can point as the '**difference feature matrix**'. A label is also inserted depicting whether a particular row of the difference feature table belongs to the same user or a different user. The difference feature calculated from both two feature sequences from the same user and from two different users can be seen in the below figures.



```
  mean_h   mean_UD  mean_DD     std_h    std_UD    std_DD  label
0.005345  0.18838  0.19719  0.002351  0.265722  0.281588      0
0.010336  0.09929  0.09080  0.008677  0.143293  0.155805      0
```

*Table 2. The above table shows 2 examples where the user data is paired with a different user and the label used is 0.*



```
 mean_h   mean_UD  mean_DD     std_h     std_UD     std_DD  label
0.000782  0.04509  0.04719  0.000954  0.044774  0.046410      1
```

*Table 3. The above table shows 2 examples where the user data is paired with the same user and hence the label used is 1.*

This same kind of pairing is done for data in the validation set. The model is now trained on the difference feature matrix using 2 Deep Learning algorithms: MLP and XGBoost.

### 3.4.3. Algorithm for training the XGBoost model

1. After obtaining the registration dataset, split all the usernames into training and testing usernames in the ratio of 4:1.
2. Split the registration data for the training set usernames into 4 folds of training and validation sets to execute cross validation.
3. Create an XGBoost Regressor with seed value set as 20.
4. Loop across all folds:
    a. Calculate the difference between all possible pairs of training and validation keystroke sequences.
    b. Fit the XGBoost regressor using Randomized Search to the difference features and labels.
5. Make prediction of the validation feature differences matrix and do a comparison to their corresponding labels.
6. Calculate AUC score, RUC, TP, FP, FN, TN and accuracy values for each split.
7. Figure out the best performing fold model based on the above output values of accuracy and AUC scores.

XGBoost offers a wide range of hyperparameters as described above. We use the Randomized Search optimization algorithm for tuning the hyperparameters. The Random search algorithm uses a large range of hyperparameter values and iterates a specified number of times over combination of these values randomly. The range of hyper parameters used for tuning the XGBoost model are described in the below table. We use scoring parameter as AUC values and the number of iterations as **25**.

| **Step #** | **Parameter** | **Range (start, stop, step)** |
|---|---|---|
| 1 | max_depth | (0, 11, 1) |
|  | min_child_weight | (1, 11, 1) |
| 2 | gamma | (0, 1, 0.1) |
| 3 | subsample | (0.6, 1.1, 0.1) |
|  | colsample_bytree | (0.6, 1.1, 0.1) |
| 4 | reg_alpha | (1e-5, 100, *10) |
|  | reg_lambda | (1e-5, 100, *10) |
| 5 | learning_rate | (0, 1, 0.02) |

*Table 4. Range of hyperparameters used for training*

### 3.4.4. Algorithm for testing the XGBoost model

For testing, we use a similar approach that we used for training the XGboost model. For each user in the test dataset, we divide his keystroke data sequence into registration and login dataset. While training the model, we don't require the login dataset since we pair the registration dataset values of a particular user with every other user so we get both positive as well as negative labels.

In order to test the model, we divide the testing dataset for each user into the registration and the login dataset. The registration dataset contains any 10 sequences of the user but for simplicity we use the first 10 keystroke sequences for each user as the registration dataset. The login dataset consists of 150 entries for each user that is not in the registration dataset.

In the next step, we pair each user's entry from the login dataset to all the entries from the registration dataset of the same user. In this way, we calculate the mean prediction of whether a given sample belongs to the same user or not.

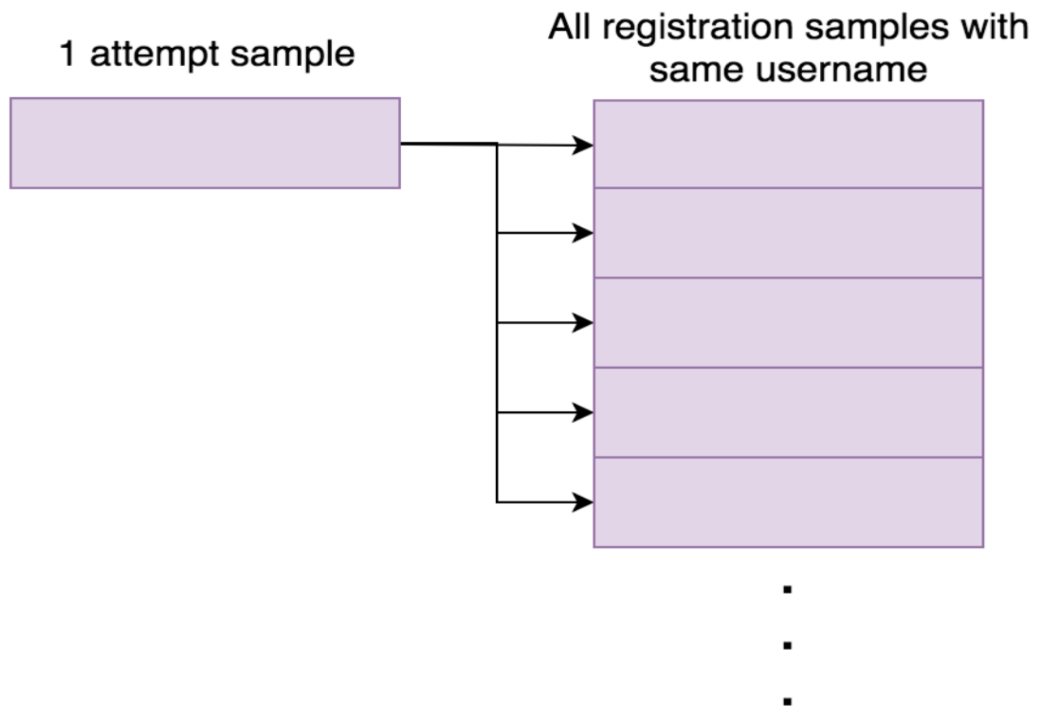


*Fig 6. Illustration showing pairing of timing sequences in testing data*

The above figure shows that each attempt sample of a particular user is paired with all the keystroke sequences of the claimed username and the mean prediction is calculated depicting whether the user is genuine or fake.

It must be noted that in the actual user interface, we output a score that is the mean of all predictions of the attempt sample paired with the registration dataset. Based on this score, the user interface decides whether to login the user or continue to the 2 factor OTP based authentication.

### 3.4.5. Hyperparameter tuning

The most important hyper parameters that require tuning are *max-depth, min_child_weight, gamma, subsample, colsample_by_tree, reg_alpha, reg_lambda* and *learning rate*. We use a Random Search algorithm for tuning the hyper parameters.

In random search we define a range of hyperparameters which is defined uniformly or with a certain sampling procedure. Here, not all values are tested and values are selected at random. For example, if there are 1000 values in the uniform distribution and thr number of iterations is set to

100, the algorithm will randomly sample 100 values to test. As Random search doesn't try every single permutation of the hyperparameters, there is no guarantee it will return the best performing values but it can be said that it returns a relatively good performing model in a very short time interval.

The parameters used in random search are *param_distributions* for defining the search distributions across which the algorithm will perform. Then, we have *n_iter* that is used to limit the total number of model runs. It must be noted that while experimenting, high values of *n_iter* increase search runtime while low values of the same parameter decrease the quality of the model. After performing the Random Search, we get the following results of hyperparameters in XGBoost model.

| | | | |
|---|---|---|---|
| **max_depth** | 5 | **n_estimators** | 500 |
| **learning_rate** | 0.04999 | **gamma** | 0 |
| **subsample** | 0.7 | **scoring** | neg_mean_squared_error |
| **colsample_bytree** | 0.7 | **N_iter** | 25 |
| **reg_alpha** | 1 | **Seed value** | 20 |
| **reg_lambda** | 1e-5 | **verbose** | 1 |

*Table 5. XGBoost hyperparameters obtained after tuning*

Using the above hyperparameters we obtain the least RMSE (root-mean-square error) value as **0.27732.**

## 3.5. The MLP model

### 3.5.1. Algorithm for training the MLP model

The MLP or Multi-Layer Perceptron model is a generic feed forward neural network model that comprises an input layer, an output layer and 2 or more hidden layers. It is trained using the back propagation algorithm. Our model uses 5 layers – 1 input layer taking the input as the **31-dimension** feature vector and 3 hidden layers with the number of neurons as **512, 256 and 144.** The output layer having **51** neurons is fed into the **softmax** function which calculates the final probability of each of the respective classes. Only 1 model is trained which learns to classify all the users based on their keystroke pattern.

To code our model, we used the **keras** python library. For experiments, we took the 200 initial timing feature vectors and reserved 10% of training data as validation data for hyper-parameter

tuning. We used the **categorical cross entropy** as our loss function as we are dealing with a multiclass classification problem.

### 3.5.2. Techniques used to optimize the MLP model

*A. DROPOUT AND LEAKYRELU/RELU*

Dropout is used after the hidden layer in both MLP and CNN experiments to prevent overfitting of the model. We use a dropout value of 0.5 which works best in our cases. We use Leaky RELU or RELU as our activation function in the MLP and CNN model respectively. The Leaky RELU activation function allows for a threshold value to pass and does not entirely eliminate the neuron.

*B. BATCH NORMALIZATION*

We use Batch Normalization after every fully connected layer to normalize the input before passing it into the activation function. This helps in stabilizing the learning.

*C. PREPROCESSING THE DATA*

We preprocess the data by normalizing it to mean 0 and standard deviation 1 before feeding it into the neural network.

### 3.5.3. Hyperparameter tuning

After applying Random Search for hyperparameter tuning, the following hyperparameters were obtained.

| | |
|---|---|
| **Number of epochs:** | 200 |
| **Weight Initialization:** | Xavier |
| **Dropout values**: | 0.3, 0.5, 0.3 |
| **Batch size** | 32 |
| **Learning Rate**: | 0.001 |
| **Optimization** : | Adam |

*Table 6. MLP Hyperparameters obtained after tuning*

# Chapter 4

# Results and analysis

## 4.1. XGBoost model results

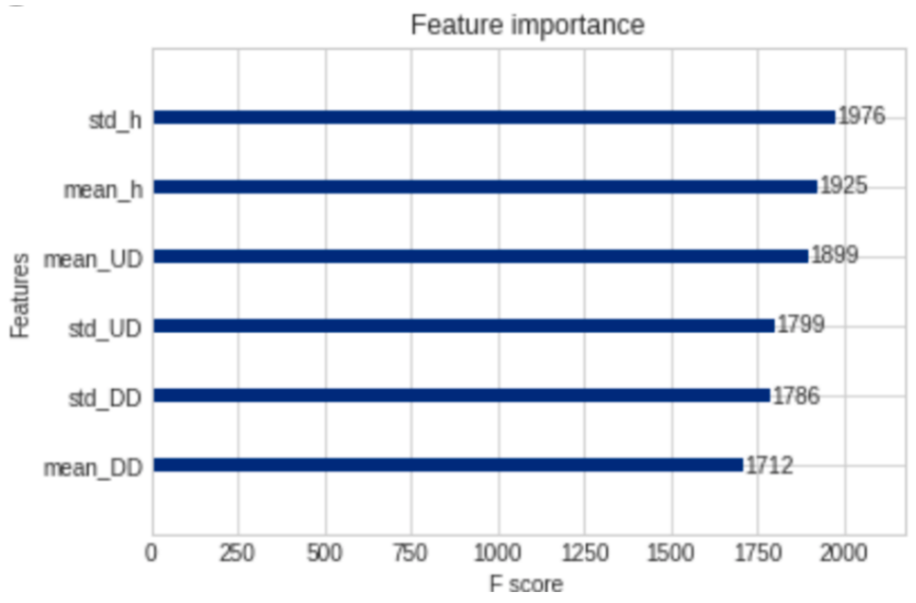


*Fig 7. Feature importance graph after tuning the hyperparameters*

The above graph shows the plot of features vs the F score that denotes the importance and relevance of each of the 6 features used in the XGBoost Model.

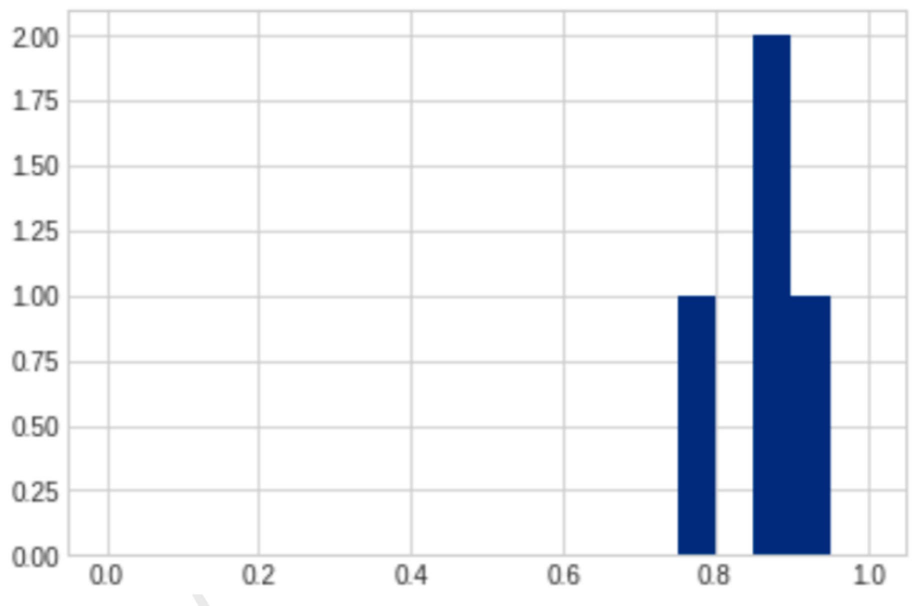


*Fig 8. Histogram plot of AUC scores*

The above histogram plot depicts the AUC scores vs the number of folds that achieved that particular AUC score. It can be seen from the above figure that the AUC scores reach up to 0.9 and the two folds perform equally well in terms of AUC scores.

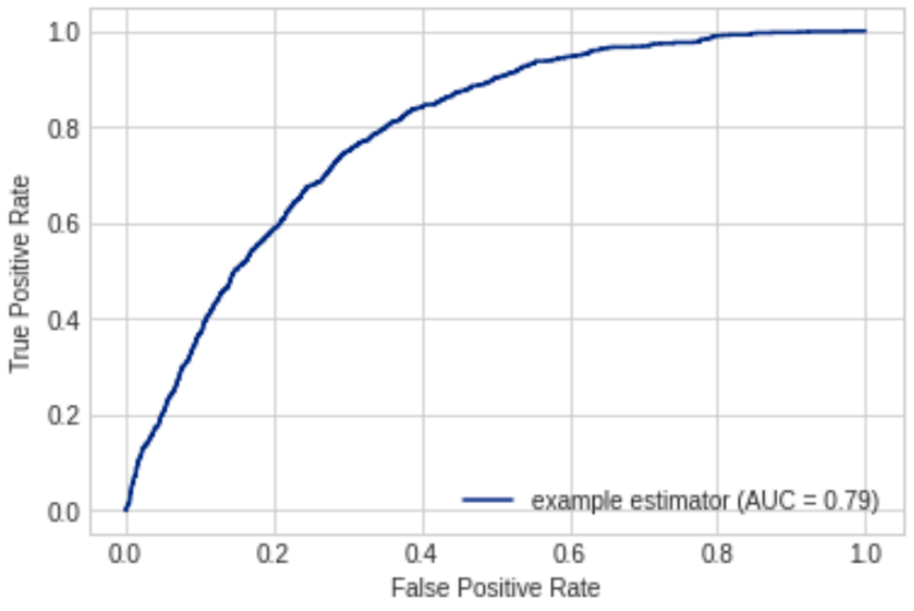


*Fig 9. ROC curve of Split 1 after tuning*

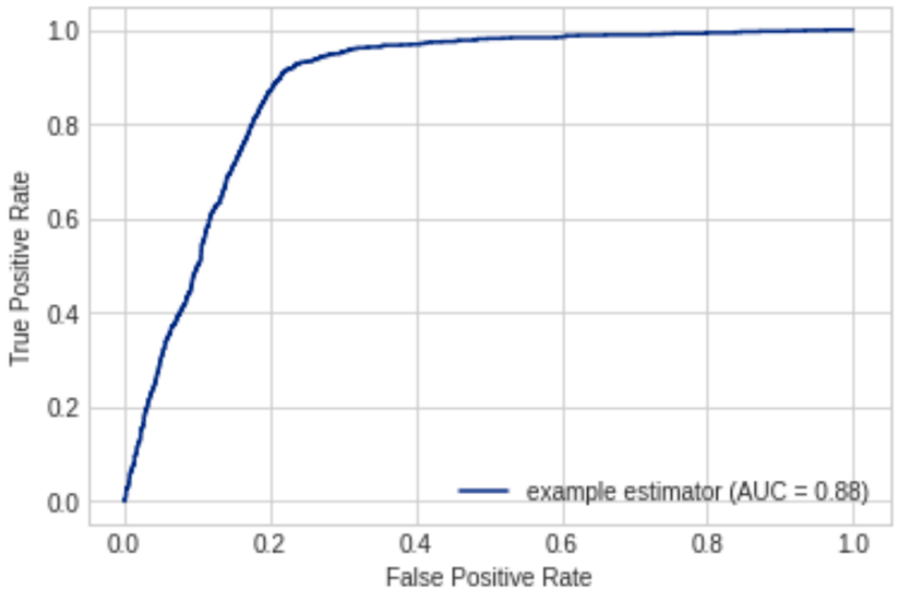


*Fig 10. ROC curve of Split 2 after tuning*

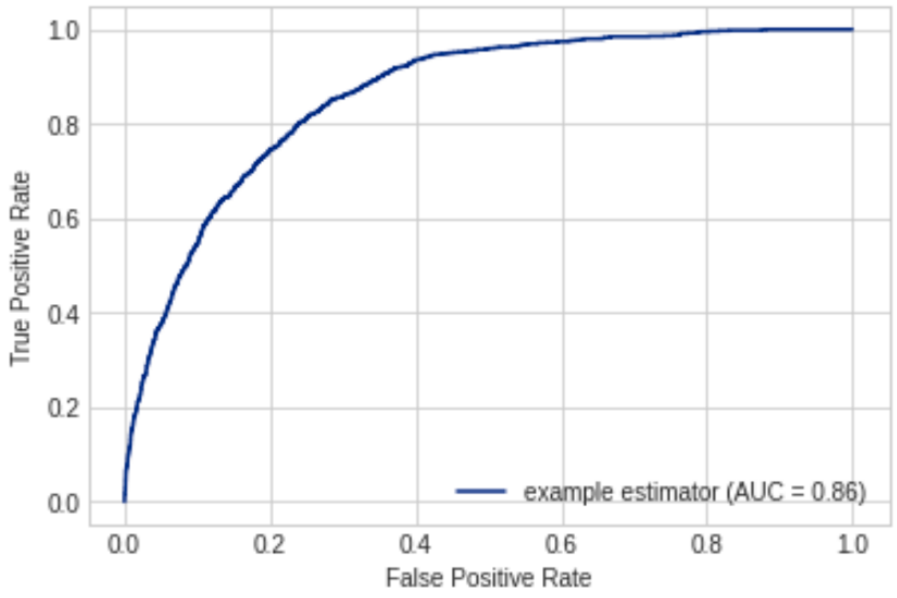


*Fig 11. ROC curve of Split 3 after tuning*

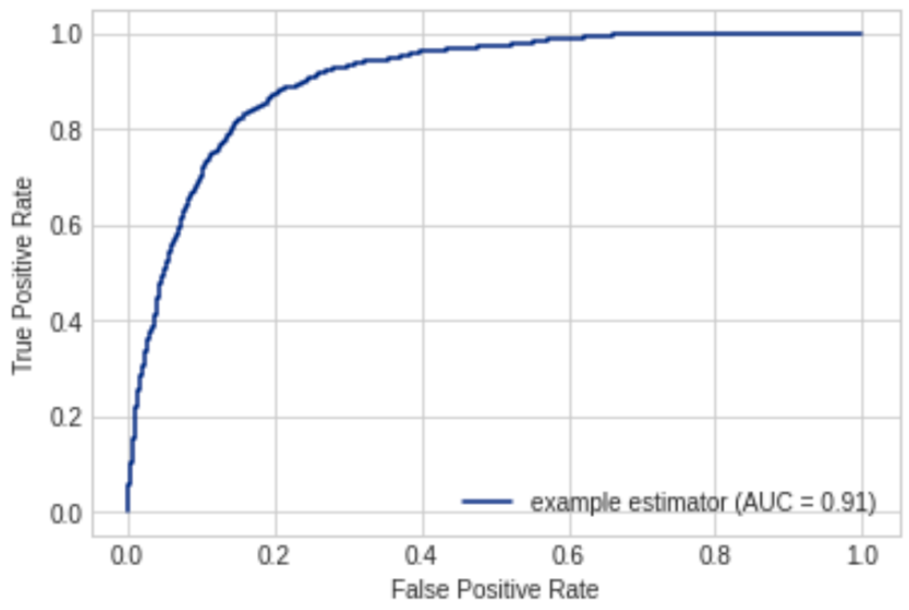


*Fig 12. ROC curve of Split 4 after tuning*

From the ROC curves, it can be observed that split 4 performs best since it has the highest score and split 1 performs worst as it has the lowest score among all folds. We can verify that higher the area under the ROC curve, higher will be the model's performance. This is because greater area implies greater values of True positives at smaller values of False positives. In split 4, we can see that the True positive rates are much higher than the false positive rates at any threshold value.

| | **SPLIT 1** | **SPLIT 2** | **SPLIT 3** | **SPLIT 4** |
|---|---|---|---|---|
| **AUC Scores** | 0.79403466 | 0.88056959 | 0.87300950 | 0.91461537 |
| **False Positive rate** | 0.00512821 | 0.0032967 | 0 | 0 |
| **False Negative rate** | 0.19230769 | 0.19084249 | 0.18553114 | 0.18772894 |

*Table 7. Statistics of XGBoost model results*

The above table shows the AUC scores, False Positive and False Negative rates for each of the 4 splits. We can observe that Split 4 performs best in terms of AUC scores and False positive rates. One important observation to note is that the False positive rate score is very less in XGBoost model which shows that the model will most likely not predict an imposter as genuine user even if the imposter gets to know the password of the user. This property is the main crux of our model since predicting true users as imposters is still acceptable since the user can move on to OTP based verification but predicting false users as true is detrimental.

## 4.2. MLP model results

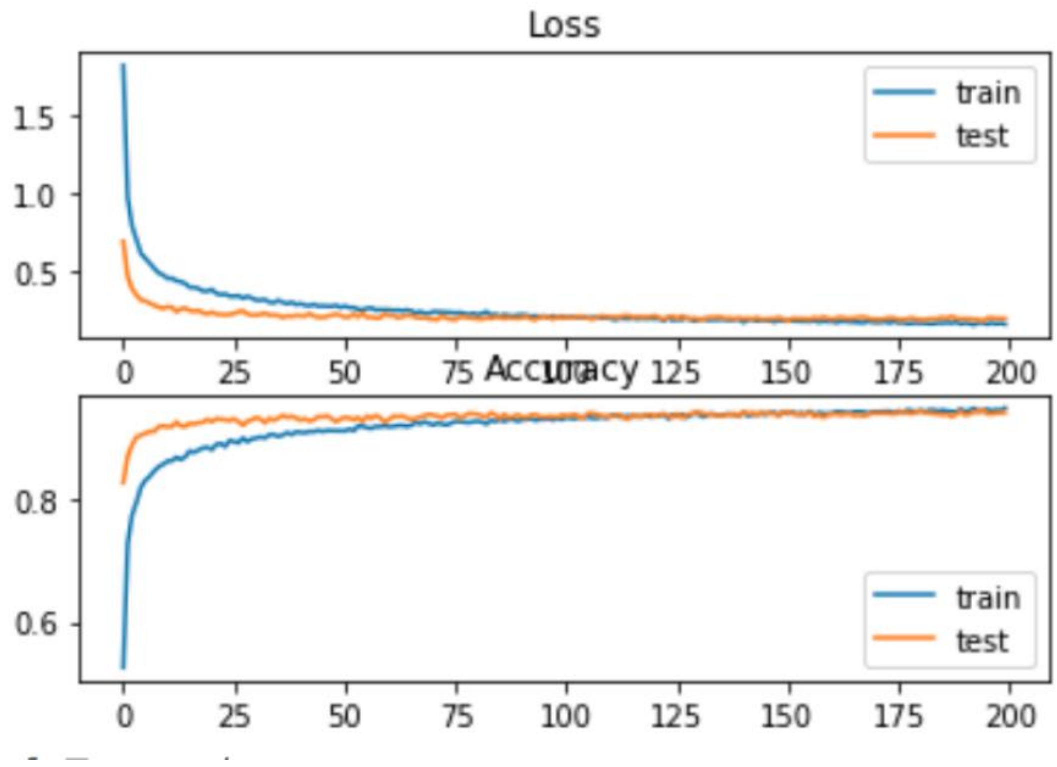


*Fig 13. Plot of Loss and Accuracy vs the number of epochs*

From the above 3 graphs, it is apparent that the validation loss function decreases with the number of epochs in the MLP neural network model. The loss function converges at around 100 epochs after which it stays constant.

| | Accuracy | False +ve (FP) | False -ve (FN) | Precision | Recall |
|---|---|---|---|---|---|
| **MLP** | 94.09% | 210 | 275 | 0.947 | 0.932 |

**AUC score**: 0.8905
**False positive rate**: 0.05

*Table 8. Statistics of MLP model results*

From the above statistics, we can conclude that the MLP model performs very well in terms of Accuracy and False positive and Negative scores. Similar to that of XGBoost, the False positive rate score is very less in MLP model which shows that the model will most likely not predict an imposter as genuine user even if the imposter gets to know the password of the user.

# Chapter 5
# Conclusion and Further work

## 5.1. Summary and Conclusion

The purpose of doing research on Keystroke Dynamics was to check if there was a plausible future that exists in security containing Keystroke dynamics-based authentication. We designed 2 models based on XGBoost and MLP and got results that matched and exceeded our expectations. We analysed the models based on ROC curves, AUC values and False positive and negative rates. For both models, the False positive scores are close to 0 which is a good achievement of our model. This is because for an authentication model we don't want to falsely predict an imposter as genuine user, even if we do predict a genuine user as imposter. Hence, this reduces the probability of hackers logging into someone else's system.

Regarding the individual models, it was very crucial decision to tune the hyperparameters which resulted in getting the best results. The random search algorithm worked perfectly well in this case as it provided optimal results in minimal time period.

As of now, our research doesn't deal with which of the 2 models it uses for predicting the model score but there are some ideas that we can use. We can either calculate an average of individual scores of each model and calculate an average score to use or, we can randomly select any model and predict whether the user is genuine or fake. To conclude, we have successfully designed a full-fledged working user interface of the Keystroke Dynamics model.

## 5.2. Future Work

Many research has been done on the topic of Keystroke dynamics but only some of them has been put to practise. The future work in this field will be to incorporate our model into various corporate specific login systems like banks and ATMs. We need a proper mechanism to integrate the login system and the AI model in the cloud. Also, further parameters can be used to check the model accuracy like taking 3 or more keys at a time or using a sequence model that is used in sentence analysis and emotion prediction like rnn, lstm and auto encoders.

Research can also be expanded to other user behavioural aspects including but not limited to touchscreen keypad timing analysis. Research can also be conducted to check the sound with which the keypad is tapped to analyse user behaviour.

# References

1. Saket Maheshwary, Soumyajit Ganguly, and Vikram Pudi. Deep Secure: A fast and simple neural network based approach for user authentication and identification via keystroke dynamics.

2. Hayreddin Çeker and Shambhu Upadhyaya. Sensitivity analysis in keystroke dynamics using convolutional neural networks. In 2017 IEEE Workshop on Information Forensics and Security, WIFS, pages 1–6, 2017.

3. Ahmet Gedikli and Mehmet Efe. A simple authentication method with a multilayer feedforward neural network using keystroke dynamics. https://web.cs

4. Fusion of Deep Learning Models for Improving Classification Accuracy of Remote Sensing Images P. Deepan 1 , L.R. Sudha 2

5. Convolutional Neural Networks for Sentence Classification Yoon Kim New York University yhk255@nyu.edu

6. Keystroke Analysis for User Identification using Deep Neural Networks Mario Luca Bernardi, Marta Cimitile , Fabio Martinelli, Francesco Mercaldo, Giustino Fortunato University, Benevento, Italy 2015-2015 IEEE, no. 2015-02, 2015, pp. 1305–1310.

7. W. Chen and W. Chang, "Applying hidden Markov models to keystroke pattern analysis for password verification," in Information Reuse and Integration, 2004. IRI 2004. Proceedings of the 2004 IEEE International Conference on. IEEE, 2004, pp. 467–474.

8. P. Kang and S. Cho, "Keystroke dynamics-based user authentication using long and free text strings from various input devices," Information Sciences, vol. 308, pp. 72–93, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.ins.2014.08.070

9. D. W. Salil Partha Banerjee, "Biometric authentication and identification using keystroke dynamics: A survey," Journal of Pattern Recognition Research., vol. 7, no. 1, 2012