Biometric Liveness Detector

Hemendu Roy
Arizona State University
Tempe, USA
hroy6@asu.edu

Nikhil Anantharaman
Arizona State University
Tempe, USA
nananth6@asu.edu

Ravindra Aditya Singh
Arizona State University
Tempe, USA
rsing122@asu.edu

Thejas Naik
Arizona State University
Tempe, USA
tanaik@asu.edu

***Abstract*—Like any software-based framework, Biometric systems are susceptible to a variety cyber-attacks. Due to the nature of such systems and their wide use in the field of Medical Science, the consequences of a malfunction could be extreme. In this project, we design a system using an Android Application and a backend that utilizes Machine Learning to detect the liveness of a biometric signal. To achieve the best possible results, an array of Machine learning models such as Random Forest, Support Vector Machine (SVM), Hierarchial Clustering, Logistic Regression, MultiLayer Perceptron Classifier and Adaboost classifiers were used. The models were then compared using accuracy, False Accept Rate, False Rejection Rate, F1 scores and Half total error rate. Finally, a computation of the time to train and time to test of each model was performed. These results are later, consolidated and analysed.**

***Index Terms*—Machine Learning, Android Development**

I. Introduction

The analysis of Brain Signals is critical to the field of Modern Neuroscience as computers and other digital systems find their way into traditional Medicine. Naturally, there is a need for safeguarding such systems against attackers. An example of an attack is when a fake and/or malicious signal is sent to the system in order to trigger unexpected behaviour. To mitigate this, the proposed system uses various Machine Learning methods to detect the liveness of a signal. The outcome of each machine learning model is then fed into a voting model which extracts the optimal outcome and presents it to the user. The interface for the system is an Android mobile application.

II. System Architecture

A. Infrastructure Setup

The framework consists of an Android Application Front-end running on a mobile device and a Machine Learning-based backend running on a server. Their specifications are as described below:
Android Device:

1) Model : Pixel 3a
2) Android API Version : 29

Verification Server:

1) CPU : Intel i9 9900K
2) RAM : 32 GB DDR4
3) Operating System : Windows 10

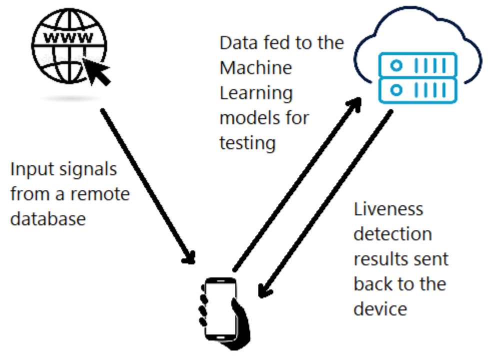

WWW
Data fed to the Machine Learning models for testing
Input signals from a remote database
Liveness detection results sent back to the device

Fig. 1. Project pipeline concept

B. Project Pipeline

The transactions between the mobile device and the verification server are illustrated in Fig 1.

C. Verification Server

1) The server is implemented using a lightweight Web Server Gateway Interface WSGI web application framework called Flask.
2) Pretrainied Machine learning models such as Support Vector Machine, Random Forest Classifier, Multi-Layer Perceptron Classifier, Hierarchial Clustering, Logistic Regression and Adaboost have been deployed on the server.
3) The server classifies an input signal as live or fake using each of the pretrained models and aggregates the results.
4) The aggregated results are then fed into a voting model that chooses the best result and sends it back to the mobile device.

III. System Implementation

2) Android Application The app has been designed to be able to download and upload data from/to the flask server running on the desktop computer. To receive biometric signal data, the user must press the 'Receive Data' button. The app then downloads the required data from the flask server filesystem. This step simulates the process of receiving data from the Internet. Next, to receive a signal detection result, the user must press the

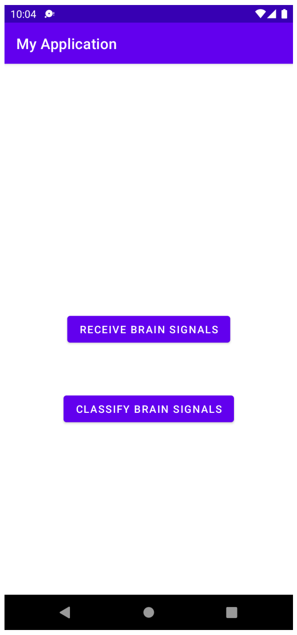
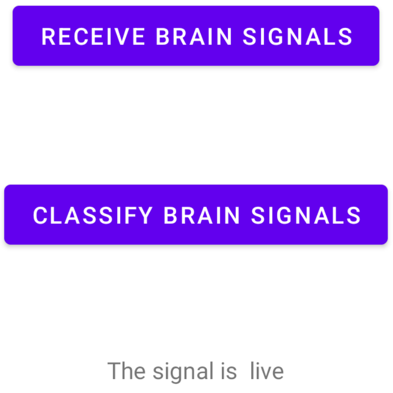


Fig. 2. Android Application User Interface

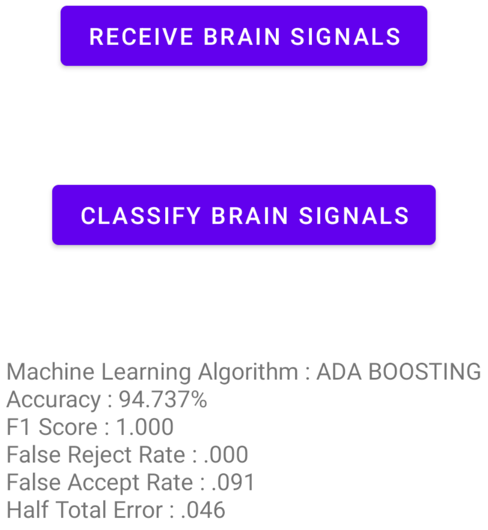
'Upload Data' button to send the received data to the machine learning backend for testing and classification. On the server side, the models are used to classify the brain signal. The results are then, consolidated and finally, the best one is segregated using a voting model.



The signal is live

Fig. 3. Received results on the mobile device for a single input

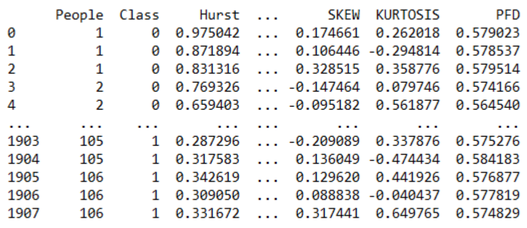
3) Verification Server using Machine Learning
   - Step 1 : An HTTP GET request is sent to the server to receive the brain signal data.
   - Step 2 : An HTTP POST request is sent to the server to upload brain signal data for liveness detection.
   - Step 3 : Python applications designed to utilize pretrained models are used to process the signals and classify them as live or fake. Additionally, performance parameters are also computed and sent back to the mobile device.
   - Step 4 : The mobile application sends an HTTP GET request to the server. The server serves this request and sends a classification result along with a set of performance metrics to the device.



Machine Learning Algorithm : ADA BOOSTING
Accuracy : 94.737%
F1 Score : 1.000
False Reject Rate : .000
False Accept Rate : .091
Half Total Error : .046

Fig. 4. Received results on the mobile device for multiple inputs

4) Training Data From the provided dataset, brain signal data has been extracted in the form of a JSON file. The original dataset is a matrix of 106 subjects x 3 trails x 2 min periods @ 160 Hz sampling rate. The The feature extraction applications use the required keys to process this data. In this case, the live signal has recorded data for a period of 2 minutes and the fake signal has recorded data for a period of 30 seconds.



```
      People  Class     Hurst  ...      SKEW  KURTOSIS       PFD
0          1      0  0.975042  ...  0.174661  0.262018  0.579023
1          1      0  0.871894  ...  0.106446 -0.294814  0.578537
2          1      0  0.831316  ...  0.328515  0.358776  0.579514
3          2      0  0.769326  ... -0.147464  0.079746  0.574166
4          2      0  0.659403  ... -0.095182  0.561877  0.564540
...      ...    ...       ...  ...       ...       ...       ...
1903     105      1  0.287296  ... -0.209089  0.337876  0.575276
1904     105      1  0.317583  ...  0.136049 -0.474434  0.584183
1905     106      1  0.342619  ...  0.129620  0.441926  0.576877
1906     106      1  0.309050  ...  0.088838 -0.040437  0.577819
1907     106      1  0.331672  ...  0.317441  0.649765  0.574829
```

Fig. 5. Features of the signal dataset

5) Feature Extraction methods: The following feature extraction techniques have been used
   a) Fourier Transform (BETA Band)
   b) Zero Crossing
   c) Zero Crossing Rate
   d) Discrete Wavelet Transform - Variance
   e) Discrete Wavelet Transform - Skewness
   f) Discrete Wavelet Transform - Kurtosis
   g) Petrosian Fractal Dimension
   h) Hurst Exponent

2) Normalization of the Data: Standard scaling has been used to normalize the signal data to improve the quality of results from the machine learning models.
3) Machine Learning Models used The following machine learning models were used in this project:
   a) Random Forest
   b) Support Vector Machine - Polynomial kernal
   c) Hierarchial Clustering - Unsupervised Learning
   d) Logistic Regression
   e) Multi-Layer Perceptron Classifier
   f) AdaBoost

IV. Results

A. Execution time analysis

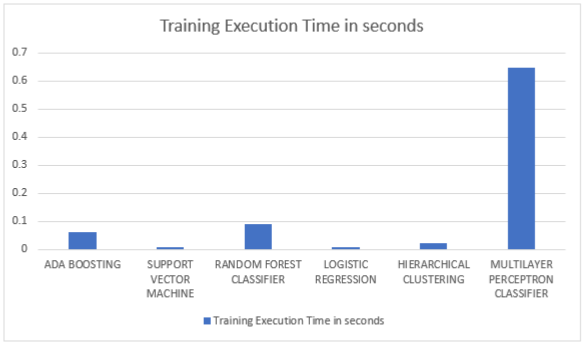


Fig. 6. Training Execution Time of the models

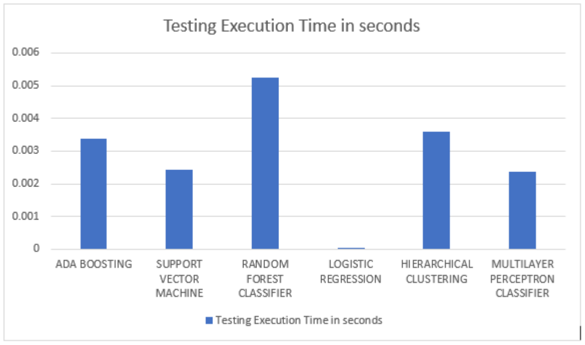


Fig. 7. Testing Execution Time of the models

This section contains the Execution Time results of both, the training (Fig 6) as well as the testing (Fig 7) phase for each Machine Learning model used.

B. Performance Parameters

1) Accuracy
   It is computed by finding the percentage of correct predictions of the brain signal data out of the total predictions made by the model. Fig 8 shows the accuracy of the Machine Learning models used in this project.
2) F1 Score
   It is defined as the harmonic mean of the precision and recall of the test. Fig 9 shows the F1 Scores of the Machine Learning models used in this project.

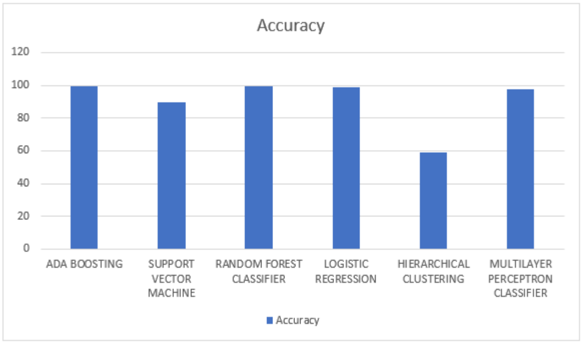


Fig. 8. Accuracy of the models

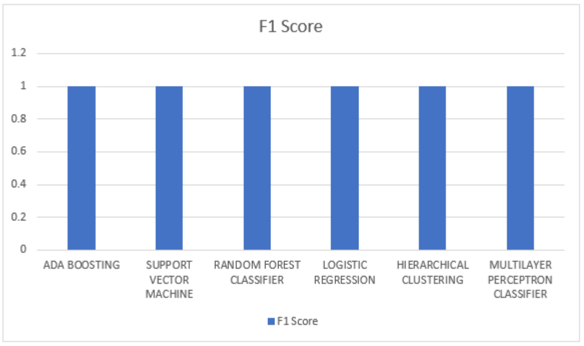


Fig. 9. F1 Score of the models

3) False Acceptance Rate
   It estimates the frequency at which the biometric system in question accepts false signals. Such false accepts can result in inaccurate results, unauthorized access and in some cases, even systemic failures. Fig 10 shows the False Acceptance Rates of the Machine Learning models used in this project.

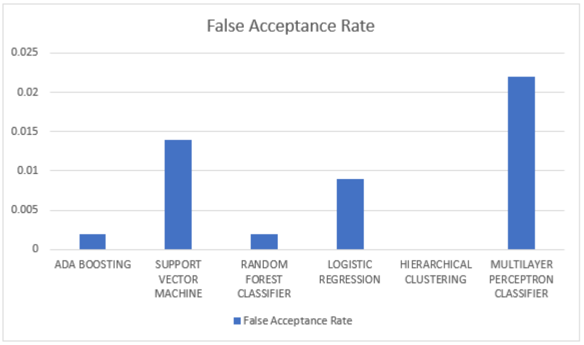


Fig. 10. False Acceptance Rate of the models

4) False Rejection Rate
   It is the percentage of events in which a legitimate user is denied access or rejected by the system. Fig

11 shows the False Rejection Rates of the Machine Learning models used in this project.

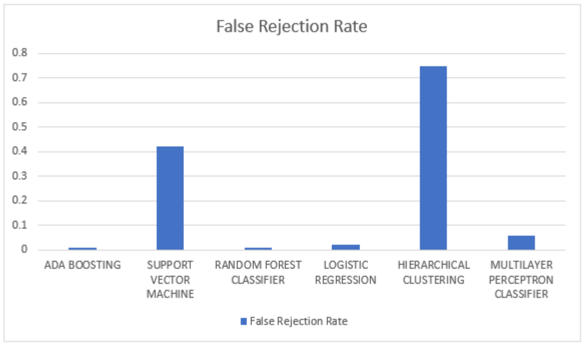


Fig. 11. False Rejection Rate of the models

5) Half Total Error
It is the average of the False Acceptance and Rejection rates. Fig 12 shows the Half Total Errors of the Machine Learning models used in this project.

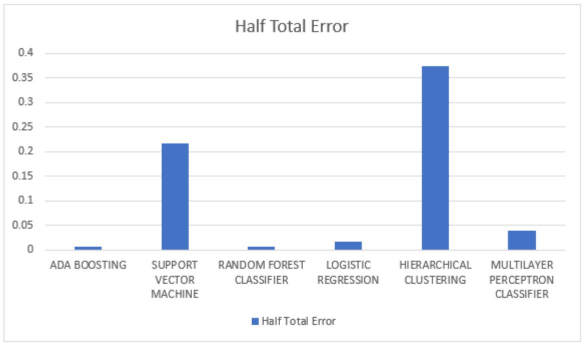


Fig. 12. Half Total Error of the models

V. Attack Vector Generation

To generate the attack vector, we added White Noise (AWGN) to the original signal. The noisy attack signal is then fed into the Machine Learning models and classification is attempted. Fig 13 shows the input signal before and after the white noise addition.

VI. Conclusion

Due to the susceptability of biometric systems to spoofing attacks, it is paramount that liveness detection and performance parameter analysis be incorporated into products in the field. In this paper, we discuss the implementation of such a solution. The framework consists of a mobile application designed using Android and a machine learning backend written in python, running on a flask server. Essentially, it uses an array of models to detect the liveness of a brain signal and compute performance parameters. As an additional enhancement feature, a voting model is employed to extract the best performing model for each signal and output the corresponding results on the mobile application's user interface. Lastly, an execution time analysis of both, the training and testing phases has been done to shed light on the applicability of each of the discussed models.

TABLE I
Completion of Tasks

No	Task	Assignee
1	Developing the Android Application	Nikhil Anantharaman
2	Normalization of the Dataset	Hemendu Roy
—	Implementing Feature Extraction Techniques	—
3	Fourier Transform (BETA Band)	Aditya Singh
4	Zero Crossing	Hemendu Roy
5	Zero Crossing Rate	Hemendu Roy
6	Discrete Wavelet Transform(DWT) - Variance	Thejas Naik
7	Discrete Wavelet Transform(DWT) - Skewness	Thejas Naik
8	Discrete Wavelet Transform(DWT) - Kurtosis	Thejas Naik
9	Petrosian Fractal Dimension	Nikhil Anantharaman
10	Hurst Exponent	Nikhil Anantharaman
—	Implementing Machine Learning Models	—
11	Random Forest	Aditya Singh
12	Support Vector Machine	Hemendu Roy
13	Hierarchical Clustering	Thejas Naik
14	Logistic Regression	Aditya Singh
15	Multi-Layer Perceptron Classifier	Nikhil Anantharaman
16	AdaBoost	Nikhil Anantharaman
17	Developing the Flask Server	Thejas Naik
18	Execution Time Analysis	Aditya Singh
19	Performance Analysis	Hemendu Roy
20	Noise Addition & Signal Generation	Aditya Singh

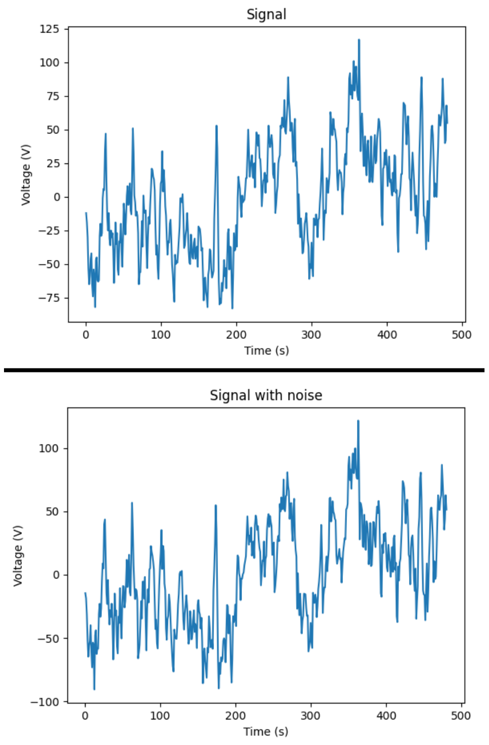


Fig. 13. The generated attack vector using white noise addition

VII. Acknowledgement

We would like to express our gratitude to Dr. Ayan Banerjee and Dr. Mohammad Javad Sohankar Esfahani for mentoring and guiding us. It is only with their help, that we were able to implement this project.

References

[1] Cao, Jiguo and Fan, Guangzh, 201, Signal Classification Using Random Forest with Kernels, doi 10.1109/AICT.2010.81

[2] S. Palazzo, C. Spampinato, I. Kavasidis, D. Giordano and M. Shah, "Generative Adversarial Networks Conditioned by Brain Signals," 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 3430-3438, doi: 10.1109/ICCV.2017.369.

[3] R. M. Mehmood and H. J. Lee, "Emotion classification of EEG brain signal using SVM and KNN," 2015 IEEE International Conference on Multimedia & Expo Workshops (ICMEW), 2015, pp. 1-5, doi: 10.1109/ICMEW.2015.7169786.