CSE 535 Fall 2017 - BraiNet Project Report
Group 1

Kannan Ganesan - kganesa8@asu.edu
Karthik Gunasekaran - kgunase2@asu.edu
Hari Kripa Omalur Chandran - homalurc@asu.edu
Pragna Munagala - pmunagal@asu.edu

ABSTRACT

The main aim of the BraiNet project is to identify the user using his brain wave signal to unlock his android phone. This report presents our application development in detail from data collection to application deployment. Decision tree machine learning technique is used to classify the user. Brainet application can also be thought as Thought Id similar to TouchId we have enabled in many smartphones available today.

Keywords

Data Preprocessing, Machine Learning, Decision Tree, Fog Server, Remote Server, Battery Level, EEG data, EDFread, Network Delay.

I - INTRODUCTION

Security in mobile applications is of primary concern for all users nowadays. Many secure unlock techniques are available in almost all the mobile devices like Touch Id, Lock Pattern, PIN, Face Recognition etc. But these techniques are not completely secure, for example Lock Pattern if seen by someone can be used to attack the mobile, Face Recognition is not twin compatible etc. To make it more secure, we are using human brain wave signal to identify the user and unlock his mobile if user is identified or else he cannot login to the application. Brain wave signals (EEG data) for a single user is collected in 14 different tasks like eyes opened, eyes closed, open or close left or right fist and so on. The classification of the user is done based on decision tree machine learning technique applied on the collected EEG data. The classifier model is deployed on both Fog Server (running in the same network as the mobile device) and Remote Server and the selected user data is sent from the android application to these servers one at a time and the classified user label is displayed on the mobile screen. In addition, the execution time of this pipeline, battery level of the mobile are also displayed. Based on network delay, power consumption, network availability, battery level the best server among Fog or Remote is chosen once user selects the Choose the Best option from the android application.

II - IMPLEMENTATION

The BraiNet application development involved the following steps: Data Collection and preprocessing, Classifier Implementation, Server Setup, Android Application and pipeline development.

A. Data Collection

The EEG dataset used for our application is collected from [1]. The dataset consists of 64-channel EEG collected from about 109 users under 14 different tasks, for example eyes opened, eyes closed, open or close left or right fist and so on. The complete description of each task used for data collection is described in [1].

B. Data Preprocessing

The EEG data collected from [1] was in raw edf format. We used MATLAB edfRead() function to read and process the edf format data. Once the data is processed we organized data into 64 feature vectors for each user and a user label is assigned to each user. All the users data is concatenated and a huge data set with feature vectors and class labels to be fed to the classifier is prepared. The processed dataset is stored into a csv file.

C. Classifier Model

Decision tree model with Gaussian Naive Bayes classifier is used to train our model. The entire dataset came up to 6GB which is very huge, so we limited our dataset to 10 users which is of 500MB. This data is fed into the classifier and decision tree .pkl is generated. Underlying classifier used is Gaussian Naive Bayes Classifier. The accuracy of the trained model turned up to

80%. The trained model is stored in decision_tree.pkl file which is used for predicting the user label for any given user input. The classifier was implemented using python's classification libraries. The classifier function takes in a file as input and returns the user number as output.

D. Fog Server

Fog Server is a server setup to run in the same network as the mobile device. We used python's Flask server as the fog server running in port 5000. We have deployed the app.py(python flask application doing the classification) and decision_tree.pkl files in the fog server. The classifier method defined in app.py is called from the android app using the server's ip. The user input feature vector is passed from the android app to the trained model in the server using the pipeline and the user label predicted is returned to the android application. The input brainwave signal of a randomly selected user coming from the android app is written into a file and this file is fed into the classifier.

E. Remote Server

Remote Server is an ubuntu server in AWS EC2 instance. This runs in a different network from the device. But we used the same python flask server. This server was made to run in port 5000.We have deployed the app.py(flask application doing the classification) and decision_tree.pkl files in the remote server. The classifier method defined in app.py is called from the android app using the server's ip. The user input feature vector is passed from the android app to the trained model in the server using the pipeline and the user label predicted is returned to the android application. The input brainwave signal of a randomly selected user coming from the android app is written into a file and this file is fed into the classifier.

F. Android Application

Our BraiNet android application consists of Fog Server, Remote Server, Choose the Best options on the user interface. When user selects the Fog Server/Remote Server, a random user is selected out of the 10 available users from the dataset and the selected user's feature vector is passed to the Fog Server/Remote Server where the users label is predicted and returned to the application. This user label along with the Execution time and Battery Label is printed on the UI. When the user selects Choose the Best option, based on the network available and the battery percentage remaining in the device a suitable server is chosen. When Wifi is available and battery level is good, Fog Server is chosen as the network delay of Fog Server is observed to be less than Remote Server. When there is no Wifi, only mobile network is available and the battery level is less, remote server is chosen as Fog Server won't be available.

III - APPLICATION DESIGN

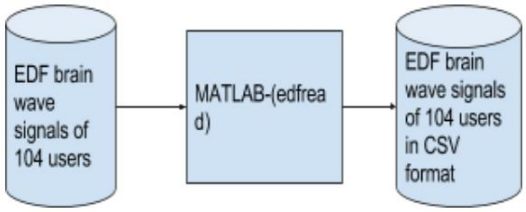

Figure.1

Figure.1 shows the data preprocessing of brainwave edf signals collected from [1].

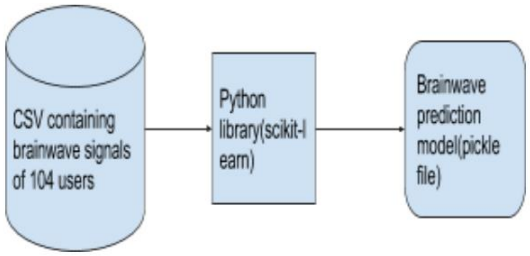

Figure.2

Figure.2 shows the architecture of the classifier that identifies the user based on his brain signal.

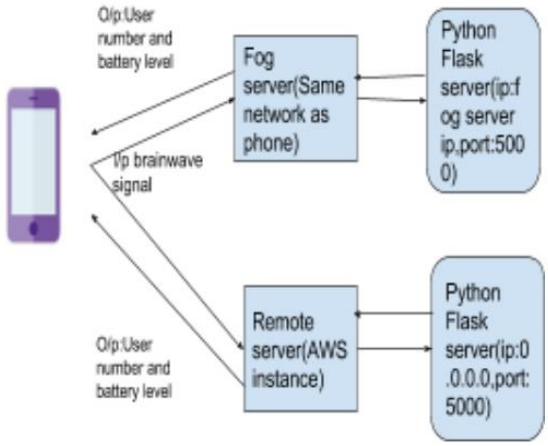

Figure.3

Figure3 shows the flow of our application from the user interface to the backend server which is deployed in both fog and remote instances.

IV - DEVELOPMENT PHASES

The table below shows the split up of work for our android application,classifier,backend server and the remote,fog instance setup.

S.No	Phase	Incharge
1	Data Collection	Hari Kripa
2	Data Preprocessing in MATLAB	Kannan
3	Classifier Model	Hari Kripa
4	Fog Server-Python Flask	Pragna
5	Configuring Python flask server in AWS instance	Kannan, Karthik
6	Android Application(Front end and Pipeline)	Karthik,Pragna

V - APPLICATION SCREENSHOTS

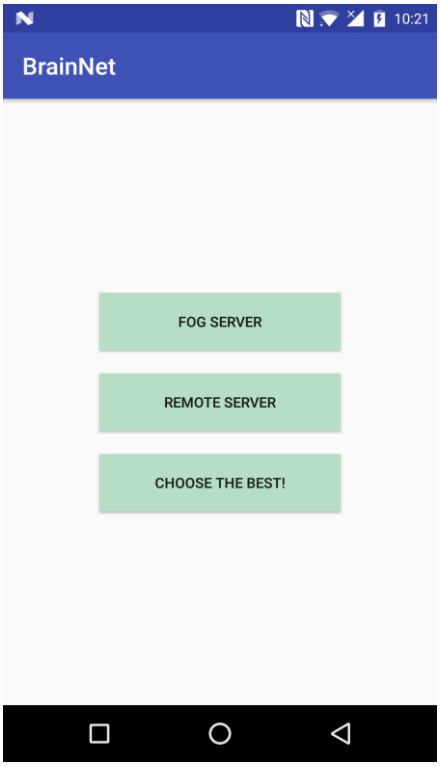

Figure 4. Home Screen of the Application

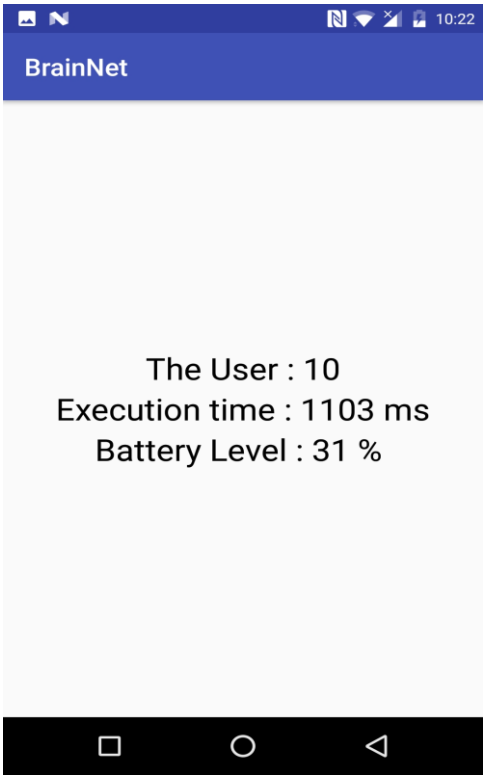

Figure 5. Results when Fog/Remote Server is selected

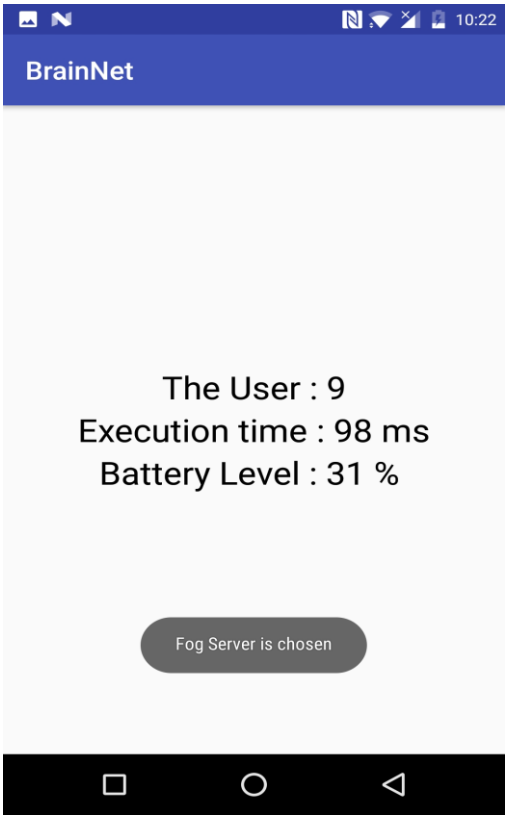

Figure 6. Scenario when Fog Server is chosen when Choose the Best option is selected

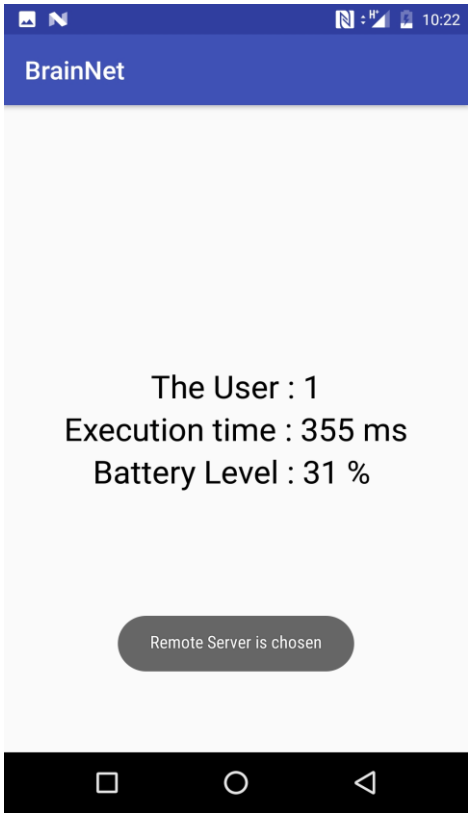

Figure 7. Scenario when Remote Server is chosen when Choose the Best option is selected

VI - CHALLENGES

The major challenges faced are:

Data Processing - As the data set is huge and in raw format, we faced few challenges in processing the data from raw format and then to convert it to csv format file to be fed to the classifier.

Server Setup - Setting up both Fog and Remote server and deploying the classifier model in the servers and accessing them from the application was a major challenge. During the setup of the remote server, we initially gave the ip of the amazon ec2 instance and started the python flask server. But we weren't able to hit the server from outside. After analysis[2], we changed the ip to 0.0.0.0 so that it's accessible without any hindrance. We also had to add an inbound rule[3] in the security group for our instance so that the flask server in our ec2 instance is accessible from anywhere outside.

Pipeline Work - Setup of the pipeline to post the request from the application to the server with input user data and then again returning with the response of user label from server was a challenge.

VII - CONCLUSION

User brain wave signal can be used to unlock the mobile making the system even more secure. The network delay of Fog Server was observed to be less than that of Remote Server. Since the built model's accuracy was close to 80%, the system predicts the users from their brain signal most of the times. We have used python's flask server as our backend server for both fog and remote servers. Amazon's EC2 ubuntu instance was used as the remote server.

VIII - FUTURE WORK

In our current application, we are using the collected brain signal data and displaying the results of the user and servers on the UI. Using the current android application, an end-to-end application can be developed in which user can be logged into the device using his brain wave signal being captured using brain wave sensor which can then be processed through the steps employed in our application and the user will be logged in based on the result from the server. In the system implemented so far, the trained model is deployed into the remote and fog server directly. An improvement could be made so that new users could be added directly and training takes place dynamically in both the servers. The model so far takes into consideration of the network status, battery level and execution time to choose between fog and remote server. This could be improved to also choose based on parameters like network delay or network bandwidth.

REFERENCES

[1]https://www.physionet.org/pn4/eegmmidb/
[2]https://stackoverflow.com/questions/31864499/how-to-enable-port-5000-on-aws-ubuntu
[3]http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/authorizing-access-to-an-instance.html