1. Introduction about OpenBcI Module

OpenBCI aims at making brain-computer interface easier to use with the development of

hardware and software tools based on biosensing and neuroscience. By using the robust

hardware platform along with the software kit, users can get access to brain wave data through EEG, EMG and ECG setups. The different boards have different channels ranging from 4 to 64. The Ganglion Board from OpenBCI is a low-cost and affordable unit to play around with brain wave data through EEG (Electroencephalogram). The Board has 4 channels and for each of the channel the sample rate is 200Hz. The board can be powered with from 3.3V to 6V dc battery. The board allows testing the impedance of the skin contact with the electrodes. While doing the impedance testing, the data stream is turned off. After this procedure is done the all the four channels will receive impedance values from the board.  The software developed by OpenBCI which is called OpenBCI GUI is a robust tool for streaming data from the OpenBCI Boards which can be seen live. The software is a powerful and easy to describe, visualization friendly software where data can be displayed in live. Data can be observed from Time Series plots and FFT Plots. From the control panel different parameters can be chosen and as required.

1. Data Collection process (Hardware)

For collecting data, the OpenBCI Ganglion board has been used. Electrodes have been used for receiving signals from four channels. For optimal result, a pair of noise cancelling headset has been used where the user is presented with periods of sound and others with no sound. The OenBCI GUI has been utilized for setting the channels parameter and for the primary visualization of the received signals.

1. Data Collection Process (Software)

MATLAB has been used incorporating the BrainFlow Library to collect data. There are several types of code have been used for experimenting with different outcomes.

Code 1: Beep\_EEG

This particular code is designed to collects data with total duration of 4 seconds of epochs which includes 2 seconds of sound and 2 seconds of no sound. In the 2 seconds of no sound, the marker has been set randomly for no sound. For the sound, the 2 seconds are fixed with marker. The Hrz is set for 500 and it runs upto 210 epochs. The sound epoch begins in 600 and ends in -600 in the marker column. On the other hand, the no sound epoch begins from 300 to -300 in the marker column. For this code, we have a random sleeping time in each of no sound periods or epoch.

The output would result in a file saved as a csv format with timestamp that contains the four channels and null values and the marker.

For the data cleaning, we remove the null values and only keep the channels and the marker.

Code 2: Different sound with different frequencies

For this particular code, the basic setup has been followed as the same as the code above (Beep\_EEG). For experimenting with different sound effects, different audio files of two seconds were downloaded, that were later played randomly each time for different files. It was made sure that each file was had the time duration of exactly two seconds.

For different experiments with this technique, used different frequencies have been used each time. For example, 1000hrz or 800hrz instead of 500hrz has been used to check the influences of the sound. Again, for one experiment if 500hrz was used, different sound files were played through the headset. It was made sure that they were 500hrz frequencies each time.

For another one, 800hrz may have been used with different sound files but with 800hrz(the same files above in different frequencies)

Code 3: Cleaned Output data code

This code design is the same for the basic setup. The only difference is, instead of manual data cleaning (DC removal and Normalization) which were required in the past two sets of codes above, this code does it by itself. The output data would be Direct Current (DC) removed and normalized with normalization function along with other columns.

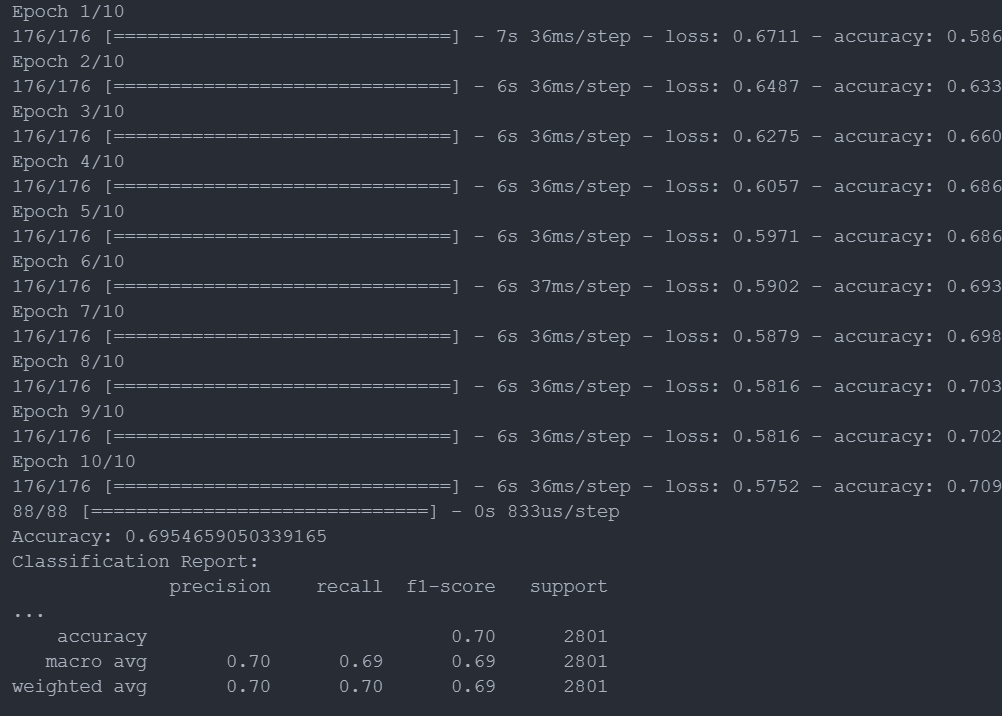
1. Data Cleaning Process
2. Open the raw data and delete all values except the channels and the marker
3. Get rid of unnecessary channels data (if any)
4. Create a time column
5. Get average of raw Data of Each Channels
6. Remove DC
7. Normalize Data
8. Data Pre-processing for applying ML and AI Models
9. Cleaned data has been split in two different files that contains all sound epochs in one file and all no sound epochs in another. For Example, all the values were marked from 600 to -600 for the sound and were made into one single file. Now the sound file only contains epochs of sounds. The same process for the no sound epochs was followed (for epochs of 300 to -300) to create a different file of no sound.
10. Sound files have been labeled as 1 and 0 accordingly for sound and no sound for the training of Machine Learning models.
11. Both data files have been combined after importing the datasets in TensorFlow.

1. ML and AI Models application

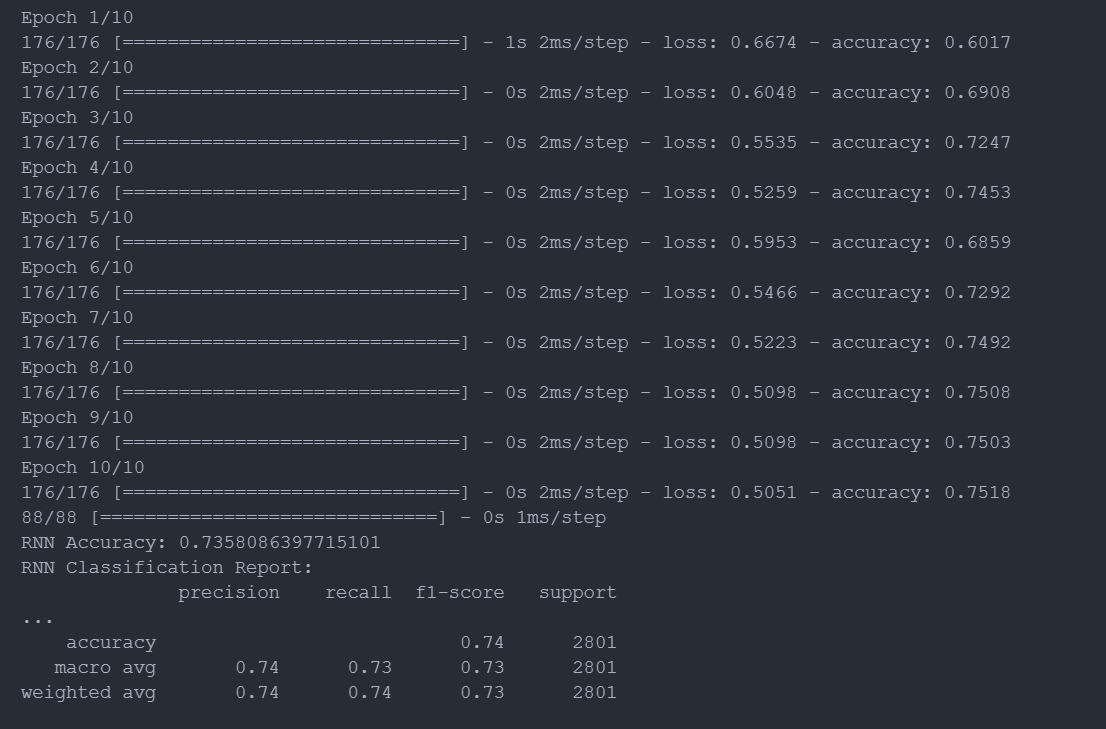
Possible models that may be suitable for the experiment:

1. Convolutional Neural Networks (CNN): CNNs are excellent at automatically detecting important features without needing any specific instructions. They can capture spatial hierarchies in data, which is beneficial for time-series data like EEG. CNN can also be computationally intensive and require more data to train effectively. Without enough data, they might be overfit to the training set.
2. Recurrent Neural Networks (RNN): Ideal for sequence data they can process data points related to previous ones. It can remember previous inputs due to internal memory. RNN Struggle with long sequences due to the vanishing gradient problem. Slower to train, can be computationally intensive.
3. Long Short-Term Memory (LSTM) Networks: Can learn long-term dependencies in data which RNNs struggle with. Often outperforms RNNs in practice, especially for longer sequences. It is more complex and take longer to train than traditional RNNs. It requires careful tuning of parameters to prevent overfitting.
4. Random Forest Classifier. Can handle outliers and nonlinear data well. Less need for feature scaling and data preprocessing. Not as interpretable as simpler models like decision trees. May not perform as well as neural networks for complex patterns.
5. Support Vector Machines (SVM). Good for high-dimensional spaces and effective in cases where the number of dimensions exceeds the number of samples. It can model non-linear relationships. Disadvantages: Not suitable for large datasets. Kernel and regularization parameters can heavily influence the performance.
6. Results of each model:

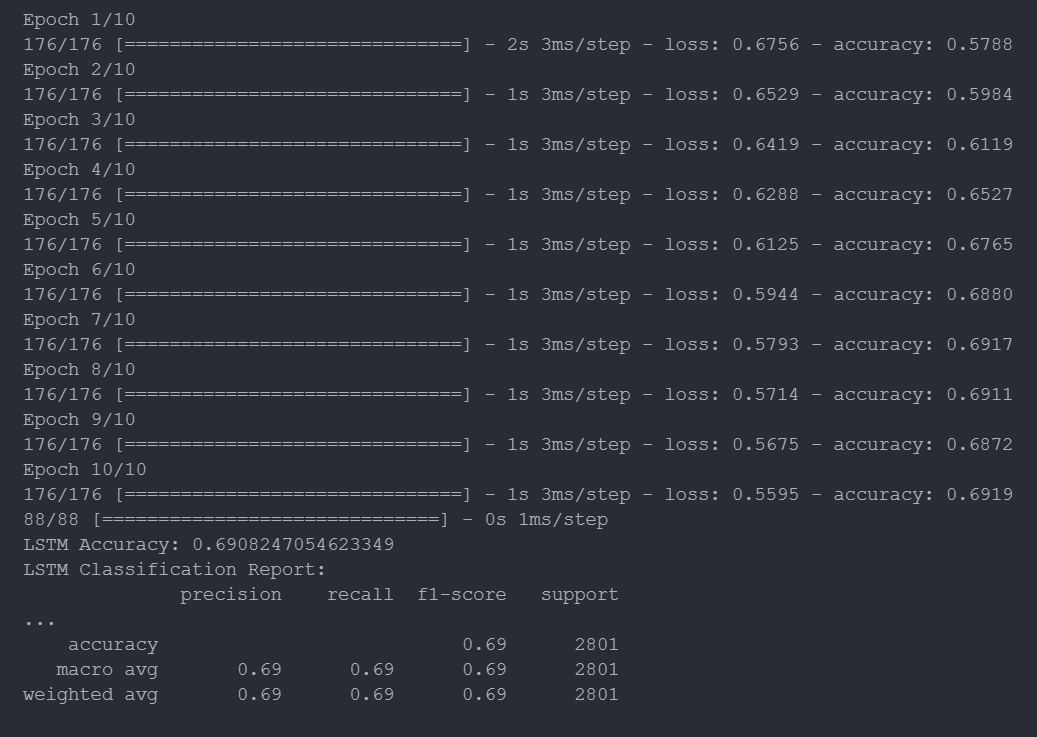
CNN:



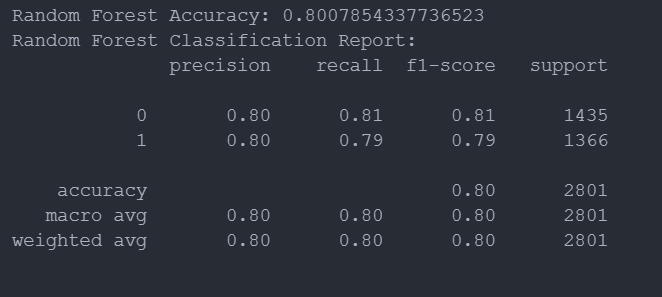
RNN:



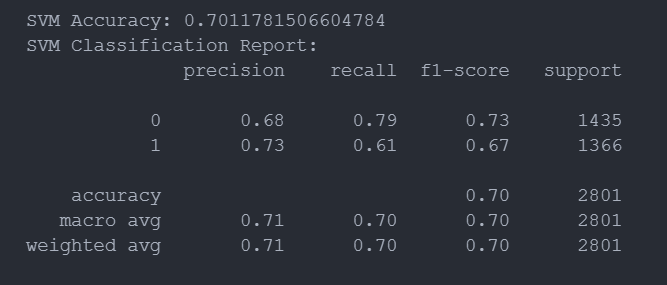
LSTM:



Random Forest:



SVM:



It looks like CNN, RNN, and Random Forest are great choices.