Epileptic seizure detection from intracranial EEG

ECE 5040: Introduction to Neural Engineering Final Project

Michael Xiao
Electrical and Computer Engineering
Cornell University
Ithaca, NY
mfx2@cornell.edu

2nd Russell Silva
Electrical and Computer Engineering
Cornell University
Ithaca, NY
rms438@cornell.edu

3rd Isabella Salas-Allende
Biological Science
Cornell University
Ithaca, NY
ias44@cornell.edu

***Abstract*—Being able to predict and detect seizures accurately would open many new doors to closed-loop stimulation treatments for patients with epilepsy. Unfortunately, detecting seizures is a challenging task due to its unique characteristics that vary greatly from patient to patient. However, with the development of modern machine learning techniques, this patient individualized detection is becoming more and more feasible. In this paper, we describe our development a machine learning algorithm able to be trained on pre-recorded labeled data to effectively distinguish a seizure state from a non-seizure state using intracranial electroencephalogram (iEEG) recordings. Our methods use 26 different features and three different classifiers to achieve a top performing algorithm on the Kaggle data science competition platform.**

I. Introduction

Epilepsy is a chronic neurological disorder that affects over 1 percent of the US population, greatly lowering the quality of life for many patients [1]. To make matters worse, for up to 40 percent of patients diagnosed with epilepsy, medications are not effective, meaning that traditional clinical solutions often have little effect [2]. Furthermore, even surgical solutions of removing epilepsy-causing brain tissues have lasting spontaneous seizures. These seizures greatly disrupt everyday life as even if they do not happen frequently, they bring a great deal of anxiety even with the possibility of a seizure occurring.

A more modern solution that has recently been developed is neurostimulation. In the past continuous stimulation has been tested but found to have significant neuronal damage in the surrounding tissue over time due to the frequency of pulses [3]. As a result, closed-loop responsive neurological stimulation (RNS) is necessary for a safe and effective solution. RNS works by sending a biphasic pulse to counteract a seizure onset to stop it before it begins to spread to other areas of the brain. To do this however, the brain must be constantly monitored a seizure must be first detected to then send a signal to the neurostimulator. This proves to be the greatest barrier that RNS faces as physicians must review very large quantities of continuous EEG data to identify qualities of seizures, which are not only very subtle, but also tend to vary greatly on a patient-to-patient basis.

Luckily, with the development of machine learning, we can now automate this process of creating an individualized classifying system able to sort through a specific patients data and detect seizures. However, these machine learning algorithms must be reliable, with low false positive and false negative rates, in order to function as a false detection can have drastic consequences for the patient. To push the boundaries of the performance of these algorithms, competitions and open source libraries and documentation have been established to share and grow code and methods. Kaggle has become a popular site on which these sort of data science competitions are held, hosting a platform to support the data and foster a healthy competition for the advancement of the field. We compete in one of these competitions in order to detect seizures and separate ictal from non-ictal states. Section 2 will describe the objectives of this Kaggle competition, section 3 will highlight the methods we used including preprocessing, features, and classifiers, section 4 will summarize our results and findings, and section 5 will conclude our work and suggest future improvements.

II. Objectives

The goal of this project is to classify two states associated with a seizure: non-ictal (i.e., non-seizure or normal) and ictal (i.e., seizure). One important difference to note in this work compared to state-of-the-art algorithms is our work of detection rather than prediction. The cutting edge of this field is working to currently distinguish between the non-ictal and preictal states to stop seizures before they even happen. In our case, we aim to merely detect seizures, a slightly easier task as the ictal and non-ictal states have more clear differences.

To create this model, we are provided with several datasets via Kaggle including iEEG recordings from 7 patients. Within this set of data, there are sampling frequencies of 500 and 5000 Hz, varying number of recording channels, and labelled data. This data is separated into ictal training samples and non-ictal samples, labelled by an expert epileptologist prior to the competition. These details are highlighted below in table 1.

III. Methods

Our algorithm can be separated into three broad sections. First, all of the data is run through a preprocessor in order to eliminate some noise and make feature extraction more streamlined. Next, different features were extracted from the

TABLE I
TRAINING DATA SUMMARY

Patient	1	2	3	4	5	6	7
Sample Frequency (Hz)	5000	500	500	500	500	500	500
Number of Channels	96	56	16	88	104	88	96
Number of Ictal Training Samples	218	191	296	424	150	313	307
Number of Non-Ictal Training Samples	600	900	900	1200	660	2100	2400

time and frequency domain and stored into a separate dataset. Lastly, the feature data was used to feed into our classifiers in order to train our model using different techniques.

A. Preprocessing

Our preprocessing step was a very light computation which simply readies the data for feature extraction. First, the input signal is parsed to search for any values of NaN (Not a Number). These NaN values are discarded and replaced with zero. Next, we utilized common average referencing (CAR) as seen in other works. CAR is performed by simply subtracting the overall mean of the signal from each index of the signal. This has been shown to reduce noise by up to 30% and reduce stimulus artifacts [4]. An example CAR preprocessing step from a non-ictal signal from patient 3 is shown in figure 1.

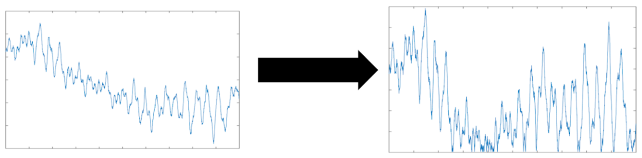


Fig. 1. Common average referencing performed on a non-ictal channel from patient 3.

TABLE II
SUMMARY OF FEATURES

Feature	Summary
Line Length	$\sum_{i=2}^{N} \lvert x_i - x_{i-1} \rvert$
Energy	$\sum_{i=1}^{N} x_i^2$
Variance	$\frac{\sum_{i=1}^{N}(x_i - x_{avg})^2}{n-1}$
Maximum	$max(x_i)$
Band power	$\theta, \alpha, \beta, \gamma, highfrequency$
Correlation Coefficients	Comparing relationships between channels
Mobility	$\sqrt{\frac{var(x')}{var(x)}}$
Variance	$\frac{mobility(x')}{mobility(x)}$
Wavelet	Mean, median, max, min of DWT
Zero Crossing Histogram	samples between separate zero crossings

B. Feature Extraction

Throughout our work, we experimented with many different features spanning across the time and frequency domains. A table of all of the features we implemented is shown below in Table 2 along with a summary of how each are calculated. After finalizing our features, we implemented a useful method that would perform the feature extraction on the patient data and store them in new excel files with the feature data concatenated. This way, when we moved on to classification, we did not have to rerun our feature extraction process, which saved processing power in our limited time constraints. The features we used are further discussed in the remainder of this section.

1) The Basic Features: line length, energy, variance, maximum, bandpower: Beginning our exploration of features, we started with the features we were already very familiar with and had used before: line length, energy, variance, maximum, and bandpower. LLine length is calculated by taking the sum of the differences between individual samples of a channel. Energy is calculated by summing the squares of each sample in a channel. Variance is a measure of how spread out a sample is and can be calculated by summing the differences between samples from the mean of the signal. Maximum simply finds the highest amplitude of all samples in the signal. Lastly, bandpower is calculated by taking the Fourier transform of the signal and finding the area under the curve between specified bands. Based on past literature, we noticed that there were five primary bands that many papers experimented with: (04 Hz), (48 Hz), (812 Hz), (1230 Hz), and (¿30 Hz). To start, we implemented two of these bandpower features from 12-30 Hz and from 100-250 Hz. Throughout this process, the numpy library was very helpful in efficiently coding and executing the feature extraction. With these basic features and a random forest classifier, we were able to achieve an accuracy score (calculated by area under ROC curve on Kaggles test dataset) of 0.84629. Though this score was decent, we wanted to explore other features to improve our score.

2) Correlation Coefficients: One of the first additions we tried to make was adding correlation coefficients. Correlation coefficients were calculated for each pair of channels yielding features for the number of permutations of two channels there were. This number could be calculated by equation (1) where c represents the number of correlation coefficients and n represents the number of channels.

$$c = \frac{n * (n + 1)}{2} \tag{1}$$

Then, for each permutation of length 2, we used the numpy argument, corrcoef, to calculate the Pearson product-moment correlation coefficients as shown in table 2. This gave us a measure of the strength of the linear relationship between two channels, which provided our model some temporal-spatial awareness of the signals.

After implementing correlation coefficients, we ran a diagnostic test to evaluate the importance of each feature. This was evaluated by comparing the result of removing a particular feature on the overall performance. If the model performed much worse without a feature, that feature had a greater importance. We ran this for each of the features we had implemented up to this point and normalized our results to graph, as seen in figure 2. As shown, correlation coefficients had a very minimal importance among all patients, much lower than all other features for every patient. This was a discouraging start to our feature exploration but was a good primer on how to add new features in the future. Additionally, with this knowledge, we removed the correlation coefficients from our code to save on computation time and added additional bandpower features due to the high importance we saw. The new bandpowers we added were the theta band (4-8 Hz) and the alpha band (8-12 Hz). This was pretty simple to implement in our existing framework as we had already written a bandpower function and created a very easy way to add on new features in our code.

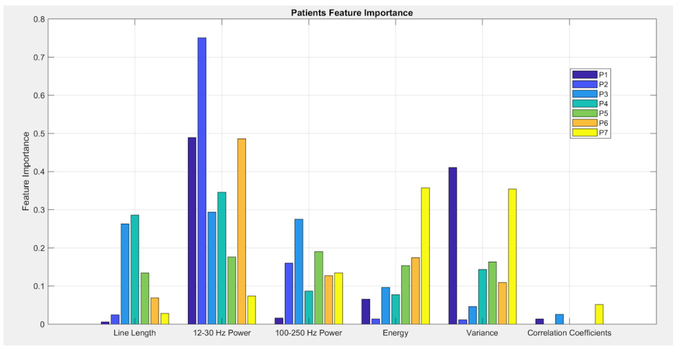


Fig. 2. Feature Importance Graph.

3) Hjorth Parameters: Hjorth parameters provide us with two new features, mobility and complexity, that are additional indicators of statistical properties in the time domain. In previous works, it was found that the addition of Hjorth parameters were able to improve the performance of EEG-based BCI systems by an average of 4.4% [5]. As seen in table 2, these features are variations on the variance and derivative of variance of a signal and are used for further analysis on determining how spread a signal is. Mobility is a measure of the proportion of the standard deviation of the power spectrum and is calculated by taking the square root of the ratio of the variance of the first derivative of the signal and that of the signal. Complexity is a comparison of the signal to see how much it resembles a pure sine wave and can be calculated by taking the ratio of the mobility of the first derivative of a signal and the signal itself. These three parameters are useful as they are able to indicate information about the frequency spectrum of a signal and the time domain. Additionally, because we utilize three parameters that stack or make use of each other, the computational cost of each is low.

4) Wavelet features: The discrete wavelet transform (DWT) is a joint time-frequency analysis that has proved to be an invaluable tool for the study of EEG signals in the past [6]. DWTs main advantage over other transforms is that because it discreetly samples waveforms, it is able to capture the instantaneous frequency in time of samples. To perform the DWT in our software, we made use of the PyWavelets library, an open source and easy to use transform software for python [7]. There are several flavors of DWTs, with the primary two being Haar and Daubechies. Haar DWTs are the most basic orthonormal wavelet transform and utilizes non-overlapping windows to produce a computationally cheap and memory efficient comparison between adjacent pixel pairs. A Daubechies wavelet transform is more complex and uses overlapping windows, with the size usually specified by the number following it. Because Daubechies averages over more pixels with an overlapping window, it produces a smoother result. In addition to type, DWT can also be given a decomposition level parameter which determines the number of detail coefficients the DWT function outputs.

In our model, we utilize a Daubechies-1 DWT with four levels with the call:

```
cA4, cD4, cD3, cD2, cD1 = pywt.wavedec
(signal, ’db1’, level=4, axis=0)
```

. For each of these four detail coefficients, we then take the mean, median, maximum, and minimum and collect them for our features. This gives us 16 additional samples to work with, providing our model insight into the temporal-frequency relationships within the signal.

5) Zero Crossing Histogram: The last feature we tried to implement was the zero crossing histogram (ZCH) as we saw its promising use in several different papers to detect Alzeimers disease, Vascular dementia, predicting seizures, and characterizing sleep spindles [8][9]. ZCH is useful because it is more robust in the presence of contaminating artefacts and ignores most of the distributed noise in the signal. An important first step we took for ZCH was performing CAR on it so that it was centered around the zero mark. It is then implemented by parsing through the signal and storing the indices in which the signal changes from positive to negative or vise versa (when it crosses zero). Then we would take the first derivative of this index to yield the amount of samples between individual zero crossings. These values were then averaged to yield the final feature.

The main challenges we saw with the ZCH was its very long computation time. It almost doubled the time it took for feature extraction to execute and unfortunately, yielded results that were not very accurate when cross validating. As a result, on our final run, we decided to omit our implementation of the ZCH despite our promising research on the feature.

6) Recursive Feature Elimination: One last layer of robustness implemented in the system was recursive feature elimination, which essentially omits the least important portion of the features. Feature importance was calculated with the decrease in performance with the removal of the particular feature. This means that the most important features see the largest decrease in performance when not included in the training data. Next, a threshold was set for what portion of features are removed. In our testing, we tried 50%, 40%, and 25%, with 25% removal rate seeing the best results. Lastly, the system would recursively consider smaller and smaller feature sets (step size = 5%) until the threshold for feature removal was achieved. This was implemented in our code using Scikit Learns RFE function.

C. Classification

1) Three Layer Classifier: After our features had been extracted and stored into our new data files, we then ran them through our classification system, consisting of a random forest classifier, an adaptive boosting classifier (AdaBoost), and an extreme gradient boosting classifier (XGBoost). Each of these classifiers was run independently on our feature data and the results from the three were averaged together to create our final and best performing submission.

The first classifier we used was random forest, a well established and well researched classifier used in the neural world [10]. Random forest classification uses an ensemble learning method which utilizes a randomly selected subset of the training set to create a forest of many decision trees. This is done in conjunction with averaging to improve the predictive accuracy and control the degree of overfitting the data. To classify the input, each tree will vote on the status of the data and the aggregated votes are used to decide the final class of the test subject.

The second classifier utilized was AdaBoost, which we found also commonly used in EEG studies [11]. Adaptive boosting is a learning algorithm that creates a strong classifier from many weaker classifiers by making predictions calculated on a changing weighted average of other classifiers. This is done in a pseudo random way by weighting (or boosting) classifiers and seeing if the results are better or worse than the previous copy. Classifiers are recursively reweighted until the best features and thresholds are heavily weighted and combined.

The last classifier we tried was XGBoost, one actually most commonly found in Kaggle competitions, recently dominating many of the leaderboards. XGBoost is similar to AdaBoost in that it utilizes weighting. It begins with a naive model and then calculates the errors from the model, creating a gradient. The next model created takes into account the error from before and moves its weighting against the gradient that was produced from the previous model, effectively optimizing the boosting parameters. The classification model is based on the gradient descent algorithm to minimize loss.

We found the scikit learn python library to be immensely helpful in implementing all of these classifiers. This was a very effective approach for us because the classifiers were able to negate the weaknesses and shortcomings of one another and provide a layer of redundancy in our model.

2) Hyperparameter Bayesian Optimization: One additional step we did in each of our classifiers was optimize the parameters fed into the classifier call. The hyperparameters for each classifier are shown below in table 3. Grid search is the traditional method used to optimize these parameters and is implemented by creating a large multidimensional grid of the different parameters and iterating across them one by one to search for the optimal hyperparameters. Though this yielded good results, it often took us a very long time to search and the range that we could search in was greatly limited by the O(n) complexity.

To optimize along a greater range in a shorter time, we utilized Bayesian optimization with the scikit-optimize module. Bayesian optimization makes use of Bayes Rule, described in equation (2) in which it makes a prediction given a condition. In optimization, this is useful because each iteration of the Bayesian optimization can jump across great bounds, rather than being limited to having to search entry by entry to arrive at a greater number in grid search. For example, if a low number of estimators performs very poorly, the optimizer can simply try again given that result, to a very large number of estimators. In this way, the optimizer is able to arrive at good hyperparameter values as soon as possible.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2}$$

In each iteration, the hyperparameters chosen for the classifier are compared by cross validation. This is done by segmenting the training data into four parts. Three of the data sets become training sets and the last becomes a test data set. The testing set can be rotated around in four different ways to extensively test the parameters. Using this system, we were able to find the best performing parameters for our classifiers with the computer rather than choosing them by hand.

TABLE III
HYPERPARAMETERS OPTIMIZED FOR EACH CLASSIFIER

Random Forest	AdaBoost	XGBoost
n_estimators	n_estimators	n_estimators
max_depth	learning_rate	learning_rate
min_samples_leaf		max_depth
max_features		min_child_weight
min_samples_split		gamma

IV. Results

The majority of our testing results was performed upon the held-out unlabeled data on the Kaggle competition page. This data was comprised samples from each of the 7 patients, totalling to 17,819 total iEEG recordings. Our machine learning algorithm then took these data sets as input and output a probability of ictal state metric (between 0 and 1), utilizing the prior training and feature extraction described above. The results were saved in a text file and then transferred into a .csv file to be formatted for submission.

Each team was allowed 15 submissions per day in order to prevent guessing and overloading Kaggles server. These entries were scored using the area under the receiving operating characteristic (AUROC) curve. This curve is composed of plotting sensitivity, or the true positive rate, against 1-specificity, or the false positive rate. A good curve is shaped like a bulge toward the top left corner, with a better score being closer to the corner. Equations for sensitivity and specificity can be seen in equations (3) and (4). The public score available to all teams before the deadline was our model evaluated using just 30% of the total test data. The private leader board was comprised of the remaining 70% of the data.

$$sensitivity = \frac{truepositives}{truepositives + falsenegatives} \tag{3}$$

$$specificity = \frac{truenegatives}{truenegatives + falsepositives} \tag{4}$$

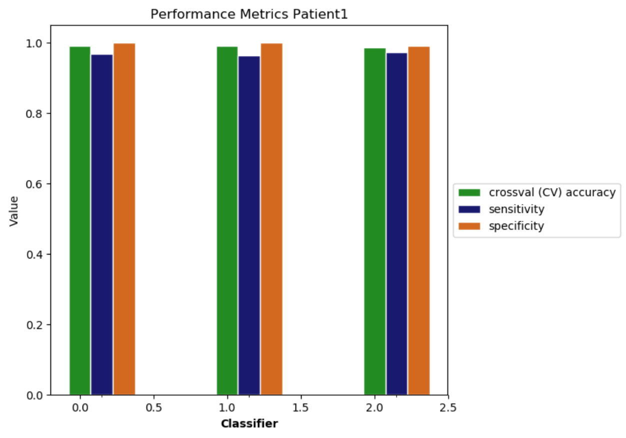
Using this structure for results, our team submitted a total of 39 different entries from various stages in our models development. Our most simple model with just the basic parameters scored a 0.84629 AUROC. With our adjustments and additions, we were able to achieve a high public score of 0.92791 AUROC, winning our competition. With the private data, our score dwindled slightly at 0.91806 AUROC, but we still remained at the top of the leaderboards. Our biggest jumps in scores occurred from when we added more features, when we introduced feature removal, and when we began averaging different classifiers. Unfortunately, we were unable to quantify the effectiveness of each of these strategies due to the fact that each built upon previous work and our increase in performance scaled marginally less as we started nearing the 0.9 mark.

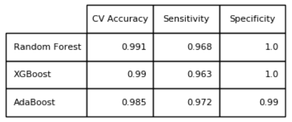
In addition to the AUROC performance, we were able to retrieve the sensitivity and specificity for each patient, utilizing the equations (3) and (4) above. Our results can be seen in table 4 below. As seen in the table, we achieved consistently high specificity across all patients while there was a fair amount of deviation within sensitivity. The most glaring example of this was patient 5 as our model had a lot of trouble classifying this particular patient.

TABLE IV
SENSITIVITY AND SPECIFICITY FOR EACH PATIENT

Patient Number	Sensitivity	Specificity
1	0.968	0.998
2	0.958	0.993
3	0.878	0.987
4	0.851	0.995
5	0.347	0.988
6	0.837	0.989
7	0.967	0.995

Lastly, we were able to compare the performances of the different classifiers. To do this, we once again used cross validation to train and then test each individual classifier on data from patient one. As seen in figure 3, each of the classifiers on their own already performed pretty well with the features we were working, achieving over 96% sensitivity. The Adaboost showed the most variance, achieving a slightly higher sensitivity at the cost of a weaker specificity score.





	CV Accuracy	Sensitivity	Specificity
Random Forest	0.991	0.968	1.0
XGBoost	0.99	0.963	1.0
AdaBoost	0.985	0.972	0.99

Fig. 3. Classifier Performances on Patient 1

V. Conclusion

This project was a very rewarding experience and helped introduce our group to a lot of the different aspects of machine learning in iEEGs. Our interest in the class material definitely peaked while we were working on the project as we had the freedom to try whatever new approaches we could find. We spent a lot of time reviewing literature and libraries and spent a lot more time creating and modifying our algorithm. It was exciting to see our model come together after many hours of hard work we put in despite the ups and downs we had throughout the project. Machine learning algorithms require a lot of grit from a software perspective. We saw this firsthand as many of our implementations that we spent a lot of effort creating simply were not effective and, in some cases, even made our model worse.

Though effective, our model was by no means perfect, as seen in table 4. This really shows how different seizures can look from patient to patient, and that there is no model that works for 100% of people, once again highlighting why machine learning can be so effective in this particular use case. It felt a little better to know that many of the other teams were also struggling with this particular patient and in the future, we would love to try to look more into the patients specific data.

If we had more time to improve our algorithm, we did have a few more items we wanted to implement. First, we wanted to try to implement more complex features we had found from literature. This includes phase locking value, cross frequency coupling, autoregressive coefficients, phase synchronizations, among many others. Though adding more features adds additional complexity to our code, our framework of separating feature extraction and classification in two discrete steps makes it easy to test our features. More features can manage to capture more of the signals discrepancies and relationships that could open new doors to how we look at the signal. Furthermore, we would have liked to try weighting the classifiers as well. Currently, the results from our three classifiers are simply averaged together, but if we could add more classifiers and run another layer of optimization among those classifiers, we could add even more levels of redundancy in our mode..

Overall, this was a great introduction into the multidisciplinary realm of neural engineering and I really enjoyed working with my team of very diverse backgrounds and skills. I am very hopeful to see how the technology grows as machine learning continues to mature and reading through literature has made me confident in the foundation we have already laid for effective closed-loop neurostimulation devices.

References

[1] C. Begley, M. Famulari, J. Annegers, D. Lairson, T. Reynolds, S. Coan, et al. The cost of epilepsy in the United States: an estimate from population-based clinical and survey data *Epilepsia* 2000; 41:34251.

[2] National Institute of Neurological Disorders and Stroke, The Epilepsies and Seizures: Hope Through Research, *Patient and Caregiver Information*, Aug 8, 2018

[3] J. Politsky, R. Estellar, A. Murro, J. Smith, P. Ray, Y. Park, et al. Effects of electrical stimulation paradigm on seizure frequency in medically intractable partial seizure patients with a cranially implanted responsive cortical neurostimulator. In: *Proceedings of the Annual Meeting of American Epilepsy Society* (AES), 2005.

[4] G. Oleary, D. Groppe, T. Valiante, N. Verma, and R. Genov, NURIP: Neural Interface Processor for Brain-State Classification and Programmable-Waveform Neurostimulation, *IEEE Journal of Solid-State Circuits* Vol. 53, No. 11, Nov 2018

[5] S. Oh, Y. Lee, H. Kim, A Novel EEG Feature Extraction Method Using Hjorth Parameter, *International Journal of Electronics and Electrical Engineering* Vol. 2, No. 2, June, 2014

[6] R. Panda, P. S. Khobragade, P. D. Jambhule, S. N. Jengthe, P. Pal, and T. Gandhi, Classification of EEG signal using wavelet transform and support vector machine for epileptic seizure diction, in *International Conference on Systems in Medicine and Biology* (ICSMB), pp. 405 408, 2010

[7] G. Lee, R. Gommers, F. Wasilewski, K. Wohlfahrt, A. OLeary, H. Nahrstaedt, and Contributors, PyWavelets - Wavelet Transforms in Python, 2006-,https://github.com/PyWavelets/pywt [Online; accessed 2019-5-1].

[8] L. Wijesinghe, D. Wickramasuriya, and A. Pasqual, Generalized Preprocessing and Feature Extraction Platform for Scalp EEG Signals on FPGA, *IEEE Conference on Biomedical Engineering and Sciences*, Dec 8-10 2014

[9] A. Zandi, R. Tafreshi, M. Javidan, G. Dumont, Predicting temporal lobe epileptic seizures based on zero-crossing interval analysis in scalp EEG in Annual International *Conference of the IEEE Engineering in Medicine and Biology Society*, 2010

[10] D. R. Edla, K. Mangalorekar, G. Dhavalikar, S. Dodia, Classification of EEG data for human mental state analysis using Random Forest Classifier, *International Conference on Computational Intelligence and Data Science*, 2018

[11] P. Das, A. Sadhu, A. Konar, B. Bhattacharya, Adaptive Parameterized AdaBoost Algorithm with application in EEG Motor Imagery Classification, *2015 International Joint Conference on Neural Networks*, July 12-17, 2015