A Study of Machine Learning Approaches to classify EEG Signals

Anway Bose
Department of Electrical Engineering
Temple University, Philadelphia
Philadelphia, PA, 19122

Introduction

An *Electroencephalogram* (EEG) is a complex multi-channel signal that captures the electrical activity of the brain by measuring voltages at various locations on the scalp. The interpretation of EEG recordings is facilitated by using montages that redefine channels as the difference between channel voltages.

The objective of this project is to classify EEG time-series signals to detect the occurrence and ending of five different events. To achieve this goal, we will employ two *Machine Learning* (ML) approaches: a generative *Random Forest* model and a *Neural Network*.

In the rest of the report we will introduce the dataset in the next dataset section. The next two sections will introduce the two models used following a results section. An at last the report is concluded with a conclusion.

Dataset

The complete dataset comprises 15 sub-datasets, out of which the first 10 are utilized to generate the training set. The remaining two sub-directories, namely /dev and /eval, each consist of two sub-directories. Within each sub-directory, multiple sessions of EEG signals are stored.

Each session comprises a distinct series of EEG signal amplitudes, with each series defining the onset and conclusion of either of five distinct events or the background event.

Given the inconsistency in the duration of each event, the data was pre-processed by dividing it into 10 consecutive signal amplitude windows. Subsequently, the data was re-sampled and redefined such that each session comprised sequences of events with 10 amplitude windows.

Given that a significant portion of the dataset corresponds to background events, and each event is characterized by different amplitudes, the data was transformed into the *Principal Component Analysis* (PCA) domain.

In the following two sections I will discuss my approaches of two different ML models to learn the occurence of an event from these re-sampled redefined dataset.

Algorithm 1: Convolutional Neural Network (CNN)

My motivation of using a CNN is from the use of a microscope to find the hidden feature in a sample.

My CNN consists of 3 Convolutional layers, one Long Short Term Memory (LSTM) layer following two fully connected layer. For implementation I defined an *EEG_CNN* class that inherits from PyTorch's *nn.Module* class. The constructor initializes the layers of the network follwing by forward pass of the network.

- The forward function takes in the input ECG signal x and passes it through the layers of the model.
- The input signal is first passed through the three convolutional layers, and ReLU activation function is applied after each convolutional layer.
- We then apply max pooling with a kernel size of 2 to downsample the output of the convolutional layers.
- The output is then transposed and passed through the LSTM layer, which processes the sequence of features from the input in a batch-oriented manner and returns the output hidden state and cell state at each time step.
- The output of the LSTM layer is then flattened and passed through the first fully connected layer with ReLU activation function applied, and
- The output is then passed through second fully connected layer to produce the final output logits for classification.

The first convolutional layer takes a batch of inputs and then try to find any hidden features with the next two convolutional layers. Then we transpose the data and feed it to the LSTM layer to find any time dependent hidden feature among the sequences. I used *Adam* optimizer to optimize the *crossentropy* loss function and after each batch a back propagation was performed to retune the network weights.

In next section I will explain my approach for *Random Forest* model.

Algorithm 2: Random Forest

I also developed a generative *Random Forest* model to learn for classification of events from the re-sampled and redefined data. I used the *sklearn* package library to define a *Pipeline*, which takes the re-sampled data with each sequence of 10 signal amplitude window, standardize it and transform it to the PCA domain and then feed into the forest of 1000 trees for training.

Results

This section presents an analysis of the performance of the two approaches employed in the project.

For the CNN approach, two convolutional layers were initially employed, and subsequently, a third convolutional layer and an LSTM layer were introduced. However, the performance of the model did not show a significant improvement with the increased complexity. The objective was to progressively increase the number of layers until the model overfits the training set, but this proved to be computationally intensive. Therefore, the reported results are based on a model trained on a small subset of the training data, while the model continues to train on the complete dataset.

The implementation of the Random Forest approach was straightforward, as described in Algorithm 2. The model was initially trained with 100 trees, and subsequently with 1000 trees. As

Table I: Performance error rate

Dataset / **Algorithm**	Train	Dev
CNN with 2 convolutional layer	7.64%	40.86%
CNN with 3 convolutional layer and LSTM layer	7.45%	40.85%
Random Forest with 100 trees	1.05%	5.48%
Random Forest with 1000 trees	0.00%	5.45%

shown in Table I, the model trained with 1000 trees over-fits the training data, and there is no significant change in performance on the /dev dataset.

Conclusion

Undoubtedly, this project was computationally intensive, both in terms of compute memory and time requirements. Prior to this project, I had no experience working with medical EEG data. Therefore, this project served as a valuable opportunity for me to familiarize myself with medical data and its unique challenges.

One significant takeaway from this project was the ability to define a custom CNN model and to adjust the sizes of the hidden layers to suit the input requirements. Although I was unable to complete the training on the entire dataset due to memory limitations on my server, the models learned on a smaller subset still provided insight into their performance, which was relatively satisfactory.