ECE 8527 Final Exam: EEG Classification and Segmentation

James Brodovsky
Department of Mechanical Engineering, Temple University
jbrodovsky@temple.edu

Introduction: In this project we were tasked with examining data from an electroencephalogram (EEG). An electroencephalogram (EEG) is a non-invasive test that measures and records the electrical activity of the brain using electrodes placed on the scalp. The test is painless and does not involve any needles or injections. The electrical activity of the brain is produced by the communication between neurons, which generate small electrical currents. These currents can be detected by the electrodes on the scalp and amplified to create a visual representation of the brain's electrical activity. The electrodes are connected to an amplifier that records the electrical activity of the brain and displays it on a computer screen or printed paper. The test usually takes about 30 minutes to an hour to complete, and the patient may be asked to perform certain tasks, such as closing and opening their eyes or hyperventilating, to elicit specific patterns of brain activity. We were tasked with using machine learning techniques to identify such events. The dataset provided contained a one-dimensional signal in the form of an (amplitude, time) pair along with a set of event labels consisting of a start time, and end time, a duration, and an event classification (zero through four with a zero-label considered 'not an event' or background noise).

General Approach: Analyzing this problem, the best way to approach it appeared to be taking it in two phases: labeling the individual points and then clustering them together. With a set of (time, amplitude, class) points in a given cluster, it would be relatively easy to translate this into an event description that included the start and end times with a class label. As such, in the training phase the raw event descriptions in each data file were decoded and the class label was applied to each point in the timeseries data file. For prediction the reverse: the timeseries was first labeled point-by-point via the machine learning model, then segmented using a clustering algorithm, and finally encoded into an event description. The clustering and encoding method were the same throughout. I used DBSCAN to cluster the labeled points. From those clusters, the class label mode was used to classify the event and the lowest and highest time values were chosen.

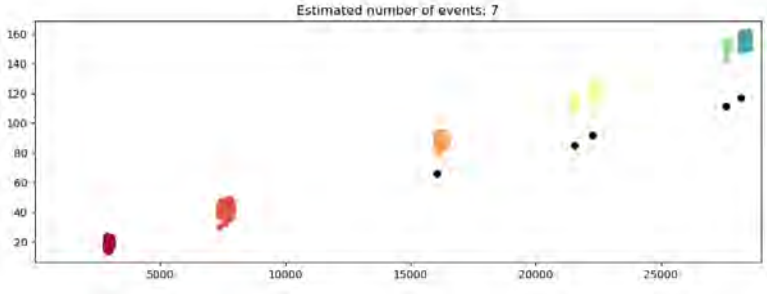

Figure 1: DBSCAN clustering example.

Classifier 1 – Random Forest: In examining the data I found two initial insights: first, that there was a clear linear trend to some of the amplitudes which did not appear to correlate with event labels. As such, since there did not appear to be any easily discernable pattern (as shown in Figure 1) I went with a Random Forest classifier as my first algorithm. Applying this to the whole dataset revealed the second problem: that there were significantly more data points labeled as 'background' than there were labeled events which gave a false impression of success for the classifier. To compensate for this, I 'normalized'

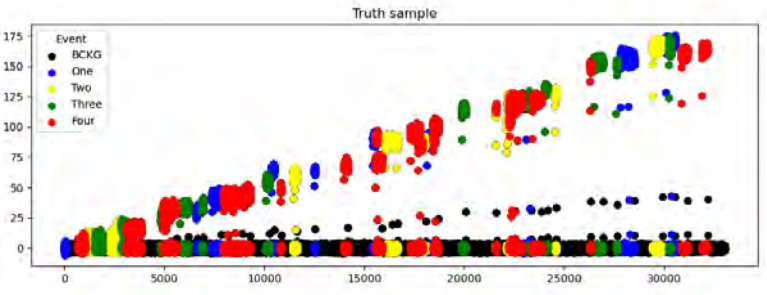

Figure 2: Sample of the training data set.

the dataset so that only one-fifth of the training data was background noise. I then retained the classifier on this normalized dataset using default random forest parameters from SciKit Learn's implementation. This classifier ended up simply memorizing the normalized training set. I modified the parameters to require a minimum leaf size of five and retrained the classifier. This yielded worse results on the normalized training set, but better results on the whole set.

Classifier 2 – CNN: My second algorithm swapped the Random Forest classifier for a Convolutional Neural Network. The network was relatively simple and two initial convolutional layers of 4 and 2 filters respectively followed by four linear hidden layers and output layer of five corresponding to the number of classes. Each of the hidden layers was activated using ReLU activation with a 25% dropout during training, with the output layer using a softmax activation. Again the dataset was normalized to contain roughly four-fifths data points corresponding to labeled classes and one-fifth background noise. This dataset was fed to the CNN using a PyTorch data loader which fed the timeseries to the CNN in windows. By feeding in the time series at a fixed window interval starting from the first index and proceeding to the last with padding on the end, I could use a typical CNN as effectively a timeseries classifier. I decided on 25 seconds after examining the duration of the events in the source files. The shortest event appeared to be approximately 100 seconds. By sampling the time series at a window of 25 I ensured that I would meet the Nyquist rate and should avoid accidentally aliasing the event signal. The model was trained using cross entropy loss and an Adam optimizer.

Results: Accuracy was calculated using the provided scoring software. The overall success rate of the classifier was calculated using the true positive and false positive rates event types one through four and the results are summarized below in Table 1. Overall, the Random Forest Classifier performed well on both sets with a success rate of 77.04% and 82.77% on the training and development sets respectively. Both results are statistically significant, with z-scores of 86.21 and 44.96.

The convolutional neural network, however, failed to train. There are two likely causes to this. First, the network was being trained on all classes, including the background noise. A five-class cross entropy loss, in theory, should be able to recognize the background noise as class zero if trained on the other four classes. Second, and more importantly, the network was being trained on periodic chunks of the data where the chunk was labeled as the most frequent class in the set. I intended for the size of the sample to only encompass one event class (or only have a small minority of other neighboring events) but it appears that a better route would have been to instead feed the network the actual segmented events.

Conclusions: In this project we examined an EEG data set that associated events occurring within it. I successfully developed a Random Forest and DBSCAN based machine learning implementation that produced reliable results on the dataset. My second method swapped the Random Forest for a Convolutional Neural Network classifier. This implementation failed to successfully train, likely due to not being supplied enough data.

	Data Set		
Algorithm	**Train**	**Dev Test**	**Eval**
Random Forest + DBSCAN	77.04%	82.77%	TBD
CNN + DBSCAN	N/A	N/A	TBD

Table 1. Results summary, where percentages are the difference of means between the true positive rate and false positive rates.