Real Time Classification of Eye Open-Closed State with Machine Learning

Andrew Pesek

Abstract

As more research has been conducted looking into the applications of Brain-Computer Interface (BCI) technology, there has never been a greater need for computational methods with the ability to extract meaningful information from brain signals. BCI technology requires some computational process in order to interpret information meant to inform the function of a designed BCI implementation. Although frameworks do already exist that can extract meaningful information from and/or classify brain wave data, many of these algorithms have drawbacks that may not allow them to be applied in real time. For instance, such algorithms can only be applied after recordings are taken, or require a large sample of recordings; they may include non-causal filters; or they may simply take too long to run in real time. The development of real time classification algorithms is pivotal in advancing the overall quality, functionality and utility of BCI implementations.

1. Introduction

In this report, I will look at a number of the many possible machine learning algorithms that can be used to make simple classifications of brain wave EEG data in real time. The key for any given method to be operational in real time is that only prior data can be used to make a classification at the present time point (causal). Any filters or structuring of the data that would affect it temporally (non-causally) must be minimal, such that classification at any point in time would only incur a small amount of latency, or is otherwise non-existent.

2. Methods

For the data set I will be using a continuous recording of brain wave data taken by an Emotiv EEG Neuroheadset with 14 spatial channels. The recording is taken over a period of roughly 117 seconds with a 128 Hz sampling rate. Alongside each datapoint in the recording there is a label corresponding to whether the individual's eyes were open or closed, denoted by 0 or 1 respectively. (Citation and accreditation in references)

In terms of pre-processing, I first removed a nominal amount of extreme outlier values from the data. The data was also filtered for 50 Hz frequency contamination.

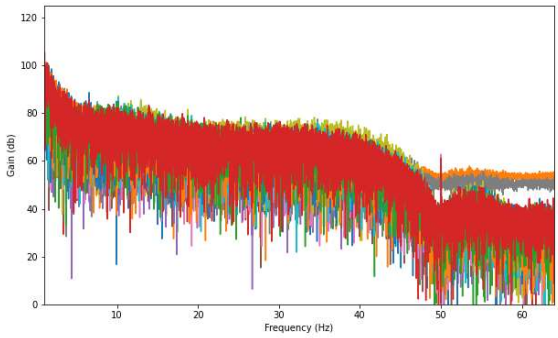

Figure 1. Frequency power graph of raw data before filtering. Spike present at 50 Hz

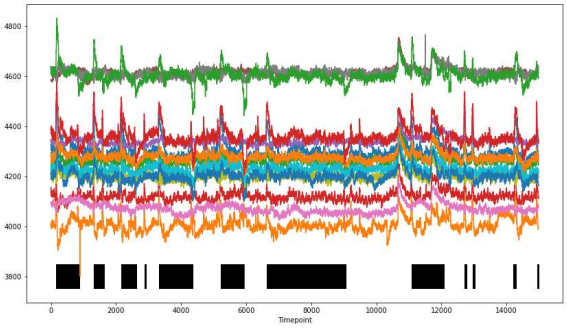

Figure 2. Filtered EEG data, bottom shaded portion shows timepoints where eye-state = 1 (closed)

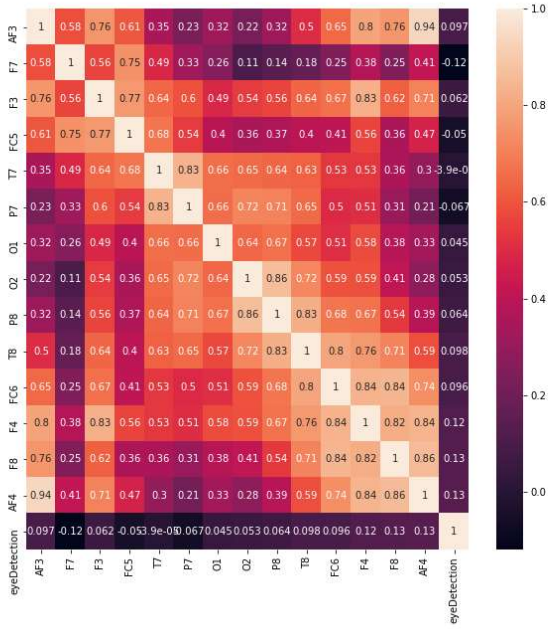

Figure 3. Correlations between each variable. Note that eye detection (eye-state) has very weak correlation with each channel

From here I restructured the data in such a way that a large degree of data can be accounted for at any point in time the algorithm tries to predict the eye state. For each eye state label spanning across the data, all recordings from the previous 128 points (1 second) are represented. Additionally, the average eye state prediction over the last 128 points is appended onto each datapoint. This makes the total dimensionality of each datapoint $14 \times 128 + 1 = 1793$.

In training I will split the data for training at the first 3000, 5000 and 7000 data points. Respectively, these split cutoffs correspond to the initial 23.438 seconds (~20%), 39.063 seconds (~33%) and 54.688 seconds (~47%) of the recording data. The rest of the data is then used to test model accuracy. For the data in the training sample, the truth-value labels are used to determine the rolling averages appended to each data point. Afterwhich, the data will be fed into the trained models sequentially, and only the previous predicted labels will be used to determine the rolling average values.

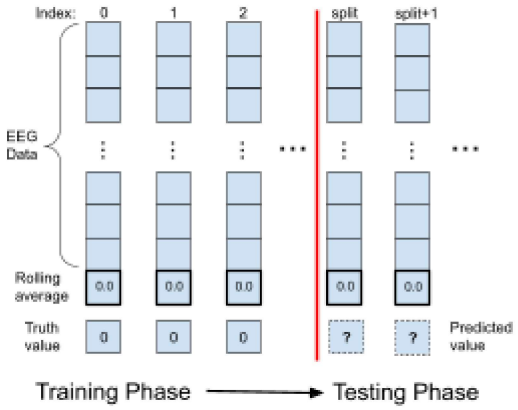

Figure 4. Recursively updating the input data to assist model accuracy

$$\text{Rolling average at } n = \frac{1}{128}\sum_{i=1}^{128}\begin{cases}\mathbf{y}_{n-i}, & n-i < \text{split} \\ \hat{\mathbf{y}}_{n-i}, & n-i \geq \text{split}\end{cases}$$

Here, $\mathbf{y}$ denotes the truth-value labels and $\hat{\mathbf{y}}$ denotes the predicted labels. For comparison, each classification paradigm will be tested on the data without the rolling average (previous prediction) variable.

For the machine learning algorithms I will test the following: Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Support Vector Classifier (SVC) and Random Forests.

3. Results

Shown in each of the following tables for each model trained and tested are the split timepoints between the training and testing data, the testing accuracy using the data with the rolling averages of previous predictions and testing accuracy without rolling

average predictions, as well as any hyperparameters specific to the model.

3.1. Linear Discriminant Analysis

split	accuracy (w/ preds)	accuracy (w/o preds)
3000	0.869558	0.496372
5000	0.932298	0.385099
7000	0.959373	0.376210

Table 1. Testing accuracy with LDA

Overall, LDA showed a high degree of accuracy classifying the data with the added rolling average prediction variable appended to the data. Simply by using the first 3000 timepoints in the data, LDA was able to accurately classify about 87% of the rest of the data. Intuitively, this is promising because it means that some BCI implementation using EEG signals to operate could rather quickly be calibrated by the operator for instance, through some means manually feeding it 'truth-values' so that the model can be trained. This is great for two reasons, as mentioned before, 3000 samples takes about 23 seconds to record, during this time the operator can repeatedly open and close his/her eyes so that both classes are represented roughly equally; and also LDA has a relatively short overhead training time (in my case using python Jupyter notebook, it took no longer than about 5-10 seconds.)

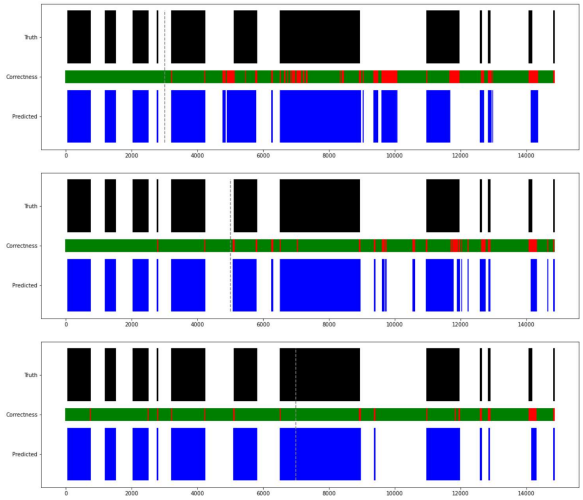

Figure 5. Model: LDA on data with prediction variable. Visual illustration over time comparing truth-value labels (black) and predicted labels (blue). The shaded black/blue represents eye-state = 1 (closed)

I used these illustrations to graphically represent overtime the predictions each model produced and compared them with the truth-values. There is a grey dashed line in each graph showing where the split cut-off was. The red regions in the middle highlighting incorrect predictions are somewhat visually overrepresented because they were drawn in matplotlib after the green region. In any case, using these illustrations I was able to get an understanding of how each model was classifying the testing data beyond a simple accuracy percentage. As shown, the LDA model in this case was clearly able to distinguish between regions of time where the eyes were closed vs. open, with only a couple odd-segments misclassified or error around points where the eye-state transitioned.

3.2. Quadratic Discriminant Analysis

split	accuracy (w/ preds)	accuracy (w/o preds)
3000	0.444566	0.444482
5000	0.644336	0.618352
7000	0.660341	0.497580

Table 2. Testing accuracy with QDA

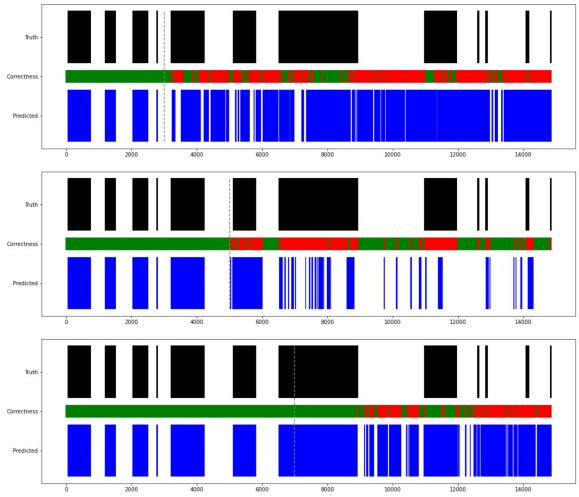

Figure 6. Model: QDA on data with prediction variable

In comparison to LDA, QDA performed much worse, and without a significant difference in accuracy

between the data with and without prediction variables. As shown in the illustrations, it seemed to over classify samples as '1' in the 3000 and 7000 splits, and as '0' in the 5000 split. This is likely due to the fact that the class labels in the training sets are not perfectly balanced, and it is possible that the model may have a tendency to 'over-learn' one class over the other.

3.3. Support Vector Classifier

split	C	accuracy (w/ preds)	accuracy (w/o preds)
3000	0.5	0.540162	0.605636
3000	1.0	0.486247	0.526325
3000	5.0	0.494684	0.624789
5000	0.5	0.456557	0.438490
5000	1.0	0.456354	0.476959
5000	5.0	0.576431	0.515631
7000	0.5	0.607998	0.395059
7000	1.0	0.450331	0.460010
7000	5.0	0.610800	0.610036

Table 3. Testing accuracy with SVC

The support vector classifier had a similar issue to the one QDA had, where it tended to over-learn one class over the other depending on the split and other hyperparameters. The prediction variable in most cases did not help the accuracy of the model and made it worse more often than not.

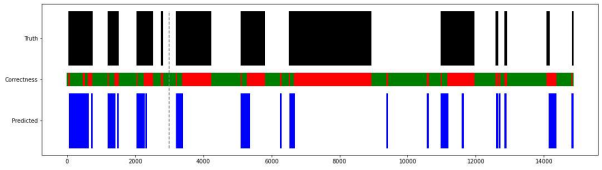

Figure 6. Model: SVC on data without prediction variable, split at 3000, C = 0.5

Interestingly, in some cases SVC was able to predictly label the first few seconds of timepoints at the onset of a transition from 0 to 1. This would likely be because the spikes in the EEG data at these transitions were prominent enough that the model could learn them. However, this pattern was too inconsistent across different models tested in order to be reliable.

3.4. Random Forests

split	depth	accuracy (w/ preds)	accuracy (w/o preds)
3000	4	0.694735	0.675667
3000	6	0.678535	0.670013
3000	8	0.692879	0.658876
3000	10	0.677185	0.661829
5000	4	0.611246	0.596123
5000	6	0.633780	0.599066
5000	8	0.641494	0.588408
5000	10	0.627588	0.584044
7000	4	0.660341	0.646205
7000	6	0.660469	0.628757
7000	8	0.662761	0.631559
7000	10	0.672822	0.625828

Table 4. Testing accuracy with random forests

Random forests performed roughly the same across the board, with the prediction variable just slightly improving accuracy. Although the best models tested achieved about 67-70% testing accuracy, this is atleast better than QDA and SVC.

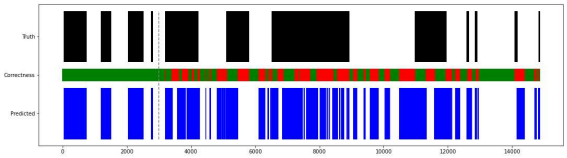

Figure 7. Model: Random forests on data with prediction variable, split at 3000, max depth = 10

Of note is that the density of predictions for each class seems to roughly correlate with the regions for the truth values. For instance predicted values of '1' seem to at least show up more often around regions where the truth-value is '1'. Therefore, with a bit more fine tuning and potentially pooling together many predictions over intervals of time, this model may be more accurate in practice.

4. Conclusions

Among the different classifiers tested, LDA performed the best especially in the case where the model used it's previous predictions as input for successive classifications along each timepoint. Furthermore, given that the classifier only required a

minimal amount of samples to achieve reasonable accuracy, and that the overhead training time was comparatively short, LDA would make the most amount of sense to implement within a BCI implementation that could theoretically be calibrated in real time and utilized asynchronously. Otherwise, it could perhaps be utilized as an activation for a synchronous BCI implement.

Out of the other classifiers tested, SVC could potentially be utilized in conjunction with a better model given its ability to accurately detect the onset of the eye being closed. However, it was difficult given the structuring of the data to design a model that could reliably do this. Additionally, SVC takes a longer amount of time to train, so it may not be the most reasonable model to be implemented. As for random forests, there may be some utility within a potential ability to predict eye-state over much longer intervals of time (within the range of multiple seconds), this would require additional tuning.

Lastly, I would argue that QDA perhaps was too sensitive to variation in the data to be able to accurate most of the time. Given if the classes were more balanced and the data more consistent, QDA could potentially beat-out LDA in terms of accuracy. However, QDA took much longer to train so given that fact and it's overall sensitivity, it would not be a reliable model to implement.

References

O. Roesler, it12148 '@' lehre.dhbw-stuttgart.de, Baden-Wuerttemberg Cooperative State University (DHBW), Stuttgart, Germany, https://archive.ics.uci.edu/ml/datasets/EEG+Eye+State

V. J. Lawhern, A. J. Solon, N. R. Waytowich, S. M. Gordon, C. P. Hung, B. J. Lance, "EEGNet: A Compact Convolutional Neural Network for EEG-based Brain-Computer Interfaces," *arXiv*, arXiv:1611.08024v4 [cs.LG] 16 May, 2018

Inspiration from Student Projects:

"Feature Visualization and Cross-Subject Training of Convolutional Neural Network for Motor Imagery," https://drive.google.com/file/d/1I-kKjwpKVQr9f2ofr2YkuuOL_CqfQ7b9/view

"Classification of EEG Seizure Data," https://docs.google.com/document/d/10_EhIT5M36oL-kZGWxZ2TkYo7SSyxlOyQphSvFEWyGk/edit?usp=sharing

"Cogs 189 Final Project," https://github.com/rwegznek/Cogs-189-Final-Project