OpenViBE Architectural Changes to Take Advantage of Full Range of Python ML Modules and Reduce BCI Development Time

Contents

Abstract

OpenViBE (OV) is an Open Source Brain Computer Interface (BCI) platform which facilitate researchers to capture the electroencephalogram (EEG) signals, to process them using its signal processing modules, to find patterns in them using its Machine Learning (ML) modules and predict the state of the brain according to the identified patterns (for a given scenario). OV is designed for programmers as well as for people who are not familiar with programming. OV's signal processing and machine learning modules are developed in C++. OV provides *Python Scripting* plugin to execute python scripts whenever required. It also provides a plugin to execute *scikit-learn* library (one of the most useful libraries) of python which allows machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction. This python library does not support deep learning. However, since OV does not allow using other powerful Python libraries except the above, firstly, this paper focuses on implementing a generic workflow to allow using full range of Python ML libraries (including *scikit-learn*). Also, presently OV workflow combines the efforts of Pre-Processing, Feature Extraction and Model Generation in a single workflow. That forces the researcher to repeat Pre-Processing and Feature Extraction steps even though he/she just wants to choose a different model. Hence secondly, this paper focuses on de-coupling Pre-Processing and Feature Extraction (mainly signal processing) steps from Model Generation. That helps researcher save the time and efforts of repeating *signal processing* steps, since they need to be executed only once for different

models. The architectural changes which are mentioned in this paper are predominantly for researchers with programming background.

Introduction

This paper is divided into two main sections. In the first section, this paper explains OV architectural changes in order to take advantage of full range of python machine learning algorithms. In the second section, it explains architectural changes to de-couple signal processing from ML. This paper demonstrates the architectural changes, with an example of P300 Speller scenario.

Architectural Changes in OV to Support Python ML

This section walks through the architectural changes made for supporting full range of Python ML Libraries. We made the below changes:

1. Modified the OV Training Workflow: This change is done to save the signal data into database. In our case we are saving the stimulation epoch data into a Comma Separated Values (csv) format.
2. Finalized on CSV File Format: We had to brain-storm for saving the epoch data collected across channels and across time-scale into csv. We needed to save the corresponding labels too. We came up with a file format which can be generic to be used across different Python libraries.
3. Implemented the Machine Learning Script Using Python and the CSV: We fed the CSV file to python ML libraries (we tried with Scikit-Learn and TensorFlow libraries) and generated the models.
4. Modified the Prediction Workflow: We used the model which is generated in the above step, for Prediction (which is an online scenario with unseen data).

Each of the above changes are explained in detail in the below sub-sections.

Modify OV Training Workflow

Our first change is to modify the OV training workflow, such that the data for each epoch (target as well as non-target) can be saved to a csv file with appropriate labels. The below figure shows the original OV workflow on the left side and the current workflow on the right. We have removed *feature extraction (removal is optional)*, *feature aggregation* (*removal is optional*) and *classifier trainer* (*must be removed*) boxes. The removal of feature extraction and feature aggregation is optional because keeping it in OV, logically groups the entire signal processing into OV. The removal of classifier is mandatory because it keeps the machine learning out of the scope of OV. Then, we have added a *Python 3 Scripting* box. This scripting box will point to the script to save the data into a csv file. In summary, this architectural change keeps the signal processing with OV and hands over the ML part to python.

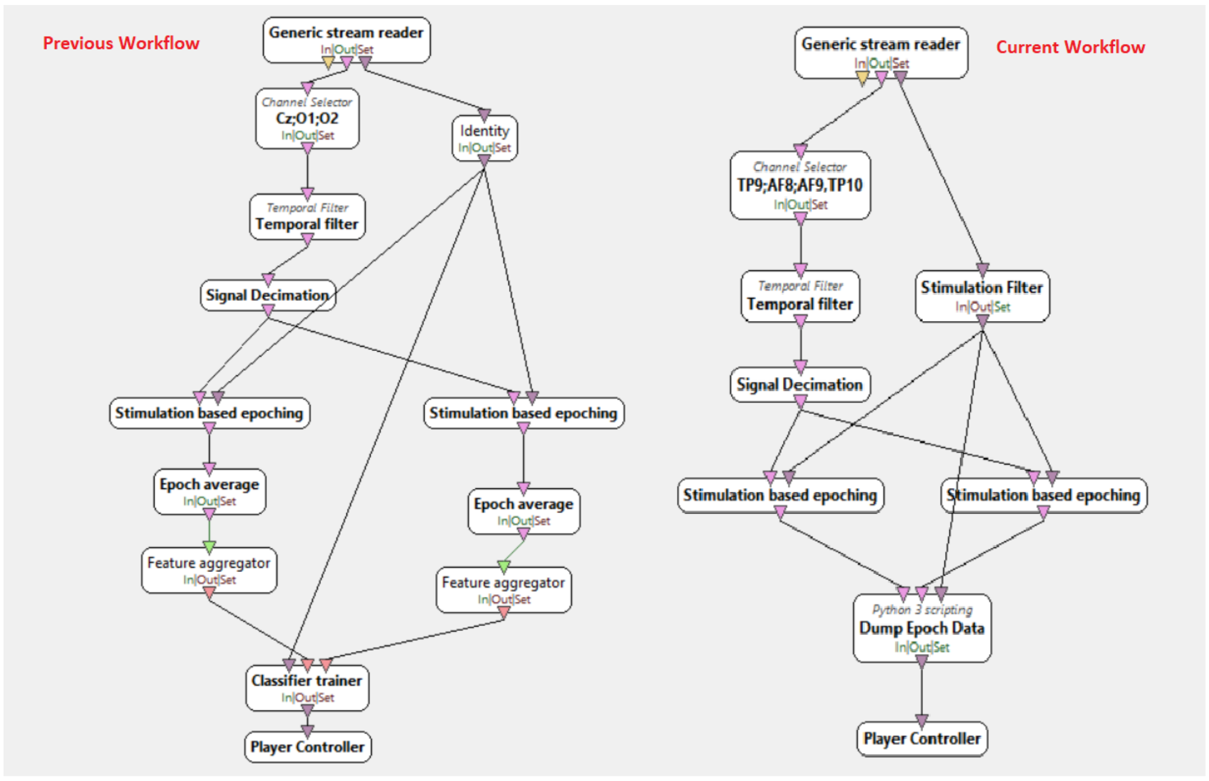

Figure 1 P300 Speller Training Scenario

Finalize on CSV File Format

Our second change is to decide upon csv file format. In our csv file, each row represents combined epoch data across all the channels for a stimulation at time t1. The last column of the spreadsheet, provides labels across each epoch as target (1) or non-target (0). The subsequent rows represent the data at time t2, t3, t4 and so on.

	Epoch Samples for Channel 1					Epoch Samples for Channel 2					Epoch Samples for Channel 3					Label
t1	C1.1	C1.2	C1.3	C1.4	C1.5	C2.1	C2.2	C2.3	C2.4	C2.5	C3.1	C3.2	C3.3	C3.4	C3.5	0
t2	C1.1	C1.2	C1.3	C1.4	C1.5	C2.1	C2.2	C2.3	C2.4	C2.5	C3.1	C3.2	C3.3	C3.4	C3.5	1
t3	C1.1	C1.2	C1.3	C1.4	C1.5	C2.1	C2.2	C2.3	C2.4	C2.5	C3.1	C3.2	C3.3	C3.4	C3.5	1
t4	C1.1	C1.2	C1.3	C1.4	C1.5	C2.1	C2.2	C2.3	C2.4	C2.5	C3.1	C3.2	C3.3	C3.4	C3.5	0

This data can be fetched into a python tensor using the following function:

```
X = loadtxt('c:\\eeg\\whole_data.csv', delimiter=',')
```

Once, the data is fetched into a tensor, it can be reshaped into any other form as required by the classifier. A couple of examples are below.

```
X = X.reshape(len(X), 3, 5)
```

Or

```
X = X.transpose(0, 2, 1)
```

Implement the Machine Learning Script Using Python and the CSV Input

Our third change is to implement the Python machine learning script with CSV as input, in order to create a model. We could implement a python script with *TensorFlow library*. This script works outside of OV (offline) to generate a model. The script performs data augmentation, data balancing, data normalization and classification using python APIs for pre-processing and machine learning. We used TensorFlow's CNN1D model to classify the target and non-target epochs. The input to the model is the data in the csv file mentioned above. The output is a model in .h5 format. In this script, we calculate the confusion matrix of the validation data and display it on the screen for evaluation of the model. We also plot the training and validation graphs to give us an idea if the model converges well.

Modify the Prediction Workflow

As our fourth change, we needed to modify the prediction workflow of the P300 Speller to take advantages of Python. Below figure shows the original OV prediction workflow of the speller on the left side. The right side shows the modified prediction workflow. As we can see, we have removed the feature extractor (optional to remove), feature aggregator (optional to remove) and classifier processor (mandatory to remove) boxes. The classifier processor box has to be removed and replaced by the *Python 3 Scripting* box, which points to the script which loads the Python model, do the pre-processing (in our case data normalization and re-shaping) of un-seen data, feed this data to Python model for prediction and get the prediction results. In P300 Speller example, these predictions are processed by the Visualization module and the targeted character is displayed on the screen.

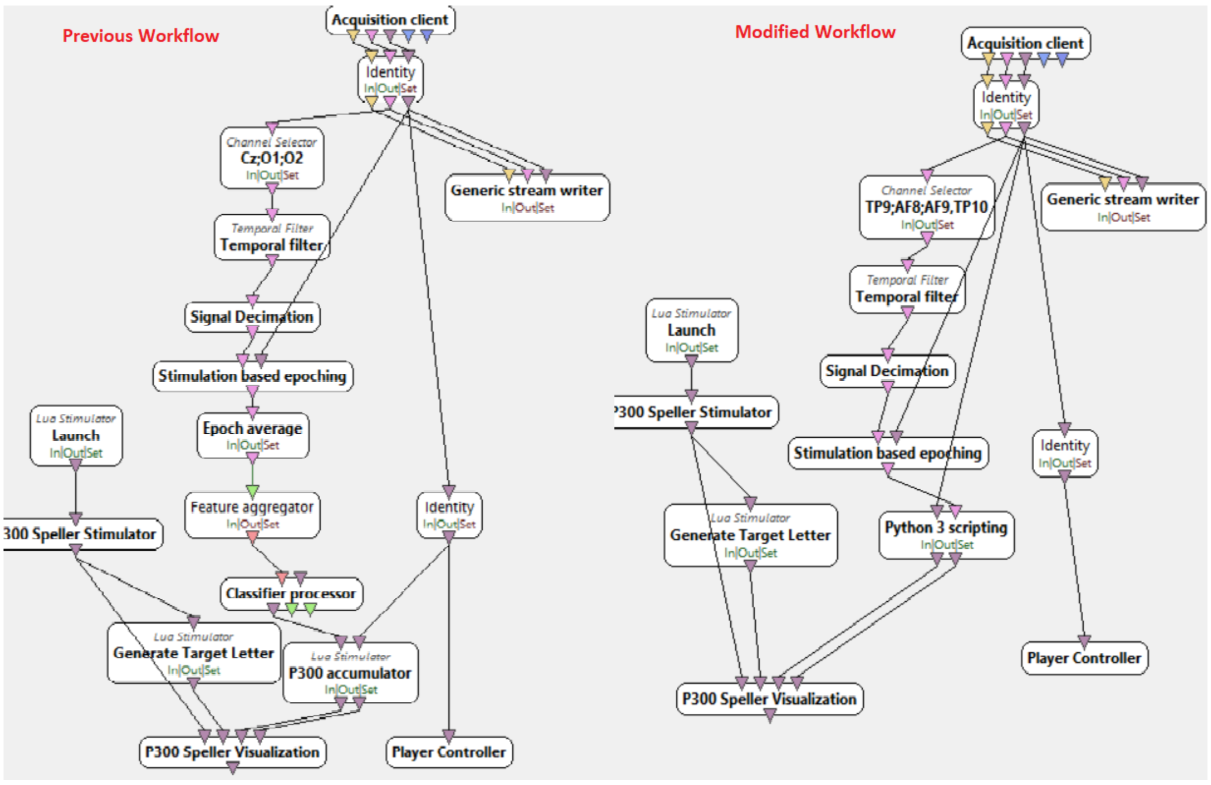

Figure 2 P300 Speller Online Scenario

Architectural Changes to De-couple Signal Processing from Machine Learning

As we can see from Figure 1 and Figure 2, in the original workflow has Signal Processing and Machine Learning combined in a single workflow. During the experiments we realized that OV signal processing takes longer than machine learning. For Example: A 50 MB signal file takes more than 1 hour to do feature extraction and feature aggregation. Whereas machine learning takes less than 10 min (Given the three classifiers LDA, MLP and SVM which OV has supported). If we want to choose a different ML model, the previous workflow forces us to repeat the time and efforts required for signal processing as well as model generation both. However, with the introduction of the modified workflow, the signal processing is done and the features gets saved to a CSV file *only once*. We can then apply different Python ML models on to these extracted features, offline. This saves a lot of time and efforts if the requirement is to change the model.

Original Training Workflow

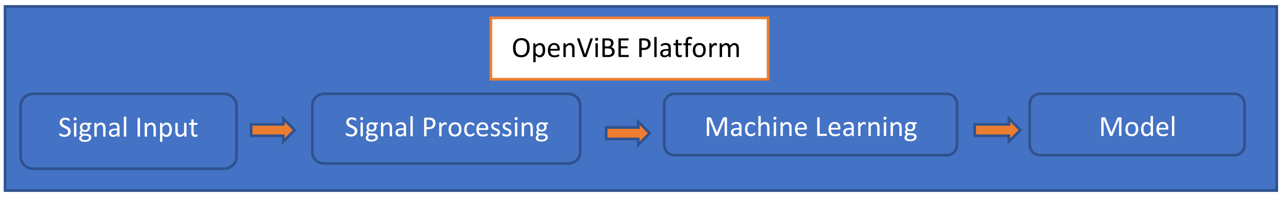

Figure 3 Original Training Workflow in OV

Modified Training Workflow

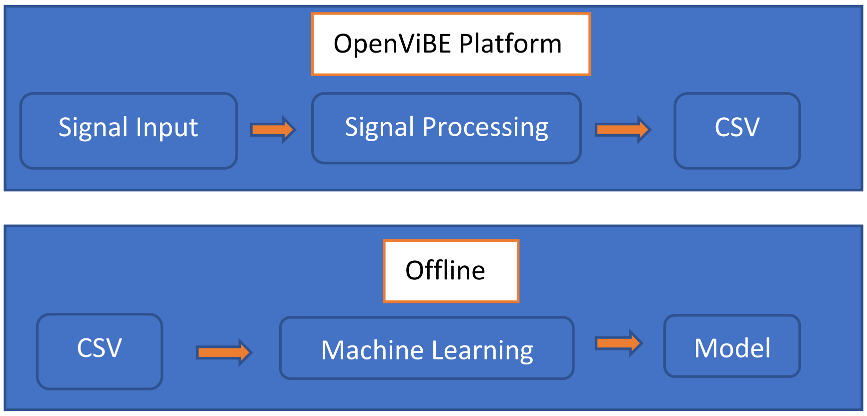

Figure 4 Modified Training Workflow

Advantages of De-coupling Signal Processing from Machine Learning

1. Two different tools can be used: One tool for signal processing and another for ML
2. Time and Efforts can be saved in case the requirement is only to try different ML model
3. Special skills in each of the areas (signal processing and ML) can be learned separately

Conclusion

This paper explains the modifications in OV architecture for two different purposes: One, to use any of the Python machine learning libraries for signal classification and Two, de-couple signal processing and machine learning. It also explains the csv file format which is representative format for stimulation-based application. This file format is a glue to the OV specific processes and the Offline processes. At the end it explains the advantages of architectural modifications.